On Constant Factors in Comparison-Based Geometric Algorithms and Data Structures

by

Patrick Lee

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Many standard problems in computational geometry have been solved asymptotically optimally as far as comparison-based algorithms are concerned, but there has been little work focusing on improving the constant factors hidden in big-Oh bounds on the number of comparisons needed. In this thesis, we consider orthogonal-type problems and present a number of results that achieve optimality in the constant factors of the leading terms, including:

- An output-sensitive algorithm that computes the maxima for a set of n points in two dimensions using $1n \log h + O(n\sqrt{\log h})$ comparisons, where h is the size of the output.
- A randomized algorithm that computes the maxima in three dimensions that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.
- A randomized output-sensitive algorithm that computes the maxima in three dimensions that uses $1n \log h + O(n \log^{\frac{2}{3}} h)$ expected number of comparisons, where h is the size of the output.
- An output-sensitive algorithm that computes the convex hull for a set of n points in two dimensions using $1n \log h + O(n\sqrt{\log h})$ comparisons and $O(n\sqrt{\log h})$ sidedness tests, where h is the size of the output.
- A randomized algorithm for detecting whether of a set of n horizontal and vertical line segments in the plane intersect that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.
- A data structure for point location among n axis-aligned disjoint boxes in three dimensions that answers queries using at most $\frac{3}{2} \log n + O(\log \log n)$ comparisons. The data structure can be extended to higher dimensions and uses at most $\frac{d}{2} \log n + O(\log \log n)$ comparisons.
- A data structure for point location among n axis-aligned disjoint boxes that form a space-filling subdivision in three dimensions that answers queries using at most $\frac{4}{3} \log n + O(\sqrt{\log n})$ comparisons. The data structure can be extended to higher dimensions and uses at most $\frac{d+1}{3} \log n + O(\sqrt{\log n})$ comparisons.

Our algorithms and data structures use a variety of techniques, including Seidel and Adamy's planar point location method, weighted binary search, and height-optimal BSP trees.

Acknowledgements

I would like to thank my supervisor, Timothy M. Chan, for his guidance. Without him, this thesis would not have been possible. I also thank my thesis readers Anna Lubiw and Therese Biedl.

I would like to thanks my wife, Disney Yan Lam, for her love and support.

Dedication

This thesis is dedicated to Dismo, Patricia, and Dory.

Table of Contents

List of Figures

List of Theorems

Chapter 1

Introduction

Many classic problems in computational geometry have solutions that are asymptotically optimal; that is, these problems have an asymptotic upper bound that matches their asymptotic lower bound. For example, Chan's algorithm can compute a convex hull of n points in the plane, with output size h, in $O(n \log h)$ time [4] and Kirkpatrick and Seidel showed that $\Omega(n \log h)$ is a matching lower bound [41]. However, asymptotic optimality may not be true optimality, as asymptotic notation ignores the leading constant factors. This thesis presents several new algorithms that improve the leading constant factors for many classic problems in computational geometry.

Note that throughout this thesis all logarithms are in base 2.

1.1 Asymptotic Notation

Due to the difficulty of determining the cost of various primitive operations, it is often difficult to analyse the exact running time of algorithms. To simplify an analysis, constant factors are typically ignored; instead the focus is placed on the rate of growth. That is, if an algorithm's exact running time is $cf(n) + g(n)$, where c is a constant and $f(n)$ is of a higher order than $g(n)$, then the asymptotic running time is $\Theta(f(n))$. Instead of exact running time, asymptotic running time is typically studied.

Formally, we say $f(n) \in \Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$. If algorithm A's running time is asymptotically smaller than algorithm B's running time, we say A is asymptotically more efficient. If A and B are asymptotically equal, then we say they are asymptotically equally

efficient. However, asymptotic optimality does not mean exact optimality, even if A and B are asymptotically equal, their running times can still differ by constant factors. These constant factors are important in practice.

1.2 Counting Comparisons

Exact running time considers all operations and their relative cost as well. However, it is often too difficult to count all operations and to determine their relative cost. We simplify this by only counting comparisons and ignore all other operations. This is better known as the comparison model of computation.

1.3 Motivation

Improving the constant factors is important in practice. The number of comparisons needed to solve various classic geometric problems is a very natural question and can be studied from a theoretical perspective. The comparison model has long been recognized as a fruitful model. This study has lead to a deeper understanding of the inherent complexities of these problems, and of the relative power of different algorithmic techniques.

We do not claim practicality of our new results, as the cost of the other operations may be much larger. However, in all of our new results, the number of other operations is asymptotically equal to the number of comparisons. Many of our new data structures have a simple enough query algorithm so that the number of comparisons is a good indicator of the actual running time.

1.4 Summary of Our Results

This thesis presents many new results; we split the result into two categories: 1. Algorithms and 2. Point location data structures.

The new algorithms are summarised below:

- We show how to compute the maxima for a set of n points in two dimensions with an output-sensitive algorithm that uses $1n \log h + O(n\sqrt{\log h})$ comparisons, where h is

the size of the maxima. In contrast, the trivial algorithm uses $1n \log n + O(n)$ comparisons, and Kirkpatrick and Seidel's output-sensitive algorithm [40] uses $\approx 3.45n \log h$ comparisons. Refer to Section 3.1 for the definition of the maxima problem.

- We give a randomized algorithm to compute the maxima in three dimensions. The naive algorithm uses at most $3n \log n + O(n)$ comparisons, a slightly better algorithm uses at most $2n \log n + O(n)$ comparisons. Our new randomized algorithm uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.

- We give a randomized output-sensitive algorithm to compute the maxima in three dimensions that uses $1n \log h + O(n \log^{\frac{2}{3}} h)$ expected number of comparisons, where h is the size of the output.

- By adapting our algorithm for maxima in two dimensions, we give a new output-sensitive algorithm for computing the convex hull for a set of n points in two dimensions. Graham's scan [35] uses $1n \log n + O(n)$ comparisons and $O(n)$ sidedness tests; a naive implementation of Chan's algorithm [4] uses $\approx 12n \log h + O(n \log \log h)$ comparisons and sidedness tests (see Section 3.3). Our new algorithm is output-sensitive and uses $1n \log h + O(n\sqrt{\log h})$ comparisons and $O(n\sqrt{\log h})$ sidedness tests, where h is the size of the output. Refer to Section 3.3 for the definition of the convex hull problem.

- By adapting our algorithm for maxima in three dimensions, we give a new randomized algorithm for computing the convex hull of n points in three dimensions. Our new algorithm uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.

- We give a new randomized algorithm to compute the vertical decomposition of a set of n horizontal line segments in two dimensions. The trivial solution uses $3n \log n + O(n)$ comparisons, our new randomized algorithm uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons. Refer to Section 3.5 for the definition of a vertical decomposition.

- We give a new randomized algorithm for detecting whether of a set of n horizontal and vertical line segments in two dimensions intersect. The trivial solution uses $3n \log n + O(n)$ comparisons and the standard sweep line algorithm uses $2.5n \log n + O(n)$ comparisons. Our new randomized algorithm uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons. Refer to Section 3.6.

We give new data structures for point location. We examine the special case where the objects are axis-aligned disjoint boxes:

- We examine the problem of point location among n axis-aligned disjoint boxes in three dimensions. The trivial data structure uses $O(n^3)$ space and $3\log n + O(1)$ comparisons to answer a query. We present a new data structure that answers queries using at most $\frac{3}{2}\log n + O(\log\log n)$ comparisons, and uses $O(n^{\frac{3}{2}})$ space. In d dimensions, the trivial solution uses $O(n^d)$ space and $d\log n + O(d)$ comparisons to answer a query. Our data structure can be extended to higher dimensions, it answers queries using at most $\frac{d}{2}\log n + O(\log\log n)$ comparisons, and uses $O(n^{\frac{d}{2}})$ space.

- We examine the problem of point location among n axis-aligned disjoint boxes in three dimensions that form a space-filling subdivision. We present a new data structure that answers queries using at most $\frac{4}{3}\log n + O(\sqrt{\log n})$ comparisons, and uses $O(n^{\frac{4}{3}})$ space. Our data structure can be extended to higher dimensions, it answers queries using at most $\frac{d+1}{3}\log n + O(\sqrt{\log n})$ comparisons, and uses $O(n^{\frac{d+1}{3}})$ space.

1.5 Outline

The rest of the thesis is organized as follows. In Chapter 2, we present various related works. In Chapter 3, we describe our new algorithm results. In Chapter 4, we describe our new point location data structures. We conclude in Chapter 5 with some open problems.

Chapter 2

Previous Work

2.1 Previous Work in the Comparison Model

Tight asymptotic bounds for many classical problems in algorithms and data structures have long been known. The exact number of comparisons needed in the worst case are also known. Some results are:

- To find the largest (or smallest) element from a list of n numbers requires exactly $n-1$ comparisons.
- To find both the largest and the smallest element from a list of n numbers requires exactly $\lceil 1.5n \rceil - 2$ comparisons [53].
- Merging two sorted lists, each of size n, requires exactly $2n-1$ comparisons [60].

The constant factor in the leading term for the number of comparisons needed has also been studied. Some results are:

- Sorting a list of n numbers requires $1n \log n - \Theta(n)$ comparisons; this bound is tight [42, 29].
- To find the median element of a list of n numbers, the best known upper bound is $2.95n$ comparisons [23] and the best lower bound is $(2+\epsilon)n$ comparisons for some $\epsilon > 0$ [24].

- Randomized algorithms for finding the median use $1.5n \pm o(n)$ expected comparisons; this bound is tight [21].
- For the problem of building a heap from a list of n numbers, the best upper bound is $1.625n + O(\log n \log^* n)$ comparison [33] and the best lower bound is $1.37n$ comparisons [44].

2.2 Previous Work in Computational Geometry

Not as many problems in computational geometry have been examined in the comparison model. We briefly review three notable results.

Dubé's One-Reporting Range Query

Thomas Dubé presented a data structure that answers one-reporting dominance range queries [25]. A one-reporting dominance range query is a special case of orthogonal range reporting; given a query point q, we only need to return one point that is dominated by q; refer to Section 3.1 for the definition of 'dominance'. His query algorithm uses $\lfloor \frac{d}{2} \rfloor \log n$ comparisons (he didn't state the lower order terms) and his data structure uses $O(n^{\lfloor \frac{d}{2} \rfloor})$ space. Dubé showed a nearly matching lower bound on the query complexity of $\lfloor \frac{d}{2} \rfloor (\log n - \log \frac{d}{2})$ comparisons. By contrast, the trivial solution uses $d \log n + O(d)$ comparisons.

The Maxima Problem

The problem of computing the maxima for a set of n points in two dimensions has been well studied. In the worst case, the standard divide-and-conquer algorithm uses $O(n \log n)$ time. Kirkpatrick and Seidel gave an output-sensitive algorithm that uses $O(n \log h)$ time, where h is the output size [40]. Their algorithm is commonly known as marriage-before-conquest, and the constant factor is ≈ 3.45. In the special case where the points are uniformly distributed points within a square, Golin gave an algorithm that uses $1n + O(n^{\frac{6}{7}} \log^5 n)$ expected number of point comparisons [32]. Clarkson later improved the running time to $1n + O(n^{\frac{5}{8}})$ expected number of point comparisons [18]. A point comparisons compares both coordinates of two points at the same time using two coordinate comparisons.

Point Location

Seidel and Adamy showed that planar point location queries can be answered with a data structure that uses at most $1 \log n + 2\sqrt{\log n} + \frac{1}{2} \log \log n + O(1)$ sidedness tests

and $O(n2^{2\sqrt{\log n}})$ space [57]. They also showed a nearly matching lower bound of $1\log n + 2\sqrt{\log n} - \frac{1}{2}\log\log n - O(1)$ sidedness tests. They improved the space usage down to $O(n)$ with a only slight increase in the cost of a query; their new query algorithm uses $1\log n + 2\sqrt{\log n} + O(\log^{1/4} n)$ sidedness tests. By contrast, the trivial solution uses $2\log n + O(1)$ comparisons and $O(n^2)$ space. We discuss their technique in Section 4.1.

Chapter 3

Algorithms

3.1 The Maxima Problem in Two Dimensions

The problem of computing the maxima is a classic problem in computational geometry. This problem is closest related to the convex hull problem (see Section 3.3), and has many applications. In the database community, the maxima problem is commonly known as the 'skyline' problem.

For simplicity, we assume the points are in general position, that is, no two points share the same x-coordinate and no two points share the same y-coordinate.

Formally, we are given a set S of n points in the plane. We say a point $p = (p_1, \ldots, p_d)$ *dominates* point $q = (q_1, \ldots, q_d)$ if $p_j > q_j$ for all j. We say a point p is *maximal* if it is not dominated by any other point. The *maxima* problem is to report all maximal points. The maxima form a staircase structure; see Figure 3.1 for an example.

In this section we present two new output-sensitive algorithms for computing the maxima in two dimensions. Our first algorithm uses a top-down approach and our second algorithm uses a bottom-up approach.

The remainder of the section is as follows: First, we discuss some previous work. Second, we present Kirkpatrick and Seidel's algorithm and our new top-down algorithm which is based on Kirkpatrick and Seidel's approach. Third, we present our new bottom-up algorithm.

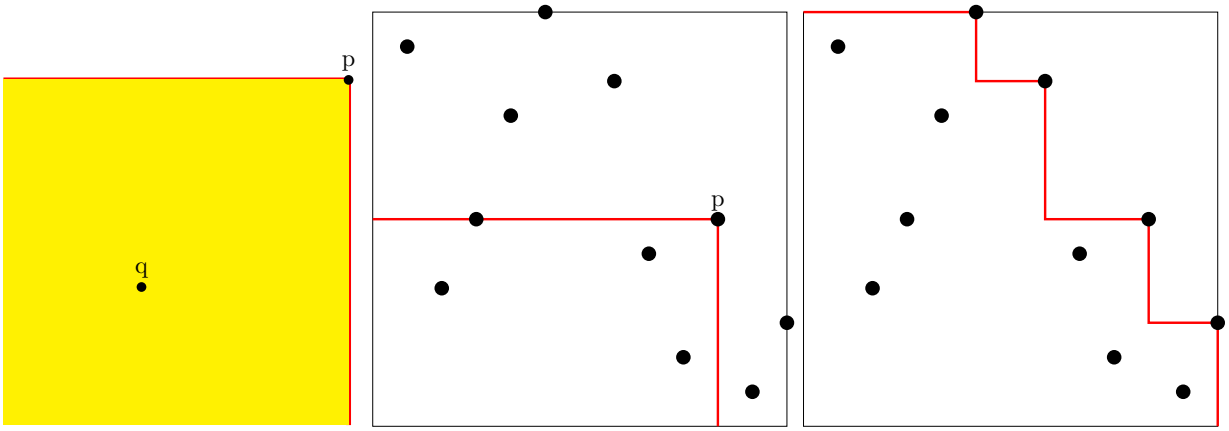

Figure 3.1: Illustrations of dominance (left), a maximal point (middle), and the maxima for a point set (right).

3.1.1 Previous Work

The two-dimensional case has been well studied. The brute-force solution is to compare every point against every other point, then output all maximal points. This uses $O(n^2)$ comparisons. A trivial solution is to sort all the points twice, once on their x-coordinates, and once on their y-coordinates. Afterward, no more coordinate comparisons are needed. This uses $2n \log n + O(n)$ comparisons.

A standard sweep line algorithm uses $1n \log n + O(n)$ comparisons, and works as follows:

Maxima(S)

```
1: sort the points by their x-coordinates in decreasing order
2: maxY ← the y-coordinate of the rightmost point
3: output the rightmost point
4: for every input point, S[i], in decreasing x-order do
5:     if maxY < S[i].y then
6:         maxY ← S[i].y
7:         output S[i]
8:     end if
9: end for
```

If the maximal points are required to be reported in sorted order, a reduction from the standard sorting lower bound implies a lower bound of $\lceil \log n! \rceil = 1n \log n - \Theta(n)$ comparisons. Otherwise, if the points are not required to be reported in sorted order,

Kung et. al. showed the lower bound is still $\lceil \log n! \rceil = 1n \log n - \Theta(n)$ comparisons using an information-theoretical argument [43]. Thus, in two dimensions, the constant factor is optimal.

The problem in three and higher dimensions has been studied as well. Kung, Luccio, and Preparata first gave a $O(n \log n)$ time algorithm for $d = 3$ and a $O\left(n \log^{d-2} n\right)$ time algorithm for $d \geq 4$ [43]. For $d = 4$, Gabow, Bentley and Tarjan gave a $O\left(n \log^{d-3} n \log \log n\right)$ time algorithm [31], Chan, Larsen, and Pătraşcu improved the time to $O\left(n \log^{d-3} n\right)$ [9]. If we only count comparisons, the trivial solution requires $dn \log n + O(dn)$ comparisons.

As mentioned in Section 2.2; in the special case where the points are uniformly distributed within a square, Golin gave a randomized algorithm that uses $1n + O\left(n^{\frac{6}{7}} \log^5 n\right)$ expected number of point comparisons [32]. Clarkson later improved the running time to $1n + O\left(n^{\frac{5}{8}}\right)$ expected number of point comparisons [18]. A point comparisons compares both coordinates of two points at the same time using two coordinate comparisons.

Kirkpatrick and Seidel gave an output-sensitive algorithm for maxima in two and three dimensions that takes $O(n \log h)$ time, where h is the output size. They gave a matching lower bound of $\Omega(n \log h)$, in an algebraic decision tree model [40]. They also gave an algorithm for higher dimensions that uses $O\left(n \log^{d-2} h\right)$ time. Afshani, Barbay and Chan gave an instance-optimal algorithm for maxima in two and three dimensions [1].

3.1.2 Kirkpatrick and Seidel's Output-Sensitive Algorithm

Our new algorithm for maxima in two dimensions is similar to Kirkpatrick and Seidel's [40]; hence, we present their algorithm in detail. Their algorithm is output-sensitive and takes $O(n \log h)$ time, where h is the output size. Their technique, known as 'marriage-before-conquest', is a top-down approach. First, it is determined how the sub-problems fit together (marriage), then the sub-problems are solved (conquest). Their algorithm is as follows:

KS-Maxima(S)

1: split S into two sets of equal size L, R, by the median x-coordinate
2: KS-Maxima(R)
3: prune(L, max $\}p.y \mid p \in R\}$)
4: KS-Maxima(L)

Prune(L,y)

1: **for** $p \in L$ **do**

2: **if** $p.y \leq y$ **then**
3: remove p from L
4: **end if**
5: **end for**

Analysis:

We inspect the recursion tree (see Figure 3.2). Let c_{med} denote the constant factor needed for median finding, $c_{\text{med}} \approx 2.95$. In each iteration, a sub-problem of size n uses $c_{\text{med}}n \approx 2.95n$ comparisons to find the median element and $\frac{1}{2}n$ to prune the left subset. Let $c = c_{\text{med}} + \frac{1}{2} \approx 3.45$. At level i, the size of each node is $\leq \frac{n}{2^i}$. The number of nodes at level i is bounded by 2^i and is also bounded by h, due to pruning.

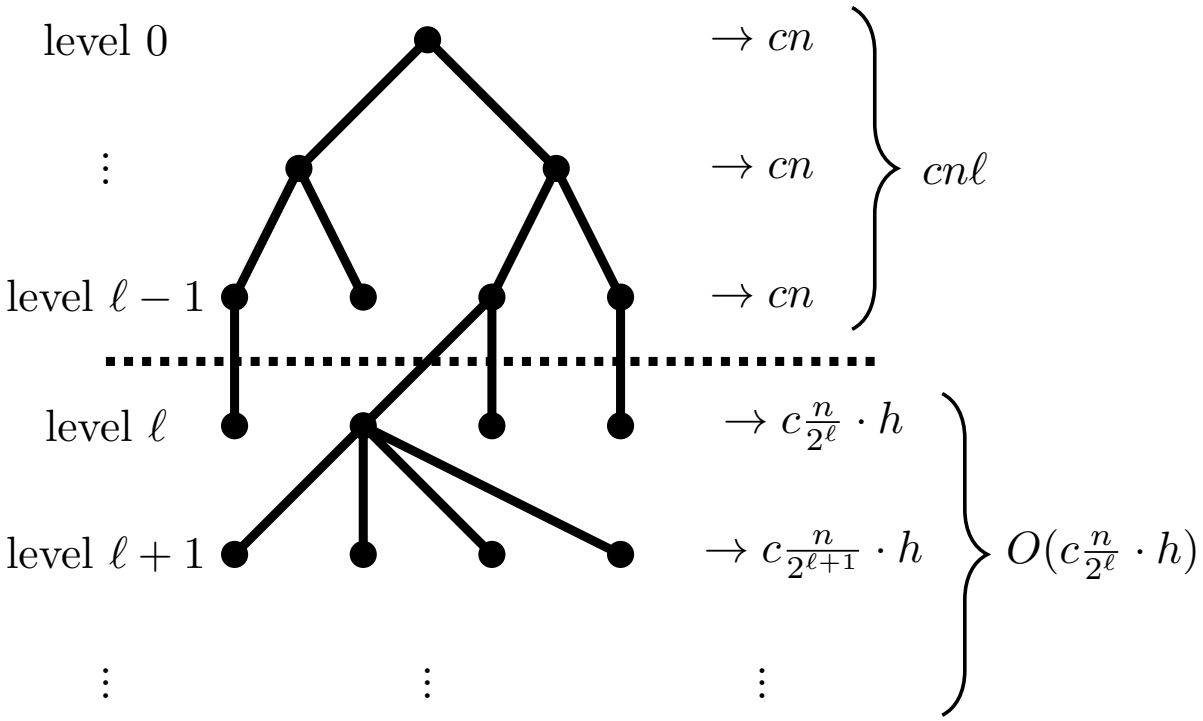

Figure 3.2: The recursion tree.

We split the recursion tree into the top ℓ levels and the bottom remaining levels. The top ℓ levels uses $\sum_{i=0}^{\ell-1} 2^i c\frac{n}{2^i} = cn\ell$ comparisons. The bottom remaining levels use $O\left(\sum_{i=\ell}^{\infty} \frac{n}{2^i}h\right) =$

$O\left(\frac{n}{2^\ell}h\right)$ comparisons. The total number of comparisons used is $cn\ell + O\left(\frac{n}{2^\ell}h\right)$. Choosing $\ell = \log h$ gives $cn\ell + O\left(\frac{n}{2^\ell}h\right) = cn\log h + O(n)$.

We show how to reduce the leading constant to 1, but first we present a few tools.

3.1.3 Optimal Multiple Selection

The multiple selection problem asks for the elements of rank $r_1, r_2, \ldots, r_k$ from a linearly ordered set of n elements. Let B denote the information-theoretic lower bound on the number of comparisons needed for multiple selection. A randomized algorithm by Kaligosi et. al. solves the problem using $B + O(n)$ expected comparisons. Here $B = \sum_{j=1}^{k+1} \Delta_j \log \frac{n}{\Delta_j} - O(n)$, where $\Delta_j = r_j - r_{j-1}$, $r_0 = 0$, and $r_{k+1} = n$ [39]. Kaligosi et. al. also gave a deterministic algorithm that solves the problem in $O(B)$ time and $B + o(B) + O(n)$ element comparisons.

We focus on the special case where $r_1, \ldots, r_k$ are equally spaced. That is, we want the $b-1$ elements of rank $\frac{n}{b}, \frac{2n}{b}, \ldots, \frac{(b-1)}{b}n$ that partition the set into b equally sized subsets. In this case, $B = n\log b - O(n)$. Kaligosi et. al.'s randomized algorithm uses $n\log b + O(n)$ expected number of comparisons.

We note that, in this case, there is a deterministic algorithm that uses at most $n\log b + O(n\log\log b)$ comparisons. We divide the input into $\frac{n}{s}$ groups of size s and sort each group in total $\frac{n}{s}(s\log s + O(s))$ number of comparisons. Frederickson and Johnson [30] described a selection algorithm for $\frac{n}{s}$ sorted arrays of size s with $O(\frac{n}{s}\log s)$ running time, so b selections take $O(b\frac{n}{s}\log s)$ time. Choosing $s = b\log b$ gives a deterministic multi-selection algorithm with the claimed bound.

Lemma 3.1.1. *Given a set of n linearly ordered elements, the elements of rank $\frac{n}{b}, \frac{2n}{b}, \ldots, \frac{(b-1)}{b}n$ can be found with a randomized algorithm that uses $n\log b + O(n)$ expected number of comparisons.*

Lemma 3.1.2. *Given a set of n linearly ordered elements, the elements of rank $\frac{n}{b}, \frac{2n}{b}, \ldots, \frac{(b-1)}{b}n$ can be found with a deterministic algorithm that uses $n\log b + O(n\log\log b)$ comparisons.*

3.1.4 Our New Top-Down Algorithm

We present our first new output-sensitive algorithm for computing the maxima in two dimensions. Our new algorithm is randomized and uses $1n\log h + O\left(n\sqrt{\log h}\right)$ expected

number of comparisons. It is a modification of Kirkpatrick and Seidel's algorithm. The key difference is that instead of dividing the set into two equal-sized subsets by the median element, we use Lemma 3.1.1 or Lemma 3.1.2 to divide our set into b equally sized subsets by the $b-1$ quantiles. Our new branching factor b grows as the level of recursion increases.

Our algorithm is as follows:
Let $b_i = \frac{r_{i+1}}{r_i}$, $r_i = 2^{i^2}$, and the initial value of i is 1.

2DMaxima(S, i)

1: split S into b_i subsets of equal-sized $S_1, S_2, \ldots, S_{b_i}$ by the $b_i - 1$ quantiles of the x-coordinates
2: $maxY \leftarrow -\infty$
3: **for** each subset S_j going right to left **do**
4: Prune($S_j, maxY$)
5: 2DMaxima(S_j, $i+1$)
6: $maxY \leftarrow \max\{maxY, \max\{p \in S_j \mid p \in S_j\}\}$
7: **end for**

Analysis:

Again, we inspect the recursion tree and break the tree into the top ℓ levels and the bottom remaining levels. A sub-problem of size n uses $n \log b + O(n)$ comparisons to find the $b-1$ quantiles and $O(n)$ comparisons to prune the sets. The branching factor is $b_i = \frac{r_{i+1}}{r_i}$, so the number of nodes in level i is at most r_i. At level i, the size of each node is at most $\frac{n}{r_i}$ and there are at most r_i such nodes in the level. Thus, for $i < \ell$, level i uses $n \log\left(\frac{r_{i+1}}{r_i}\right) + O(n)$ comparisons. Over the top ℓ levels, the total number of comparisons used is

$$
\begin{aligned}
&\leq \sum_{i=0}^{\ell-1} n \log\left(\frac{r_{i+1}}{r_i}\right) + O(n) \\
&\leq n \sum_{i=0}^{\ell-1} (\log r_{i+1} - \log r_i) + \sum_{i=0}^{\ell-1} O(n) \\
&\leq n \log r_\ell + O(n\ell) \quad \text{by a telescoping sum (see Figure 3.3).}
\end{aligned}
$$

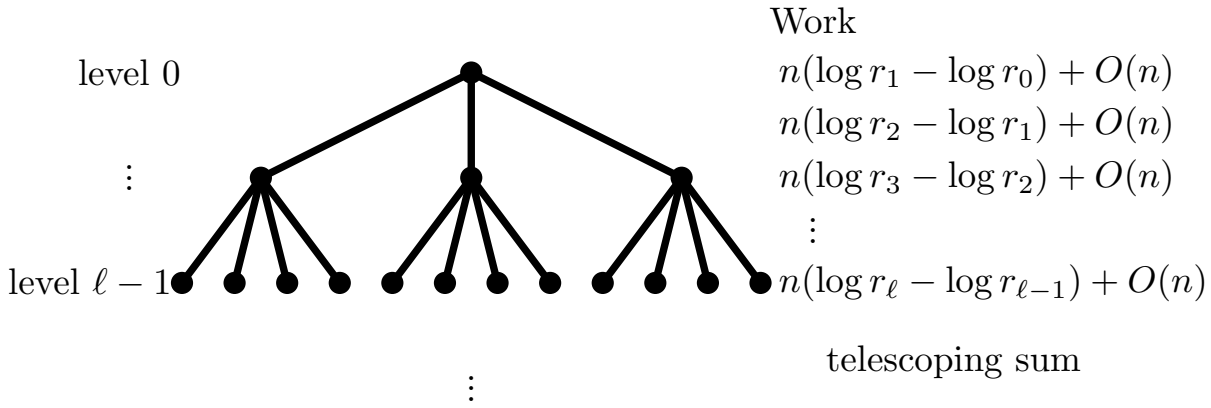

Figure 3.3: The top ℓ levels of the recursion tree.

Due to pruning, the number of nodes at any level is also bounded by h. At level i, $i \geq \ell$, the work per level is $O\left(h\frac{n}{r_i}\log\left(\frac{r_{i+1}}{r_i}\right)\right)$. Since we chose r_i to be super-exponential in i, then the total number of comparisons used for the bottom levels is bounded by $O\left(h\frac{n}{r_\ell}\log\left(\frac{r_{\ell+1}}{r_\ell}\right)\right)$; see Figure 3.4.

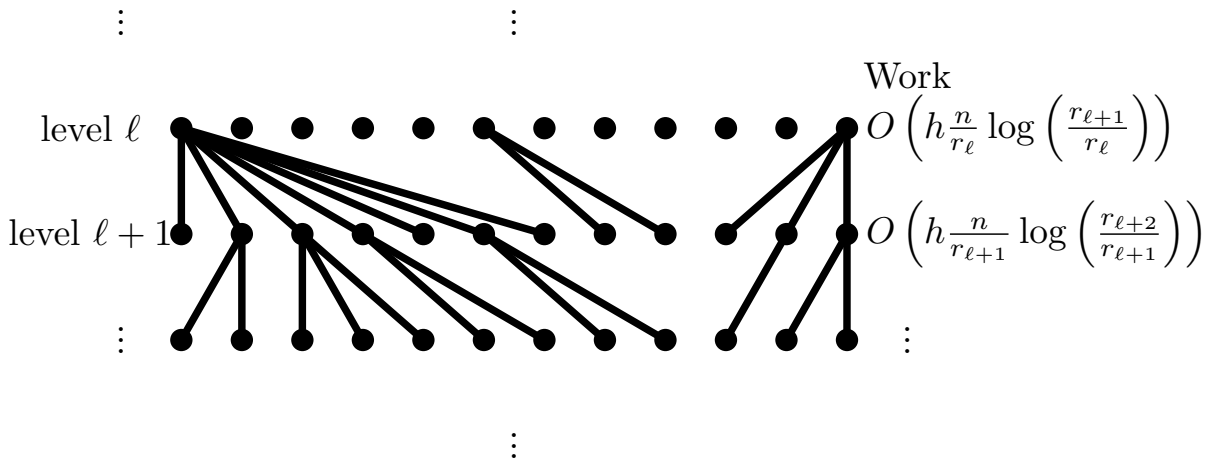

Figure 3.4: The bottom levels of the recursion tree.

Our goal is to minimize the expression $n\log r_\ell + O(n\ell) + O\left(h\frac{n}{r_\ell}\log\left(\frac{r_{\ell+1}}{r_\ell}\right)\right)$. Since $r_i = 2^{i^2}$ and ℓ such that $r_{\ell-1} < h\log h \leq r_\ell$, then, we get $\ell < \sqrt{\log(h\log h)} + 1$. Notice

that $\ell \in O\left(\sqrt{\log\left(h \log h\right)}\right)$, and $r_\ell \in O\left(h \log h\right)$.

The total work is bounded by

$$1n \log r_\ell + O\left(n\ell\right) + O\left(h\frac{n}{r_\ell} \log\left(\frac{r_{\ell+1}}{r_\ell}\right)\right).$$

The first term is bounded by

$$\begin{aligned}
&\leq 1n \log r_\ell \\
&\leq 1n \log 2^{\ell^2} \\
&\leq 1n\ell^2 \\
&\leq 1n\left(\sqrt{\log\left(h \log h\right)} + 1\right)^2 \\
&\leq 1n\left(\sqrt{\log\left(h \log h\right)}\right)^2 + 2n\sqrt{\log\left(h \log h\right)} + n \\
&\leq 1n\log h + O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The second term is bounded by

$$\begin{aligned}
&\leq O\left(n\ell\right) \\
&\leq O\left(n\left(\sqrt{\log\left(h \log h\right)}\right)\right) \\
&\leq O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The third term is bounded by

$$
\begin{aligned}
&\leq O\left(h\frac{n}{r_\ell}\log\left(\frac{r_{\ell+1}}{r_\ell}\right)\right)\\
&\leq O\left(h\frac{n}{h\log h}\log r_{\ell+1}\right)\\
&\leq O\left(h\frac{n}{h\log h}\log 2^{(\ell+1)^2}\right)\\
&\leq O\left(\frac{n}{\log h}(\ell+1)^2\right)\\
&\leq O\left(\frac{n}{\log h}\left(\sqrt{\log\left(h\log h\right)}+2\right)^2\right)\\
&\leq O\left(\frac{n}{\log h}\log\left(h\log h\right)\right)\\
&\leq O\left(n\right).
\end{aligned}
$$

Thus, the total number of comparisons is bounded by

$$
\begin{aligned}
&\leq 1n\log r_\ell + O\left(n\ell\right) + O\left(h\frac{n}{r_\ell}\log\left(\frac{r_{\ell+1}}{r_\ell}\right)\right)\\
&\leq n\log h + O\left(n\sqrt{\log h}\right).
\end{aligned}
$$

We note that deterministic multiple selection uses only $n\log b + O(n\log\log b)$ comparisons. In using the deterministic multi-selection algorithm, the cost of the second term grows to $O\left(n\left(\sqrt{\log h}\right)(\log\log h)\right)$ comparisons. In total the deterministic variant of our algorithm uses $1n\log h + O\left(n\left(\sqrt{\log h}\right)(\log\log h)\right)$ comparisons.

Theorem 3.1.3 (Randomized Top-Down 2D Maxima)**.** *The maxima for a set of n points in two dimensions can be computed using a randomized algorithm that uses $1n\log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons, where h is the size of the maxima.*

Theorem 3.1.4 (Deterministic Top-Down 2D Maxima)**.** *The maxima for a set of n points in two dimensions can be computed using a deterministic algorithm that uses $1n\log h + O\left(n\left(\sqrt{\log h}\right)(\log\log h)\right)$ comparisons, where h is the size of the maxima.*

3.1.5 Our New Bottom-Up Algorithm

In this section, we present our second output-sensitive algorithm for computing the maxima in two dimensions. The algorithm is similar to Chan's algorithm for convex hulls [4], we present Chan's algorithm in Section 3.3.6. It is a bottom-up approach where the work is first done in the leaves of the recursion and then the sub-solutions are combined together.

We first define an r-grouping and prove two lemmas. An *r-grouping* is a partition of the problem into $\lceil \frac{n}{r} \rceil$ groups, each of size at most r together with the maxima (staircase) of each group. Each staircase stores the points in x-sorted order.

Lemma 3.1.5. *Given an r-grouping, we can compute the h maximal points using $O\left(h\frac{n}{r}\log r\right)$ comparisons.*

Proof. We use a technique similar to Jarvis' march [38]. Given a current maximal point q, we can find the next maximal point q' using $O\left(\frac{n}{r}\log r\right)$ comparisons. We repeat this operation h times to find all maximal points using $O\left(h\frac{n}{r}\log r\right)$ comparisons.

We describe how to find the next maximal point using $O\left(\frac{n}{r}\log r\right)$ comparisons. Let q be the current maximal point. If there is only one staircase S, the next maximal point is the rightmost point on S that is above q. Such a point can be found using $O(\log r)$ comparisons by a binary search on S. We have $\frac{n}{r}$ staircases; each staircase emits one candidate point, each found using a binary search. The next maximal point is the rightmost candidate point. Thus, we can find the next maximal point using $O\left(\frac{n}{r}\log r\right)$ comparisons. □

An r-grouping can be computed from scratch using $\frac{n}{r}\left(r\log r + O(r)\right) = 1n\log r + O(n)$ comparisons by using the trivial solution for each group. Computing an r-grouping and the h maximal points uses a total of $1n\log r + O\left(h\frac{n}{r}\log r\right)$ comparisons. If h is known, we can choose $r = h\log h$. This uses the following number of comparisons:

$$
\begin{aligned}
&\leq 1n\log r + O\left(h\frac{n}{r}\log r\right)\\
&\leq 1n\log\left(h\log h\right) + O\left(h\frac{n}{(h\log h)}\log\left(h\log h\right)\right)\\
&\leq 1n\log h + 1n\log\log h + O\left(\frac{n}{\log h}\left(\log h + \log\log h\right)\right)\\
&\leq 1n\log h + O\left(n\log\log h\right).
\end{aligned}
$$

The value of h is not known, so we use a guessing trick that is similar to Chan's algorithm for convex hull. To keep the constant factor small, instead of computing the r-groupings from scratch, we compute a new r'-grouping from the previous r-grouping, where $r < r'$.

Lemma 3.1.6. *If $r < r'$, given an r-grouping, an r'-grouping can be computed using $1n \log \frac{r'}{r} + O(n)$ comparisons.*

Proof. It is sufficient to show how to compute a single r'-maxima from $\frac{r'}{r}$ r-maximas.

First, recall that each r-maxima is already x-sorted, so we can treat the maximas as x-sorted lists. We combine the $\frac{r'}{r}$ x-sorted lists, each of size r, into a single x-sorted list of size r'. Merging these x-sorted lists together using a divide-and-conquer approach uses at most $r' \log \frac{r'}{r} + O(r')$ x-comparisons. Second, we use the trivial sweep line algorithm on the x-sorted list to find the new maxima, the sweep line uses at most $O(r')$ y-comparisons. Thus, the total number of comparisons used for a group of $\frac{r'}{r}$ r-maximas is $1r' \log \frac{r'}{r} + O(r')$ comparisons. This is repeated $\frac{n}{r'}$ times, once for each group of r-maximas. □

Our new algorithm is as follows:

2DMaxima(S)

1: **for** $i = 1, 2, \ldots$ until all the maximal points are found **do**
2: compute the r_i-grouping from the previous r_{i-1}-grouping (Lemma 3.1.6)
3: find the maxima of S using Lemma 3.1.5, but stop when $h_i \log h_i > r_i$
4: ▷ h_i is the number of maximal points found in the i^{th} iteration
5: **end for**

Where $r_i = 2^{i^2}$.

Analysis:

Let ℓ such that $r_{\ell-1} < h \log h \leq r_\ell$, where h is size of the output. We derive some bound for convenience: $\ell < \sqrt{\log(h \log h)} + 1 \in O\left(\sqrt{\log h}\right)$, $\log r_{\ell+1} \leq \log h + O\left(\sqrt{\log h}\right)$, and

$$\begin{aligned}
\log r_\ell &\leq \log 2^{\ell^2} \\
&\leq \ell^2 \\
&\leq \left(\sqrt{\log(h \log h)} + 1\right)^2 \\
&\leq \log(h \log h) + O\left(\sqrt{\log(h \log h)}\right) \\
&\leq \log h + O\left(\sqrt{\log h}\right).
\end{aligned}$$

The number of comparisons used for combining staircases (line 2) is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} 1n \log\left(\frac{r_{i+1}}{r_i}\right) + O(n) \\
&\leq 1n \log r_{\ell+1} + O(n\ell) \quad \text{by a telescoping sum} \\
&\leq 1n \log h + O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The number of comparisons used for computing the maximal points (line 3) is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(h_i \frac{n}{r_i} \log r_i\right) \quad \text{where } h_i \text{ is the number of maximal points found in the } i^{\text{th}} \\
&\qquad\qquad \text{iteration} \\
&\leq O\left(h\frac{n}{r_\ell}\log r_\ell\right) + \sum_{i=1}^{\ell-1} O\left(h_i \frac{n}{(h_i \log h_i)} \log\left(h_i \log h_i\right)\right), \quad \text{since we terminate early,} \\
&\qquad\qquad r_i = h_i \log h_i \text{ for } i < \ell \\
&\leq O\left(h\frac{n}{h\log h}\log h\right) + O(n\ell), \quad \text{since } h\log h \leq r_\ell \text{ and } \log r_\ell \in O(\log h) \\
&\leq O\left(n\sqrt{\log h}\right).
\end{aligned}$$

In total the algorithm uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons.

Theorem 3.1.7 (Bottom-Up 2D Maxima)**.** *The maxima for a set of n points in two dimensions can be computed using an algorithm that uses $1n \log h + O\left(n\left(\sqrt{\log h}\right)\right)$ comparisons, where h is the size of the maxima.*

3.1.6 Discussion

Our new top-down algorithm is randomized and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons, and its deterministic variant uses $1n \log h + O\left(n\left(\sqrt{\log h}\right)(\log\log h)\right)$ comparisons.

Our new bottom-up algorithm is deterministic and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons; in particular, it removes the $\log\log h$ term in the lower order term.

3.2 The Maxima Problem in Three Dimensions

In this section we discuss the problem of computing the maxima in three dimensions. We will consider the case where the points in general position, that is, no two points share a common x-coordinate, y-coordinate, or z-coordinate. Again, this classic problem has been solved asymptotically optimally.

3.2.1 The Trivial Solution

The trivial solution is to do three sorts, one along each axis; afterwards, no more comparisons are needed. This costs $3n \log n + O(n)$ comparisons.

A simple improvement is an incremental sweep plane algorithm:

3DMaxima(S)

1: sort all the points by the z-coordinates
2: **for** each point p in decreasing z-order **do**
3: let M be the current maxima projected onto the xy-plane
4: using a binary search, determine if p is above M
5: if p is 'above' M then add p to the maxima staircase
6: **end for**

If the two-dimensional maxima staircase is maintained as a sorted list then the algorithm uses $2n \log n + O(n)$ comparisons.

3.2.2 Clarkson-Shor Random Sampling

The Clarkson-Shor random sampling technique [17, 19] is a cornerstone of computational geometry. Notable results include: an algorithm for finding all k intersections among a set of n line segments in the plane in $O(n \log n + k)$ expected time; an algorithm for finding the diameter of a set of n points in three dimensions in $O(n \log n)$ expected time; and an algorithm for computing the convex hull of n points in d dimensions in $O(n^{\lfloor d/2 \rfloor} + n \log n)$ expected time. Informally, Clarkson and Shor showed that for various problems, given a set S of n objects, a random sample $R \subset S$, where $|R| = r$, can be expected to divide the problem into $O(r)$ sub-problems of size $O(\frac{n}{r})$.

For example, given a set of n points on a line and a random sample $R \subset S$ of size r, R partitions the line into $O(r)$ intervals, we can expect each interval to contain $O(\frac{n}{r})$

points and the largest interval contains at most $O(\frac{n}{r}\log r)$ points. The probability that a random sample satisfies these conditions is greater than a constant and the expected number of trials needed until a random sample satisfies these conditions is a constant. Clarkson and Shor gave a general framework to extend this type of random sampling to higher dimensions. We use Clarkson-Shor's random sampling technique instead of optimal multi-selection.

3.2.3 Helpful Definitions and Facts

For a point (x, y, z) we define the *orthant* of the point to be the region $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$. The *staircase polyhedron* $P(S)$ is the union of the orthants of S (see Figure 3.5), the staircase polyhedron has $O(|S|)$ edges, faces, and vertices. We can think of the staircase polyhedron as the region 'under' the staircase. Computing the maxima reduces to computing the staircase polyhedron, as the maximal points are vertices of the polyhedron. The vertices of the polyhedron are the nodes of the graph, and the edges of the polyhedron are the edges of the graph. The staircase polyhedron can be represented as a planar graph.

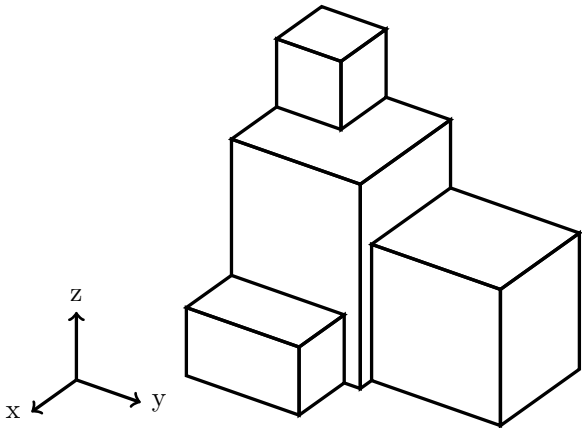

Figure 3.5: The staircase polyhedron in three dimensions.

We define $VD(S)$ to be the *vertical decomposition* of the complement of the staircase polyhedron, that is the vertical decomposition of the region 'above' the staircase. We take each horizontal face and subdivide it into rectangles by extending line segments that are parallel to the x-axis from the vertices. Each rectangle of the subdivision is then extended upward to $z = \infty$ resulting in a decomposition consisting of rectangular boxes. See Figure

3.14 for an example. The graph of the vertical decomposition represents the adjacency structure of the cells. Every cell has a corresponding node in the graph, and if two cells are adjacent to each other their nodes are adjacent in the graph as well.

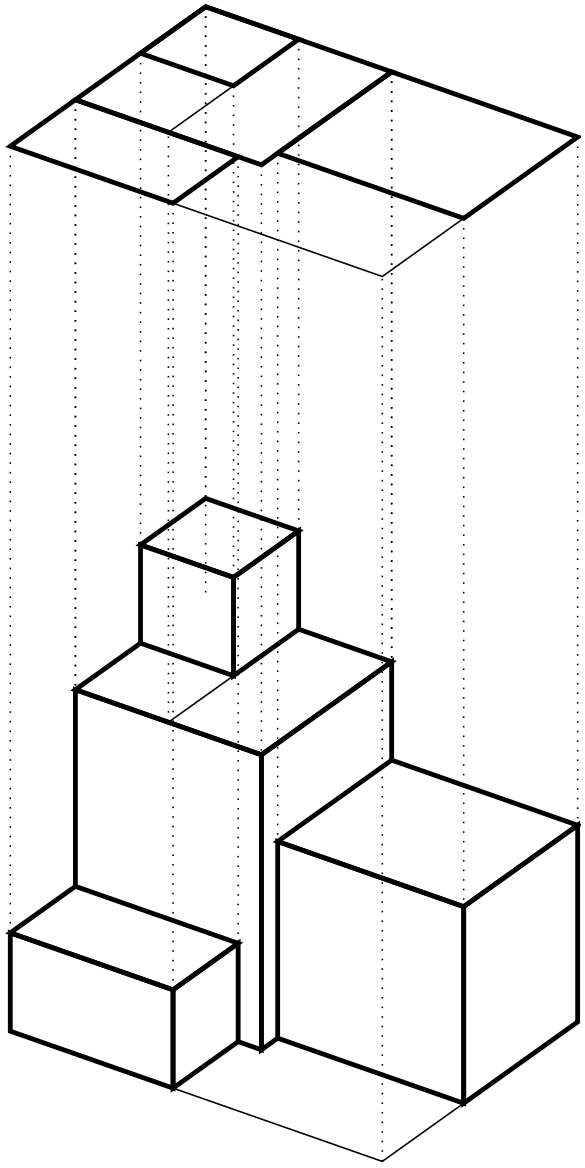

Figure 3.6: The vertical decomposition of a staircase polyhedron in three dimensions.

The *conflict list* for a cell $\Delta_j \in VD(R)$ holds the list of vertices of S whose orthants

intersect Δ_j; that is, the members of the sub-problem for the box Δ_j. In Figure 3.7, A's conflict list is $\{1, 2\}$, B's is $\{2, 3\}$, and C's is $\{1, 4\}$.

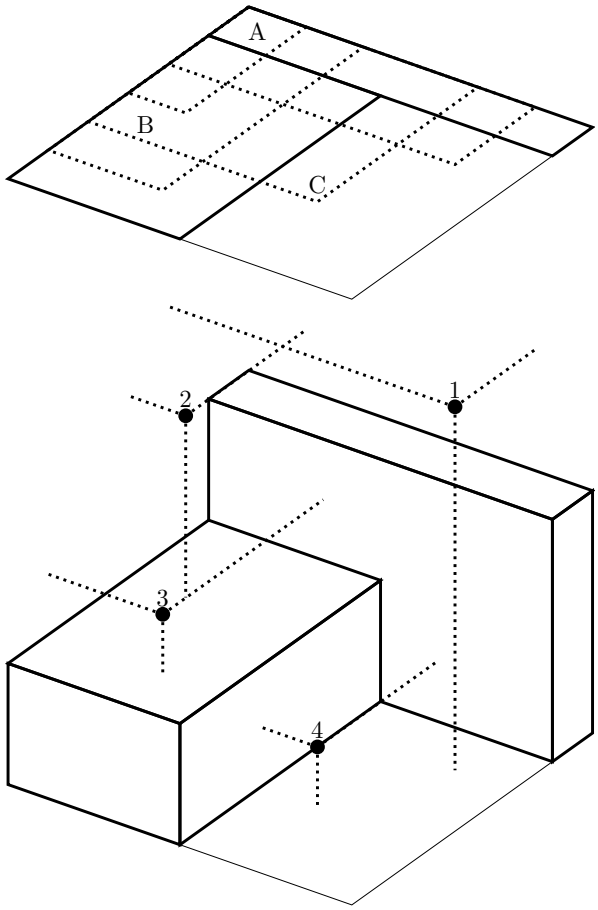

Figure 3.7: An r-grouping and the conflicting orthants.

By Clarkson-Shor random sampling, the vertical decomposition of the staircase polyhedron of r randomly chosen points is expected to partition the problem into $O(r)$ roughly equal-sized sub-problems, with the largest sub-problem expecting to have at most $O(\frac{n}{r} \log r)$ points. Let denote Δ a cell of $VD(R)$ and let S_Δ denote the conflict list of Δ. Formally, given a set S of n points in the plane and a random sample $R \subset S$ of size r, Clarkson and Shor showed that the vertical decomposition of R has the following properties:

- $E\left[\sum_{\Delta \in VD(R)} |S_\Delta|\right] = O(n)$.

- $E\left[\sum_{\Delta \in VD(R)} |S_\Delta| \log |S_\Delta|\right] = O(n \log \frac{n}{r})$.
- $E\left[\sum_{\Delta \in VD(R)} \delta_\Delta |S_\Delta|\right] = O(n)$, where δ_Δ is the number of cells adjacent to Δ in $VD(R)$.

We define an *r-division* to consist of:

- A random sample $R \subset S$ of size r.
- A point location data structure for $VD(R)$ with a query algorithm that uses $1 \log r + O(\sqrt{\log r})$ comparisons.
- The conflict list S_Δ for every $\Delta \in VD(R)$.

The vertical decomposition of R partitions the maxima problem into independent sub-problems. The solution of each sub-problem can be glued together to produce the final solution.

Lemma 3.2.1. *It is possible to construct an r-division using* $1n \log r + O(n\sqrt{\log r}) + O(r \log r)$ *expected number of comparisons.*

Proof. 1) Point location: We use Seidel and Adamy's data structure for planar point location [57]. Their data structure has a construction algorithm that uses $O(r \log r)$ comparisons for $O(r)$ regions and a query algorithm that uses $1 \log r + O(\sqrt{\log r})$ comparisons.

2) To generate the conflict lists: It is sufficient to show how to compute the list of cells of $VD(R)$ that conflict with the orthant of a query point q. First, use the point location data structure to determine which cell Δ contains q using $1 \log r + O(\sqrt{\log r})$ comparisons. Starting from cell Δ, walk along the vertical decomposition to find all the cells that the orthant of q intersects. A walk is similar to a breath-first or depth-first search in the graph of the vertical decomposition with the additional condition that a cell is only visited if it intersects the orthant of q. That is, starting in the cell Δ compare the orthant of q to all the neighbours of Δ (cell adjacent to Δ in $VD(R)$). For each neighbour that intersects the orthant, visit those neighbours and recurse until all the intersecting cells have been visited. One step of the walk uses δ_Δ comparisons, where δ_Δ is the degree of a cell Δ (number of neighbours in $VD(R)$). The size of the conflict list is $O\left(\sum_{\Delta \in VD(R)} \delta_\Delta |S_\Delta|\right)$; by

a Clarkson-Shor analysis its expected value is $O(n)$. Thus, the expected cost to compute the conflict lists is $1n \log r + O(n\sqrt{\log r})$ comparisons.

We compute an r-division as follows:

r-division(S)

1: take a random sample $R \subset S$ of size r
2: build Seidel and Adamy's point location data structure for $VD(R)$
3: **for** each $s \in S$ **do**
4: locate the cell $\Delta \in VD(R)$ that contains the corner of s
5: find all cells intersecting s by walking in $VD(R)$, starting from Δ
6: **end for**
7: **for** each $\Delta \in VD(R)$ **do**
8: obtain the conflict list $S_\Delta = \{s \in S : \text{orthant of } s \text{ intersects } \Delta\}$
9: **end for**

□

3.2.4 Our New Algorithm

Our new algorithm is a divide-and-conquer approach; it closely resembles Kirkpatrick and Seidel's marriage-before-conquest algorithm. We would like to divide the problem into r regions of approximately the same size; in two dimensions this was achieved by a multi-selection algorithm; in three dimensions this is much harder. Instead, we use Clarkson-Shor random sampling to pick r random points whose vertical decomposition divides the problem into $O(r)$ regions each with an expected size of $O(\frac{n}{r})$.

3DMaxima(S)

1: build an r-division for S, denote R as the random sample of S
2: **for** each $\Delta \in VD(R)$ **do**
3: solve the sub-problem for S_Δ inside Δ
4: **end for**
5: glue all the solutions together

Analysis:

By Lemma 3.2.1, an r-division can be created using $1n \log r + O(n\sqrt{\log r}) + O(r \log r)$ comparisons.

We use the sweep line algorithm to solve each sub-problem, so line 3 uses

$O\left(\sum_{\Delta \in VD(R)} |S_\Delta| \log |S_\Delta|\right)$ comparisons; by a Clarkson-Shor analysis, the expected value is $O\left(n \log \frac{n}{r}\right)$.

In total, we use $1n \log r + O(n\sqrt{\log r}) + O\left(r \log r\right) + O(n \log \frac{n}{r})$ comparisons. By choosing $r = \frac{n}{\log n}$, the expected number of comparisons used is:

$$\leq 1n \log\left(\frac{n}{\log n}\right) + O\left(n\sqrt{\log\left(\frac{n}{\log n}\right)}\right) + O\left(\left(\frac{n}{\log n}\right) \log\left(\frac{n}{\log n}\right)\right) + O\left(n \log \frac{n}{\left(\frac{n}{\log n}\right)}\right)$$

$$\leq 1n \log n + O(n\sqrt{\log n}) + O(n) + O(n \log \log n)$$

$$\leq 1n \log n + O(n\sqrt{\log n}).$$

Theorem 3.2.2 (3D Maxima)**.** *The maxima for a set of n points in three dimensions can be computed using a randomized algorithm that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.*

3.2.5 Our New Output-Sensitive Algorithm

We present a new output-sensitive algorithm for computing the maxima in three dimensions that uses at most $1n \log h + O(n \log^{\frac{2}{3}} h)$ comparisons. We combine ideas from both previous algorithms for maxima. Similar to the previous top-down maxima algorithm in three dimensions, we use r-divisions to partition our problem into smaller sub-problems. But, instead of solving each sub-problem directly, we use the wrapping idea from the bottom-up output-sensitive algorithm for maxima in two dimensions. We also use a guessing trick to make it output-sensitive, and merging of r-divisions to ensure that the cost of overshooting stays small.

Ray Shooting

Lemma 3.2.3. *Given an r-division, with a random sample R, we can compute the h maximal points in $O(h\frac{n}{r} \log^2 r)$ expected number of comparisons.*

Proof. We use an algorithm similar to gift wrapping or Jarvis' march [38] to compute the staircase polyhedron $P(S)$. Given a current vertex q on $P(S)$, we can find a neighbouring vertex q' using $O(\frac{n}{r} \log^2 r)$ comparisons, by a ray shooting query. A *ray shooting query*

determines the first face of $P(S)$ hit by an axis-parallel ray shot from q. From the face we can determine which point q' generates the orthant which contains the face. Each vertex has at most six neighbours, one in each direction. To generate the entire polyhedron, we can use a breadth-first search or depth-first search on the graph of the staircase polyhedron (see Section 3.2.3). At a vertex a ray shooting query along one of six directions will find the next vertex of the polyhedra along that direction. A ray shooting query can be thought of as a step in a breadth-first search or depth-first search. By using $O(h)$ ray shooting queries, where h is the size of maxima, the entire polyhedra can be generated.

The ray shooting query is a complex operation; we first describe a related subroutine and its relationship to ray shooting. Then we describe how they are used together to perform a ray shooting query.

Membership test: Given a point q' and an orthogonally convex polyhedron $P(S)$, determine if q' is within $P(S)$. To perform a membership test, first locate the cell $\Delta \in VD(R)$ that contains q', then check q' against every orthant in the conflict list of Δ. This uses a total of $O(\log r) + O\left(\max_{\Delta \in VD(R)} |S_\Delta|\right) = O(\frac{n}{r} \log r)$ expected number of comparisons.

A ray shooting query reduces to $O(\log r)$ membership tests as follows: Let ℓ be the axis-aligned ray shot from q; project all the vertices of the random sample, R, onto ℓ. By a binary search over ℓ, we can find two consecutive points q_i and q_{i+1} on ℓ such that $q_i \notin P(S)$ and $q_{i+1} \in P(S)$ using $O(\log r)$ membership tests.

These two points, q_i and q_{i+1}, lie in a common cell $\Delta \in VD(R)$. Otherwise, let Δ_i be the cell that contains q_i and Δ_{i+1} be the cell that contains q_{i+1}. The cell Δ_{i+1} has a face f that is orthogonal of ℓ. Since q_i and q_{i+1} are in different cells, f and ℓ intersect at some point q_f between q_i and q_{i+1}. By the construction of the vertical decomposition, face f is incident to a vertex v of R. The projection of v onto ℓ is q_f, this contradicts the fact that q_i and q_{i+1} are consecutive points.

Since $P(S)$ is orthogonally convex, $q_i \notin P(S)$, $q_{i+1} \in P(S)$, and are both in the same cell Δ, the first orthant hit by the ray ℓ must be an orthant intersecting Δ; do a brute-force search over the conflict list of Δ to find that orthant.

In total a ray shooting query uses $O(\log r)$ membership tests, a membership test uses $O(\frac{n}{r} \log r) = O(\frac{n}{r} \log^2 r)$ expected number of comparisons, thus in total a ray shooting query uses $O(\log r)O(\frac{n}{r} \log r) = O(\frac{n}{r} \log^2 r)$ expected number of comparisons. In Figure 3.8, shooting a ray from q to find the next orthant reduces to a binary search on A', B' and C' using membership tests, where A', B' and C' is the projection of A, B and C onto the ray.

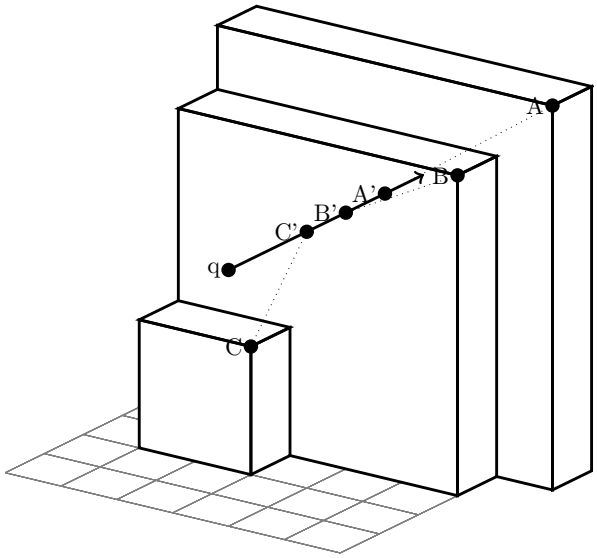

Figure 3.8: An illustration of a ray shooting query.

The ray shooting algorithm can be summarized as follows:

rayshooting(q)

1: let ℓ by the axis-parallel ray from q
2: project the vertices of $P(R)$ onto ℓ
3: do a binary search over ℓ to find q_i and q_{i+1}, such that $q_i \notin P(S)$ and $q_{i+1} \in P(S)$
4: let Δ be the cell containing q_i and q_{i+1}
5: do brute-force search over the conflict list of Δ to find the first orthant hit by the ray

Naively line 2 uses $O(r)$ time but zero comparisons. We can reduce the time complexity to $O(r \log r)$ (amortized) time by precomputing the projections.

We note that since ℓ is an axis-aligned ray, it must have one of at most six possible directions. Without loss of generality we consider the positive x direction. The projections onto these rays are in fact the same regardless of the starting vertex q. In fact, the projections are simply the vertices of $P(R)$ sorted by their x-coordinate. Thus, the projections can be precomputed. Prior to any ray shooting, project all the vertices of $P(R)$ onto a line ℓ that is parallel to the x-axis. This pre-computation uses at most $O(r \log r)$ comparisons and can be done as an additional step for computing an r-division.

The projection step of a ray shooting query for point q reduces to a binary search for q on ℓ, this binary search uses at most $O(\log r)$ comparisons.

□

Growing r-divisions

Computing an r-division from scratch uses $1n \log r + O(n\sqrt{\log r}) + O(r \log r)$ expected comparisons and computing the h-maxima uses $O(h\frac{n}{r} \log^2 r)$ expected number of comparisons. If h is known, we set $r = h \log^2 h$ to get $1n \log h + O(n\sqrt{\log h})$ comparisons. But h is not known, so we employ a guessing trick and a growing trick to keep the cost of overshooting small.

Lemma 3.2.4. *Given an r-division and $r' > r$, we can compute an r'-division in $1n \log \frac{r'}{r} + O\left(n\sqrt{\log \frac{r'}{r}} + n \log\log r + r' \log r'\right)$ expected number of comparisons.*

Proof. Given an r-division corresponding to a random sample $R \subset S$, we:

1. Take a random sample $R' \subset S$ of size r' such that R' contains R.
2. Compute a point location data structure for $VD(R')$ in $O(r' \log r')$ comparisons.
3. Compute the conflict list for R'. This is the hardest part and we describe how to do this in the following section.

Notice that the conflict list for a cell Δ can be computed independently from all the other cells. We generate each conflict list independently and then glue them together.

ConflictList(R')

```
1: for each Δ ∈ VD(R) do
2:     make a point location data structure on the xy-projection of VD(R') restricted to Δ
3:     identify all the conflicting orthants by examining the conflict list for Δ
4:     for each orthant with point p do
5:         identify the cell Δ' ∈ VD(R') that contains p using the point location data structure
6:         starting from Δ' walk along VD(R') to find all conflicting cells
7:     end for
8: end for
```

For the definition of a walk used in line 6 see the proof of Lemma 3.2.1.

Line 2 uses $O\left(\sum_{\Delta \in VD(R)} |R'_\Delta| \log |R'_\Delta|\right)$ comparisons. Since R is random sample of R', intuitively we expect each cell of R to have $O(\frac{r'}{r})$ points; by a Clarkson-Shor analysis, the expression has an expected value of $O(r\frac{r'}{r} \log \frac{r'}{r}) = O(r' \log \frac{r'}{r})$.

Line 3 can be done with a linear scan over the conflict list for R, in $O\left(\sum_{\Delta \in VD(R)} |S_\Delta|\right)$ comparisons. By a Clarkson-Shor analysis, this has an expected value of $O(r\frac{n}{r}) = O(n)$.

Line 5 uses $1n \log \eta + O(n\sqrt{\log \eta})$ comparisons, where $\eta = \max_{\Delta \in VD(R)} |R'_\Delta|$. By a Clarkson-Shor analysis, the expected value of η is $O(\frac{r'}{r} \log r)$. By Jensen's inequality the expected value of $1n \log \eta + O(\sqrt{\log \eta})$ is

$$\leq 1n \log \left(\frac{r'}{r} \log r\right) + O\left(n\sqrt{\log \left(\frac{r'}{r} \log r\right)}\right)$$
$$\leq 1n \log \frac{r'}{r} + O(n \log \log r) + O\left(n\sqrt{\log \frac{r'}{r}}\right).$$

Line 6 use $O\left(\sum_{\Delta' \in VD(R')} \delta'_\Delta |S'_\Delta|\right)$ comparisons. By a Clarkson-Shor analysis, its expected value is $O(n)$.

In total, the expected number of comparisons used is:

$$\leq 1n \log \frac{r'}{r} + O(n \log \log r) + O\left(n\sqrt{\log \frac{r'}{r}}\right) + O(r' \log \frac{r'}{r}) + O(n) + O(r' \log r')$$
$$\leq 1n \log \frac{r'}{r} + O(n \log \log r) + O\left(n\sqrt{\log \frac{r'}{r}}\right) + O(r' \log r').$$

□

The Algorithm

We combine the two ideas to get our new algorithm:

We choose the sequence $r_i = \min\left\{2^{i^3}, \frac{n}{\log n}\right\}$.

3DMaxima(S)

1: **for** $i = 1, 2, \ldots,$ until all maximal points are found **do**

2: compute the r_i-division from the previous r_{i-1}-division by Lemma 3.2.4
3: try to compute all maximal points of S using the r_i-division by Lemma 3.2.3, until $r_i < h_i \log^2 h_i$ ▷ h_i is the number of maximal points found in this iteration
4: if $\frac{n}{\log n} < h_i \log^2 h_i$, we stop and directly compute the maxima using an $\frac{n}{\log n}$-division
5: **end for**

Analysis:

In iteration i, line 2 uses $1n \log \frac{r_i}{r_{i-1}} + O\left(n\sqrt{\log \frac{r_i}{r_{i-1}}} + n \log\log r_{i-1} + r_i \log r_i\right)$ comparisons. Line 3 uses $O(h_i \frac{n}{r_i} \log^2 r_i)$ comparisons.

Let ℓ satisfy $r_{\ell-1} < h \log^2 h \leq r_\ell$, where h is the output size. Then

$$\begin{aligned} r_{\ell-1} &< h \log^2 h \\ 2^{(\ell-1)^3} &< h \log^2 h \\ \log 2^{(\ell-1)^3} &< \log (h \log^2 h) \\ (\ell-1)^3 &< \log (h \log^2 h) \\ \ell &< \log^{\frac{1}{3}} (h \log^2 h) + 1. \end{aligned}$$

We also see that $\ell \in O(\log^{\frac{1}{3}} h)$.

Overall the total cost is

$$\sum_{i=1}^{\ell} 1n \log \frac{r_i}{r_{i-1}} + O\left(n\sqrt{\log \frac{r_i}{r_{i-1}}} + n \log\log r_{i-1} + r_i \log r_i + h_i \frac{n}{r_i} \log^2 r_i\right) \text{ comparisons.}$$

The first term is

$$
\begin{aligned}
&\leq \sum_{i=1}^{\ell} 1n \log \frac{r_i}{r_{i-1}} \\
&\leq 1n \log r_\ell, \quad \text{by a telescoping sum} \\
&\leq 1n \log 2^{\ell^3} \\
&\leq 1n\ell^3 \\
&\leq 1n(\log^{\frac{1}{3}}(h \log^2 h) + 1)^3 \\
&\leq 1n\left(\log(h \log^2 h) + O(\log^{\frac{2}{3}}(h \log^2 h))\right) \\
&\leq 1n \log h + 1n \log \log^2 h + O(n \log^{\frac{2}{3}} h) \\
&\leq 1n \log h + O(n \log^{\frac{2}{3}} h).
\end{aligned}
$$

The second term is

$$
\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(n\sqrt{\log \frac{r_i}{r_{i-1}}}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(n\sqrt{\log \frac{2^{i^3}}{2^{(i-1)^3}}}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(n\sqrt{i^3 - (i-1)^3}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(n\sqrt{\Theta(i^2)}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(ni\right) \\
&\leq O\left(n\ell^2\right) \\
&\leq O\left(n \log^{\frac{2}{3}} h\right).
\end{aligned}
$$

The third term is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(n \log\log r_{i-1}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(n \log\log 2^{(i-1)^3}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(n \log (i-1)^3\right) \\
&\leq O\left(\ell n \log \ell^3\right) \\
&\leq O\left(\left(\log^{\frac{1}{3}} h\right) n \log\log^{\frac{1}{3}} h\right) \\
&\leq O\left(n \left(\log^{\frac{1}{3}} h\right) \log\log h\right).
\end{aligned}$$

The fourth term is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(r_i \log r_i\right) \\
&\leq \sum_{i=1}^{\ell} O\left(r_i \log r_i\right); \quad r_i = \min\left\{2^{i^3}, \frac{n}{\log n}\right\} \text{ so } r_i \in O\left(\frac{n}{\log n}\right) \\
&\leq \sum_{i=1}^{\ell} O\left(\left(\frac{n}{\log n}\right) \log\left(\frac{n}{\log n}\right)\right) \\
&\leq \sum_{i=1}^{\ell} O(n) \\
&\leq O(n\ell) \\
&\leq O(n \log^{\frac{1}{3}} h).
\end{aligned}$$

The fifth term is

$$
\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(h_i \frac{n}{r_i} \log^2 r_i\right) \\
&\leq O\left(\ell h \left(\frac{n}{r_\ell}\right) \log^2 r_\ell\right) \\
&\leq O\left(\ell h \left(\frac{n}{h \log^2 h}\right) \log^2 r_\ell\right); \quad h \log^2 h < r_\ell \\
&\leq O\left(\ell h \left(\frac{n}{h \log^2 h}\right) \log^2 h\right); \quad \text{by the first term } \log r_\ell \in O(\log h) \\
&\leq O\left(\ell n\right) \\
&\leq O\left(n \log^{\frac{1}{3}} h\right).
\end{aligned}
$$

If $\frac{n}{\log n} < h_i \log^2 h_i$, we compute the maxima directly using a $\frac{n}{\log n}$-division in line 4. This uses an extra $O\left(\sum_{\Delta \in VD(R)} |S_\Delta| \log |S_\Delta|\right)$ number of comparisons; by a Clarkson-Shor analysis its expected value is $O(n \log \log n)$.

Thus, the expected number of comparisons used is $1n \log h + O(n \log^{\frac{2}{3}} h)$.

Theorem 3.2.5 (Output-Sensitive 3D Maxima)**.** *The maxima for a set of n points in three dimensions can be computed using a randomized algorithm that uses $1n \log h + O(n \log^{\frac{2}{3}} h)$ expected number of comparisons, where h is the size of the maxima.*

3.3 The Convex Hull Problem in Two Dimensions

In this section, we discuss the problem of computing a convex hull. The convex hull problem is one of the most fundamental and well known problems in computational geometry. The convex hull problem is closely related to the maxima problem; in fact many convex hull algorithms have analogous maxima algorithms. Our algorithm for convex hull is analogous to our previous maxima algorithms. For simplicity, we focus on computing the upper hull; we also assume that the points are in general position, that is, no three points are collinear

A subset S of the plane is *convex* if and only if for any pair of points $p, q \in S$, the line segment $\overline{pq}$ is completely contained in S. We define the *convex hull* of a set of points S to

be the smallest convex set that contains S. Commonly, the convex hull is given as the set of vertices of the convex set in counter-clockwise (or clockwise) order. A useful alternative definition in two dimensions is: the convex hull is the list of points such that if p and q are consecutive vertices on the hull, then all the points of S are on the same side of the line $\overleftrightarrow{pq}$. See Figure 3.9 for an example.

In two dimensions, we can imagine the points being nails sticking out of the plane; if we take a rubber band and snap it around the nails, so that, the band minimizes its length, the band represents the convex hull of the points.

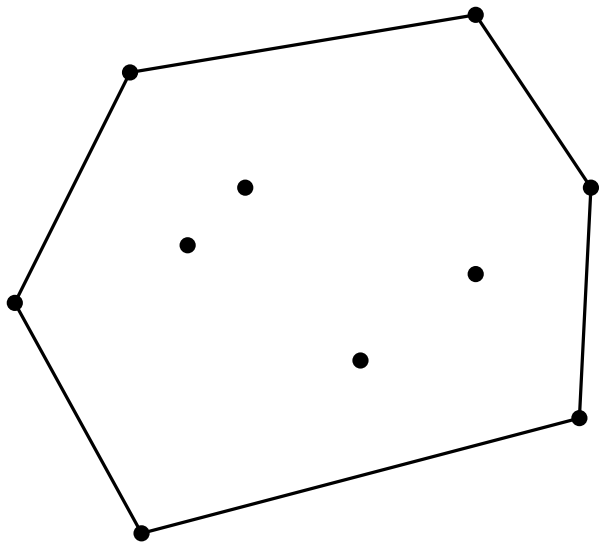

Figure 3.9: A convex hull.

A convex hull can be split into two parts: An *upper hull* that consists of the downward facing edges, and the *lower hull* that consists of all the upward facing edges (see Figure 3.10).

In addition to comparisons between two coordinates, we also use sidedness tests. A sidedness test compares a point to a line and determines which side of the line contains the point. A sidedness test is also defined over three points, a, b, and c; it determines which side of the line $\overleftrightarrow{ab}$ contains c.

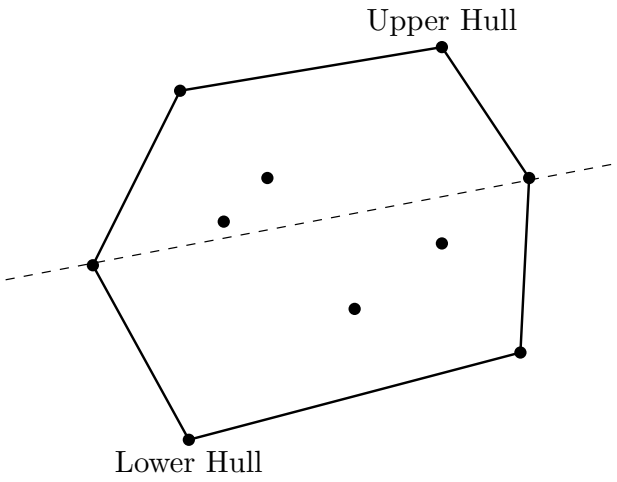

Figure 3.10: An upper hull and a lower hull.

The points a, b, and c form a left turn if the signed area of the triangle abc is positive. The points form a right turn if the signed area is negative. The signed area is equal to half of the determinant of the matrix $\begin{bmatrix} a.x & a.y & 1 \\ b.x & b.y & 1 \\ c.x & c.y & 1 \end{bmatrix}$. So a sidedness test can be implemented as:

CounterClockwise(a, b, c)

1: return $(b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x)$

where a positive number is returned if the points are counter-clockwise and negative number is returned if the points are clockwise [54]. In Figure 3.11 p_3 is to the left of the line $\overleftrightarrow{p_1 p_2}$.

We consider a sidedness test to be more expensive than a comparisons. However, we can not quantify how much more expensive a sidedness test is without making restrictive assumption on the computer architecture and implementation details. We will explicit count both types of operations.

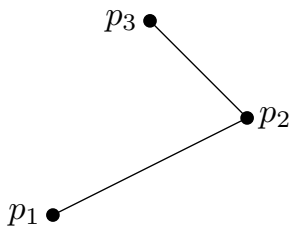

Figure 3.11: A counter-clockwise (or left) turn.

The remainder of the section is as follows: First, we discuss some previous work and present a solution known as Graham's Scan. Second, we present Kirkpatrick and Seidel's algorithm and our new top-down algorithm which is based on Kirkpatrick and Seidel's approach. Third, we present Chan's algorithm and our new bottom-up algorithm which based upon Chan's algorithm.

3.3.1 Previous Work

There are many $O(n \log n)$ time algorithms to compute the convex hull in two and three dimensions. In the algebraic computation tree model the worst case requires $\Omega(n \log n)$ time [54]. There are many output-sensitive algorithms that solve two and three dimensional problem in $O(n \log h)$ time, where h is the output size. In two dimensions, the first output-sensitive algorithm was given by Kirkpatrick and Seidel [41], later simplified by Chan, Snoeyink, and Yap [10]. A different approach by Chan gave a simpler algorithm [4].

Other notable results include: an average case algorithms that run in $O(n)$ expected time for certain probability distributions [54] and an instance-optimal algorithm in two and three dimensions by Afshani, Barbay and Chan [1]. The dynamic version of the problem was investigated by Overmars and van Leeuwen [49] and Chan [6].

Other popular algorithms include: Graham's Scan, Jarvis' March, divide-and-conquer, randomized incremental construction, and quick-hull [54].

The special case of 'orthogonal' convex hull is the maxima problem. The dual of the convex hull problem is the half-plane intersection problem (see Section 3.4.3).

3.3.2 Upper and Lower Hull

Previous convex hull algorithms, such as [41], often computes an upper hull and lower hull separately then glues them together to yield the convex hull. This can be accomplished

by naively executing an upper hull algorithm twice. As our focus is on constants factors, this naive approach is unsatisfactory as it increases the constants by a factor of two.

We reduce the problem of computing a convex hull to computing an upper hull, but we instead of naively executing the algorithm twice, we partition the points S into an upper portion U and a lower portion L. We note that in two dimensions the upper hull and lower hull can separated by a line ℓ through the left most endpoint and the right most endpoint (see Figure 3.10). All the points that are above ℓ belong to the upper portion U, and all the points below ℓ belong to the lower portion L. This partitioning can be accomplished using $O(n)$ comparisons and sidedness tests.

The upper hull of U can be combined with the lower hull of L to produce the convex hull of S in constant time. Our new convex hull algorithms will uses this partitioning trick to reduce the convex hull problem to two upper hull problems without increasing the constant factors.

Let S be a set of n points with a convex hull of size h. Let U be the upper portion, such that the upper hull of U has h_U points. Let L be the lower portion, such that the lower hull of L has h_L points. If there exists an algorithm that computes an upper hull of a set of n points using $cn \log h + o(n \log h)$ comparisons and sidedness tests, where h is the size of the upper hull, then the convex hull of S can be found using $c|U| \log h_U + o(|U| \log h_U) + c|L| \log h_L + o(|L| \log h_L) \leq cn \log h + o(n \log h)$ comparisons and sidedness tests, where $h = h_U + h_L$.

3.3.3 Graham's Scan

Graham's Scan is a simple algorithm for upper hull in two dimensions [35]. It consists of sorting the points by the x-coordinates, followed by a linear scan going right to left using $O(n)$ sidedness tests. It is as follows:

GrahamScan(S)

```
1: sort S by the x-coordinates
2: add p_0 and p_1 to the hull                          ▷ the two rightmost points
3: for each point, p_i, going right to left starting from p_2 do
4:     let h_1 and h_2 be the last and second to last recently added hull points respectively
5:     while h_1, h_2, and p_i form a clockwise turn do
6:         remove h_1 from the hull
7:         if h_2 is the only point left in the hull then
8:             exit the while loop
```

```
 9:         end if
10:         update h_1 and h_2
11:     end while                          ▷ h_1, h_2, and p_i form a counter-clockwise turn
12:     add p_i to the hull
13: end for
```

This uses $1n \log n + O(n)$ x-comparisons and $O(n)$ sidedness tests. The constant factor of 1 is optimal in the worst case. We consider output-sensitive algorithms.

3.3.4 Kirkpatrick and Seidel's Upper Hull Algorithm

The first optimal output-sensitive upper hull algorithm was given by Kirkpatrick and Seidel [41]. Their algorithm is analogous to their two dimensional maxima algorithm, it is a divide and conquer technique called marriage-before-conquest.

First, divide the point set into two equal halves by a vertical line L. Second, find a 'bridge' crossing L; the bridge is an edge of the hull. Third, prune all the point lying underneath the bridge, and finally recursively solve the sub-problems.

Suppose we have a set of points S_1 and S_2 that are separated by a vertical line $L : x = a$; that is, the points of S_1 are right of L and the points of S_2 are left of L. A *bridge* for the sets is an edge with one endpoint from S_1 and the other from S_2 such that the edge is an edge on the upper hull of $S_1 \cup S_2$ (see Figure 3.12). Finding a bridge reduces to linear programming in two dimensions and can be done with algorithms by Megiddo and Dyer using $O(n)$ time [27, 47]. These algorithms use predicates that are neither coordinate comparisons nor sidedness tests. In addition to comparisons and sidedness tests, we also count the number of these 'other predicates' used by our algorithm.

Lemma 3.3.1. *[27, 47] Given the sets S_1 and S_2 and a line $L : x = a$, a bridge can be found in $O(n)$ time in the worst case.*

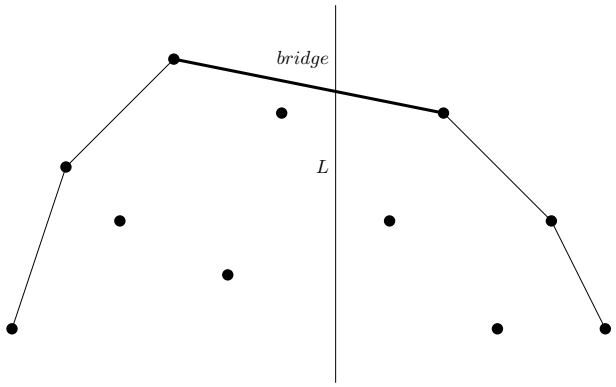

Figure 3.12: A bridge between two upper hulls. This figure is based on one from [41].

KS-UpperHull(S)

1: partition S into two equal-sized subsets S_1 and S_2 by the median x-coordinate, let S_1 be the right half and S_2 be the left half
2: find the bridge e between S_1 and S_2
3: let e_1 be the left endpoint of e
4: remove all the points in S_1 whose x-coordinate $< e_1.x$
5: let e_2 be the right endpoint of e
6: remove all the points in S_2 whose x-coordinate $> e_2.x$
7: output KS-UpperHull(S_1)
8: output e
9: output KS-UpperHull(S_2)

Analysis:

We uses a similar analysis as Section 3.1.2. We inspect the recursion tree. In each iteration, a sub-problem of size n uses $O(n)$ time. At level i, the size of each node is $\frac{n}{2^i}$. The number of nodes at level i is bounded by 2^i and is also bounded by h, due to pruning.

We split the recursion tree into the top ℓ levels and the bottom remaining levels. The top ℓ levels use $\sum_{i=0}^{\ell-1} 2^i O\left(\frac{n}{2^i}\right) = O(n\ell)$ time. The bottom remaining levels use $\sum_{i=\ell}^{\infty} O\left(\frac{n}{2^i}h\right) = O\left(\frac{n}{2^\ell}h\right)$ time. The total cost is $O(n\ell) + O\left(\frac{n}{2^\ell}h\right)$; choosing $\ell = \log h$ gives $O(n \log h)$ total cost.

Thus, the total cost is $O(n \log h)$ time.

3.3.5 Our New Top-Down Algorithm

Our new algorithm is a modification of Kirkpatrick and Seidel's algorithm. The key difference is the use of optimal multi-selection to partition the set into b subsets of equal size.

It is not immediately obvious how to compute the $b-1$ bridges. Chan and Chen in [8] gave an algorithm that computes the bridges between such subsets. Their algorithm is a variant of Graham's Scan and is as follows:

bridges$(S_1, S_2, \ldots, S_b)$

1: $e_1 \leftarrow$ bridge between S_1 and S_2
2: $i_0 \leftarrow 1$
3: $i_1 \leftarrow 2$
4: $j \leftarrow 1$
5: **for** $i = 3, \ldots, b$ **do**
6: **while** $j > 0$ and S_i is not entirely below the line through e_j **do**
7: $j \leftarrow j - 1$ ▷ go back until we get a bridge that form a ccw turn
8: **end while**
9: $e_{j+1} \leftarrow$ bridge between S_{i_j} and S_i ▷ advance and find the next bridge
10: $i_{j+1} \leftarrow i$
11: $j \leftarrow j + 1$
12: **end for**
13: return $e_1, e_2, \ldots, e_j$

Analysis:

Line 1 and 9 computes the bridges between subsets using Lemma 3.3.1. We note that this lemma uses predicates that are neither comparisons nor sidedness tests.

Line 6 takes $O(\frac{n}{b})$ sidedness tests to test if a subset is completely beneath a bridge. Line 9 takes $O(\frac{n}{b})$ time to find a bridge between subsets by Lemma 3.3.1. Each iteration of line 6 and 9 can be associated with a change in the value of j. The value of j changes at most b times. Thus, the algorithm takes $O(b\frac{n}{b}) = O(n)$ time.

Our new algorithm is as follows:

UpperHull(S, i)

1: split S into b_i equal-size subset $S_1, S_2, \ldots, S_{b_i}$ by the $b_i - 1$ quantiles of their x-coordinates
2: compute the bridges between the subsets
3: **for** each subset S_j going right to left **do**

4: prune all the points from S_{j-1} and S_j that are underneath the bridge between S_{j-1} and S_j
5: **end for**
6: **for** each subset S_j going right to left **do**
7: UpperHull$(S_j, i+1)$
8: **end for**

Where $b_i = \frac{r_{i+1}}{r_i}$, $r_i = 2^{i^2}$, and the initial value of i is 0.

Analysis:

We note that in addition to comparisons, we also use other predicates as well (in particular line 2). We can not relate the cost of comparisons to the cost of these other predicates, so we explicitly count both types of operations.

The analysis is similar to Section 3.1.4. We inspect the recursion tree and break the tree into the top ℓ levels and the bottom remaining levels. In each iteration, a node of size n uses $n \log b + O(n)$ expected number of x-comparisons to find the $b-1$ quantiles and $O(n)$ comparisons and other predicates to find the bridges and prune the subsets.

Let ℓ such that $r_{\ell-1} < h \log h \leq r_\ell$.

The top ℓ levels uses the following number of comparisons

$$\begin{aligned}
&\leq \sum_{i=0}^{\ell-1} n \log b_i + O(n) \\
&\leq 1n \log r_\ell + O(n\ell) \\
&\leq 1n \log h + O\left(\sqrt{\log h}\right),
\end{aligned}$$

and the following number of other predicates

$$\begin{aligned}
&\leq \sum_{i=0}^{\ell-1} O(n) \\
&\leq O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The bottom remaining levels use the following number of comparisons and other pred-

icates

$$\begin{aligned}
&\leq \sum_{i=\ell}^{\infty} O\left(h\frac{n}{r}\log r\right) \\
&\leq O\left(h\frac{n}{r_\ell}\log\frac{r_{\ell+1}}{r_\ell}\right) \\
&\leq O\left(n\right).
\end{aligned}$$

Thus, in total our new upper hull algorithm uses $1n\log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons and $O\left(n\sqrt{\log h}\right)$ expected number of other predicates.

We note that by using deterministic multi-selection (refer to Section 3.1.3), we get a deterministic algorithm that uses at most $1n\log h + O\left(n\left(\sqrt{\log h}\right)\log\log h\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ other predicates.

By using the reduction in Section 3.3.2, we obtain a convex hull algorithm with the desired bounds.

Theorem 3.3.2 (Randomized Top-Down 2D Convex Hull)**.** *The convex hull for a set of n points in two dimensions can be computed using a randomized algorithm that uses $1n\log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons and $O\left(n\sqrt{\log h}\right)$ expected number of other predicates, where h is the size of the convex hull.*

Theorem 3.3.3 (Deterministic Top-Down 2D Convex Hull)**.** *The convex hull for a set of n points in two dimensions can be computed using an algorithm that uses $1n\log h + O\left(n\left(\sqrt{\log h}\right)\log\log h\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ other predicates, where h is the size of the convex hull.*

We note that the number of comparisons used by our new algorithms is of a higher order than the number of other predicates used.

3.3.6 Chan's Output-Sensitive Algorithm

Chan presented an output-sensitive algorithm that uses $O\left(n\log h\right)$ time, where h is the output size [4]. His technique can be described as guess-group-and-wrap. It is a bottom-up approach; first partition the point set into groups and solve each group, then use the solutions of the groups to help build the grand solution. His algorithm is as follows: First, we present a subroutine 'Chan-UpperHull-1' and its runtime analysis. Second, we present his complete algorithm and its runtime analysis.

The following subroutine takes as input: a set of n points S, an integer parameter m (whose value will be chosen later), and H is the size of he upper hull of S. The subroutine will return the H points that are the convex hull of S.

Chan-UpperHull-1(S, m, H)

1: partition S into subsets $S_1, S_2, \ldots, S_{\lceil \frac{n}{m} \rceil}$ each of size at most m
2: **for** $i = 1, \ldots, \lceil \frac{n}{m} \rceil$ **do**
3: compute the upper hull of S_i using Graham's scan; note that the points are returned in ccw order
4: **end for**
5: $p_0 \leftarrow (0, -\infty)$ ▷ p's are the hull points
6: $p_1 \leftarrow$ the rightmost point of S
7: **for** $k = 1, \ldots, H$ **do**
8: **for** $i = 1, \ldots, \lceil \frac{n}{m} \rceil$ **do**
9: compute the point $q_i \in P_i$ that maximizes $\angle p_{k-1} p_k q_i$ $(q_i \neq p_k)$ by performing a binary search on the vertices of the upper hull of S_i.
10: **end for**
11: $p_{k+1} \leftarrow$ the point q from $\{q_1, \ldots, q_{\lceil \frac{n}{m} \rceil}\}$ that maximizes $\angle p_{k-1} p_k q$
12: if p_{k+1} is the leftmost point stop and return the upper hull
13: **end for**

Analysis:

We focus on counting the number of comparisons and sidedness tests, ignoring all other operations. Line 3 uses $\frac{n}{m}\left(1m \log m + O\left(m\right)\right) = 1n \log m + O\left(n\right)$ comparisons and sidedness tests. In line 9, each binary search requires $1 \log m + O(1)$ comparisons plus $1 \log m + O(1)$ sidedness tests. So, line 9 uses a total of $H\frac{n}{m}\left(2 \log m + O\left(1\right)\right) = 2H\frac{n}{m} \log m + O\left(H\frac{n}{m}\right)$ comparisons and sidedness tests. If h is known we choose $m = h$ to get the following bound on the number of comparisons and sidedness tests:

$$\begin{aligned} &\leq 1n \log m + O\left(n\right) + 2H\frac{n}{m} \log m + O\left(H\frac{n}{m}\right) \\ &\leq 1n \log h + O\left(n\right) + 2h\frac{n}{h} \log h + O\left(H\frac{n}{h}\right) \\ &\leq 3n \log h + O\left(n\right). \end{aligned}$$

The output size is not known, so we use a guessing trick. We guess the size of the hull at a rate of $H = 2^{2^i}$ and modify the above subroutine so that if more than H hull points are found, immediately terminate and return a failure.

Chan-UpperHull-2(S)

1: **for** $i = 1, 2, \ldots$ **do**
2: $\quad m = H = \min\{2^{2^i}, n\}$
3: $\quad$ result $\leftarrow$ Chan-UpperHull-1(S, m, H)
4: $\quad$ if result is a failure, try again, else return result
5: **end for**

Analysis:

Let ℓ be the iteration in which $2^{2^{(\ell-1)}} < h \leq 2^{2^\ell}$, then $\ell \leq \log\log h + 1$. Let m_i and H_i be the values of m and H in the i^{th} iteration respectively.

The total number of comparisons and sidedness tests used is

$$
\begin{aligned}
&\leq \sum_{i=1}^{\ell} 1n \log m_i + O(n) + 2H_i \frac{n}{m_i} \log m_i + O\left(H_i \frac{n}{m_i}\right) \\
&\leq \sum_{i=1}^{\ell} 3n \log H_i + O(n) \\
&\leq \sum_{i=1}^{\ell} 3n \log 2^{2^i} + O(n) \\
&\leq 3n \sum_{i=1}^{\ell} 2^i + O(n\ell) \\
&\leq 3n 2^{\ell+1} + O(n\ell) \\
&\leq 3n 2^{\log\log h + 2} + O(n \log\log h) \\
&\leq 12n \log h + O(n \log\log h).
\end{aligned}
$$

3.3.7 Our New Bottom-Up Algorithm

We present a new approach for computing the upper hull that uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ sidedness tests. The constant factor of 1 is optimal. We modify our bottom-up algorithm for maxima in two dimensions and combine it with the wrapping portion from Chan's algorithm.

The previous top-down algorithm is randomized and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons and $O\left(n\sqrt{\log h}\right)$ expected number of other predicates, and

its deterministic variant uses $1n \log h + O\left(n\left(\sqrt{\log h}\right) \log\log h\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ other predicates.

Our new bottom-up algorithm is deterministic and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ sidedness tests; in particular, it removes the $\log\log h$ term in the lower order term, and uses only comparisons and sidedness tests.

We again use the idea of an r-grouping. An *r-grouping* is a partitioning of the input into $\lceil \frac{n}{r} \rceil$ groups each of size at most r, together with the upper hull of each group. Clearly, an r-grouping can be computed using $n \log r + O(n)$ comparisons. We note that lines 1-4 of Chan-UpperHull-1 computes an r-grouping, where $r = m$. Using the wrapping portion of Chan-UpperHull-1 (lines 5 - 12), the h-vertex upper hull can be computed from the r-grouping using $O\left(h\frac{n}{r}\log r\right)$ comparisons.

Lemma 3.3.4. *Given an r-grouping and $r' > r$, an r'-grouping can be computed using $1n \log \frac{r'}{r} + O(n)$ x-comparisons and $O(n)$ sidedness tests.*

Proof. The method is the same as that of combining maximas (see Lemma 3.1.6). Recall that each hull is already x-sorted, thus the $\frac{n}{r}$ hulls can be treated as x-sorted lists. Take $\frac{r'}{r}$ hulls and merge them together to produce an x-sorted list of size $\frac{n}{r}$. Combining the lists uses $1n \log \frac{r'}{r} + O(n)$ x-comparisons. Graham's Scan can then be applied to each list to find the upper hull of each list. □

UpperHull(S)

1: **for** $i = 1, 2, \ldots$ **do**
2: compute the r_i-grouping from the previous r_{i-1}-grouping
3: try to compute all the hull points of S using the r_i-grouping; until $r_i < h_i \log h_i$
4: ▷ h_i is the number of hull points found in this iteration
5: **end for**

Where $r_i = 2^{i^2}$.

Analysis:

By a similar method as Section 3.1.5.

Let ℓ such that $r_{\ell-1} < h \log h \leq r_\ell$; thus,

- $\ell < \sqrt{\log(h \log h)} + 1 \in O\left(\sqrt{\log h}\right)$.
- $\log r_\ell \leq \log h + O\left(\sqrt{\log h}\right)$.

- $\log r_{\ell+1} \leq \log h + O\left(\sqrt{\log h}\right)$.

The number of comparisons used in line 2 is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} 1n \log\left(\frac{r_{i+1}}{r_i}\right) + O(n) \\
&\leq 1n \log h + O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The number of sidedness tests used in line 2 is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} O(n) \\
&\leq O\left(n\sqrt{\log h}\right).
\end{aligned}$$

The number of sidedness tests used in line 3 is

$$\begin{aligned}
&\leq \sum_{i=1}^{\ell} O\left(h\frac{n}{r_i}\log r_i\right) \\
&\leq O\left(n\sqrt{\log h}\right).
\end{aligned}$$

In total, the algorithm uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ sidedness tests.

By using the reduction in Section 3.3.2, we obtain a convex hull algorithm with the desired bounds.

Theorem 3.3.5 (Bottom-Up 2D Convex Hull)**.** *The convex hull for a set of n points in two dimensions can be computed using an algorithm that uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ sidedness tests, where h is the size of the convex hull.*

3.3.8 Discussion

Our new top-down algorithm is randomized and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ expected number of comparisons and $O\left(n\sqrt{\log h}\right)$ expected number of other predicates, where h is

the size of the convex hull. The deterministic variant uses $1n \log h + O\left(n\left(\sqrt{\log h}\right) \log\log h\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ other predicates, where h is the size of the convex hull. The cost of these other predicates may be larger than the cost of a sidedness test.

Our new bottom-up algorithm is deterministic and uses $1n \log h + O\left(n\sqrt{\log h}\right)$ comparisons and $O\left(n\sqrt{\log h}\right)$ sidedness tests, where h is the size of the convex hull. In particular, it removes the $\log\log h$ term in the lower order term and uses only comparisons and sidedness tests instead of 'other predicates'.

3.4 The Convex Hull Problem in Three Dimensions

In this section, we present a new randomized algorithm that computes a convex hull of a set of n points in three dimensions using $1n \log n + O(n\sqrt{\log n})$ comparisons. First, we discuss some previous work; Second, we introduce the notion of duality and half-space intersection; Finally, we present our new algorithm.

3.4.1 Previous Work

In three dimensions, Clarkson and Shor gave a randomized algorithm to solve the problem [19]. The first deterministic algorithm was by Chazelle and Matoušek, obtained by derandomizing Clarkson and Shor's algorithm [15]. The approach by Chan, mentioned in Section and described in Section 3.3.6, could be extended to three dimensions to give a simpler algorithm [4].

In higher dimensions, the worst case complexity of the convex hull is $\Theta\left(n^{\lfloor \frac{d}{2} \rfloor}\right)$. Chazelle gave an algorithm that uses $\Theta\left(n \log n + n^{\lfloor \frac{d}{2} \rfloor}\right)$ time and $O\left(n^{\lfloor \frac{d}{2} \rfloor}\right)$ space [12]. Chan showed that an f-face convex hull in a fixed dimension $d \geq 2$ can be constructed in $O\left(n \log f + (nf)^{1-\frac{1}{\lfloor d/2 \rfloor + 1}} \log^{O(1)} n\right)$ time [5].

3.4.2 Projective Space

The projective space is the extension of euclidean space to include points at infinity. Incidence is extended to have the following properties:

- Given any three distinct points, there is exactly one plane incident with them.

- Given any three distinct planes, there is exactly one point incident with them.

Points and planes in projective space are written using homogeneous coordinates. A euclidean point (a, b, c) is written as $(a, b, c, 1)$ in homogeneous coordinates, and is equal to $(\lambda a, \lambda b, \lambda c, \lambda)$ for $\lambda \neq 0$. A point at infinity is written as $(a, b, c, 0)$, for $a, b, c \neq 0$. A plane $ax + by + cz + d = 0$ is written as $[a, b, c, d]$ in homogeneous coordinates, where $a, b, c \neq 0$ if $d = 0$. The plane $[a, b, c, d]$ is equal to $[\lambda a, \lambda b, \lambda c, \lambda]$ for $\lambda \neq 0$.

3.4.3 Duality

In projective space, the dual of the point (a, b, c, d) is the plane $[a, b, c, d]$, and the dual of the plane $[a, b, c, d]$ is the point (a, b, c, d).

The dual transformation $o \to o^*$ is:

1. Incidence-preserving: point p is on plane ℓ if and only if point ℓ^* is on plane p^*.
2. Order-preserving: point p lies above plane ℓ if and only if point ℓ^* lies above plane p^*.

3.4.4 Half-Space Intersection

A *half-space* h is the region $\{(x, y, z) \mid ax + by + cz + d \geq 0\}$. Given a set H of n half-spaces, we want the region of all points that belong to all the half-spaces. The intersection is a convex polyhedron (polygon in two dimensions).

It is well known that the dual of a convex hull is precisely an intersection of half-spaces. Three points p_1, p_2 and p_3 are on the convex hull of a set P if and only if all the points in P are on the same side of the plane ℓ through p_1, p_2 and p_3. In the dual, three planes p_1^*, p_2^* and p_3^* are on the intersection if and only if their intersection point $\ell^* = p_1^* \cap p_2^* \cap p_3^*$ is contained in all the half-spaces of P^*. The adjacency structure of the convex hull is identical to the adjacency structure of the intersection; thus, the problems are equivalent.

If all the half-spaces are upward facing, then the intersection is known as the *upper envelope*, see Figure 3.13.

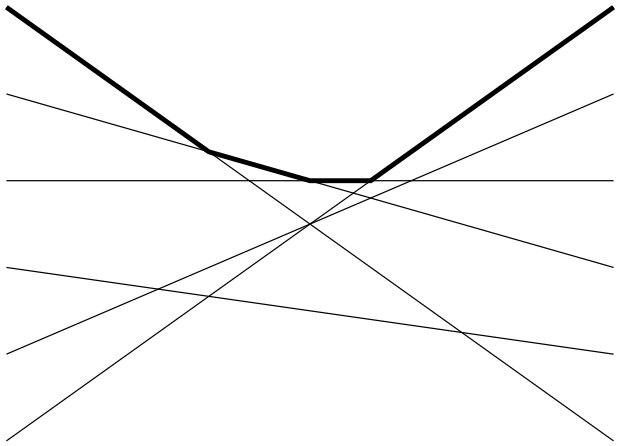

Figure 3.13: An intersection of half-planes forming an upper envelope.

3.4.5 Our New Algorithm in Three Dimensions

Instead of directly computing the convex hull, we apply a projective transformation to reduce the problem to a lower hull problem and then use duality to transform the lower hull problem to an upper envelope problem. The transformations are as follows:

1. Find the point p_0 with the smallest z coordinates.
2. Apply a translation that maps p_0 to the origin, now $p.z \geq 0$ for all points p.
3. Transform the points into homogeneous coordinates.
4. Apply the projective transformation $(x, y, z, 1) \rightarrow (\frac{x}{z}, \frac{y}{z}, \frac{1}{z}, 1)$ if $z \neq 0$ and $(x, y, 0, 1) \rightarrow (x, y, 0, 0)$ if $z = 0$. This projective transformation maps the origin to a point at infinity, and transforms the convex hull to a lower hull with the same adjacency structure.
5. Apply the dual transformation to transform the lower hull to an upper envelope with the same adjacency structure.

For an upper envelope E, we define the vertical decomposition $VD(E)$ to be the decomposition obtained by: first triangulating each face of the polyhedral surface, and then extending each edge of the polyhedron to $z = \infty$. The vertical decomposition consists of several vertical columns and each column is adjacent to exactly one face of the envelope.

Similarly, for a lower hull H, we define the vertical decomposition $VD(H)$ to be the decomposition obtained by: first triangulating each face of the polyhedral surface, and then extending each edge of the polyhedron to $z = \infty$. The vertical decomposition consists of several vertical columns and each column is adjacent to exactly one face of the hull.

The graph of the vertical decomposition represents the adjacency structure. Each cell of the vertical decomposition has a corresponding node in the graph and two nodes in the graph are adjacent if their cells are adjacent in the vertical decomposition.

Lemma 3.4.1. *Given an n half-space upper envelope E, and an query plane q, if there exists a vertex of E that is below q, then at most $1 \log n + O(1)$ comparisons are needed to find such a vertex.*

Proof. Consider the dual lower hull H of E and the projection of H into the xy-plane. The query plane q becomes a query point q^*. Using Seidel and Adamy's query algorithm (see Section 4.1.4) over the projection, the column containing the query point q^* can be located using $1 \log n + O(\sqrt{\log n})$ comparisons. This column contains at most one face f of the hull, and one sidedness test is used to determine if the point is below the hull or not. If q^* is below the hull then f^* is a vertex of E below q. If q^* is above f then there is no vertex of E that is below q. □

Our new algorithm is as follows:

UpperEnvelope(S)

1: take a random sample $R \subset S$ of size r
2: build the point location data structure for the xy-projection of the vertical decomposition of the lower hull of R^*, $VD(R^*)$
3: **for** each $q \in S$ **do**
4: in the dual lower hull, locate the cell Δ that contains q^*
5: in the primal upper envelope, find all the cells of $VD(R)$ that intersect with q by walking, starting from the vertex Δ^*
6: **end for**
7: **for** each $\Delta \in VD(R)$ **do**
8: obtain the conflict list for Δ
9: solve the sub-problem for S_Δ inside Δ
10: **end for**

In line 5, the walk is similar to the walk in the proof of Lemma 3.2.1. The walk finds the cells of $VD(R)$ that intersect q, so that the conflict list can be generated. A walk is

similar to a breath-first or depth-first search with the additional condition that a cell is only visited if it intersects q.

The vertical decomposition of R partitions the upper envelope problem into independent sub-problems. Each sub-problem can be solved independently and the solution of each sub-problem can be glued together to produce the final solution.

Analysis:

The analysis is the same as in Section 3.2.4.

Theorem 3.4.2 (3D Convex Hull)**.** *The convex hull for a set of n points in three dimensions can be computed using a randomized algorithm that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons and $O(n\sqrt{\log n})$ sidedness tests.*

Output-Sensitive Convex Hull

We do not have an analogous result for output-sensitive convex hulls in three dimensions. We were able to give an output-sensitive algorithm for maxima in three dimensions (in Section 3.2.5), but that approach does not apply here.

3.5 Vertical Decompositions in Two Dimensions

In this section, we discuss the problem of computing the vertical decomposition for a set of n horizontal line segments in two dimensions. A *vertical decomposition* for a set of n horizontal line segments in two dimensions is a subdivision of the plane into cells. It is obtained by the following construction: At each endpoint, a ray is extended upward until it hits another horizontal line and a ray is extended downward until it hits another horizontal line (see Figure 3.14).

The graph of the vertical decomposition represents the adjacency structure of the cells. Each cell of the vertical decomposition has a corresponding node in the graph, if two cells are adjacent then their corresponding nodes are adjacent in the grapg.

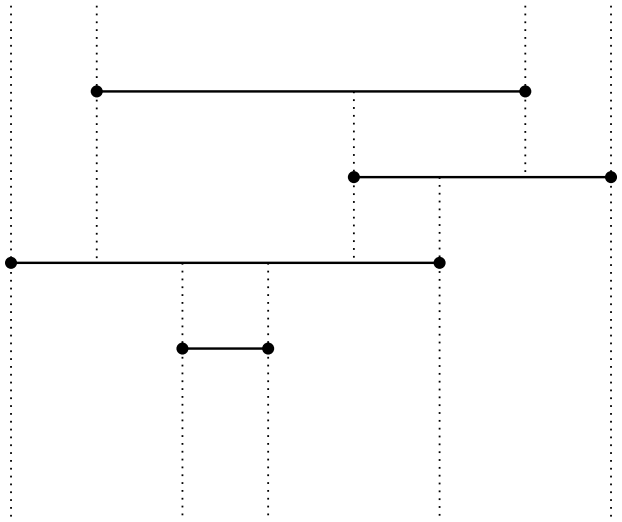

Figure 3.14: A vertical decomposition.

3.5.1 Previous Work

The trivial solution is to sort all the coordinates using $3n \log n + O(n)$ comparisons, afterwards no more comparisons are needed. The standard seep line algorithm uses $2n \log n + O(n)$ comparisons. We use a horizontal sweeping line that sweeps from the top downwards. We maintain data structure that can answer an upward ray shooting query for rays shot from the sweeping line. Such a data structure can be implemented as a sorted array, query can be answered using $1 \log n + O(1)$ comparisons. The sweep line can also be implemented as a self-balancing binary search trees whose height is always bounded by $1 \log n + o(\log n)$.

VD(L)

1: sort the lines by the y-coordinates
2: initialize the sorted array A
3: **for** each line $\ell \in L$ in descending order **do**
4: do a binary search in A for the left endpoint of ℓ, this corresponds to finding the first line hit by an upward ray shot from the left endpoint
5: in A, from the left endpoint walk to the right endpoint, ℓ is the first line by downward rays shot from vertices encountered during the walk
6: remove all the vertices encountered during the walk from A
7: add the left and right endpoints of ℓ to the array

8: **end for**

3.5.2 Our New Algorithm

Our new algorithm is a variant of our randomized algorithm for computing the maxima in three dimensions. By using Clarkson-Shor random sampling and Seidel and Adamy's point location, our algorithm expects to use $1n \log n + O(\sqrt{\log n})$ comparisons.

Our new algorithm is as follows:

VD(L)

1: take a random sample $R \subset L$ of size r
2: compute $VD(R)$
3: build a point location data structure for $VD(R)$
4: **for** each $\ell \in L$ **do**
5: locate a cell Δ that contains an endpoint of ℓ
6: starting from Δ walk along $VD(R)$ to find all the cells that intersect ℓ
7: **end for**
8: **for** each $\Delta \in VD(R)$ **do**
9: obtain the conflict list for Δ
10: solve the sub-problem for S_Δ inside Δ
11: **end for**

The conflict list for a cell $\Delta \in VD(R)$ is the list of line segments that intersect Δ.

In line 6, the walk finds all cells of $VD(R)$ that intersect ℓ, in a similar method to Lemma 3.2.1. The walk is a breath-first or depth-first search in the graph of the vertical decomposition with the additional condition that a cell is only visited if it intersects ℓ.

That is, starting in the cell Δ compare ℓ to all the neighbours of Δ (cell adjacent to Δ in $VD(R)$). For each neighbour that intersects ℓ, visit those neighbours and recurse until all the intersecting cells have been visited. One step of the walk uses δ_Δ comparisons, where δ_Δ is the degree of a cell Δ (number of neighbours in $VD(R)$). The size of the conflict list is $O\left(\sum_{\Delta \in VD(R)} \delta_\Delta |S_\Delta|\right)$; by a Clarkson-Shor analysis its expected value is $O(n)$. Thus, the expected cost to compute the conflict lists is $1n \log r + O(n\sqrt{\log r})$ comparisons.

We note that each sub-problem Δ includes the two line segments of R that define the cell, so that sub-problems can be easily merged together.

Analysis:

The analysis is similar to Section 3.2.4. By choosing $r = \frac{n}{\log n}$, our algorithm uses the following number of comparisons:

$$1n \log n + O\left(n\sqrt{\log n}\right).$$

We note that our algorithm can be extended to disjoint non-orthogonal line segments, where in addition to comparisons, sidedness tests are also used.

Theorem 3.5.1 (Vertical Decompositions in Two Dimensions)**.** *The vertical decomposition for a set of n horizontal line segments in two dimensions can be computed using a randomized algorithm that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.*

3.6 Orthogonal Line Segment Intersection in Two Dimensions

In this section, we discuss the problem of detecting an intersection between a set of n_v vertical line segments and n_h horizontal line segments, where $n_v + n_h = n$. We first give a trivial solution and discuss previous works. Second, present our new algorithm for the decision version of the problem in the orthogonal and general settings. Third, we present our new algorithm for the reporting version of the problem.

3.6.1 The Trivial Solution

The brute-force solution is to compare every pair of segments using $O(n^2)$ comparisons. A slightly better algorithm is to pre-process the segments using three sorts, one on each coordinate. This uses $3n \log n + O(n)$ comparisons; afterward, no more comparisons are needed.

A standard sweep line algorithm by Bentley and Ottmann [54] is:

SegmentIntersection(L)

1: sort all the x-coordinates, that is, the x-coordinates of the two endpoints of a horizontal line segment and the x-coordinate of a vertical line segment. Let P be the resulting list of points.
2: initialise an empty sorted list L to hold the line segments on the sweep line in y-sorted order

```
3: for point p in P going left to right do
4:     if p is the left endpoint of a horizontal line segment ℓ then
5:         add the horizontal line segment ℓ to L
6:     end if
7:     if p is the right endpoint of a horizontal line segment ℓ  then
8:         remove the horizontal line segment ℓ from L
9:     end if
10:    if p an endpoint of a vertical line segment ℓ_v then
11:        find the horizontal line segment ℓ_h that is above its lower endpoint
12:        compare ℓ_h an ℓ_v, if they intersect report it and terminate
13:    end if
14: end for
```

Analysis:

Line 1 uses $(2n_h + n_v)\log n + O(n)$ comparisons. By implementing the sweep line using a perfectly balanced binary search tree or a sorted array, insertions, deletions and predecessor searches use $1\log n + O(1)$ comparisons. Line 5 uses $n_h \log n + O(n_h)$ comparisons. Line 8 uses zero comparisons. Line 11 and 12 use $n_v \log n + O(n_v)$ comparisons. The total cost is $(3n_h + 2n_v)\log n + O(n)$ comparisons. Without loss of generality, we can assume $n_h < n_v$, otherwise we rotate by 90 degrees. In the worst case, $n_h = n_v = \frac{1}{2}n$; thus, at most $2.5n \log n + O(n)$ comparisons are used.

3.6.2 Previous Work

Detecting whether geometric objects intersect is one of the most fundamental classes of problems in computational geometry. There is a large variety of problems with many applications. Some classic variations include: detecting whether a set of n objects intersect; reporting all the intersections among n objects; and counting the number of intersections among n objects. This field of study was initiated by Shamos and Hoey [59].

In two dimensions, there are many classic results. A common point between two convex polygons can be found in $O(\log n)$ time [22]. Given a set of n lines and n points, it can be determined if any point is incident to any line in $O\left(n^{4/3}2^{O(\log^* n)}\right)$ time [46]. Notably, Shamos and Hoey gave the first $O(n \log n)$ time algorithm for determining whether a set of n line segments in the plane intersect [59].

We have presented Bentley and Ottmann's algorithm for detecting whether a set of orthogonal line segments intersect. Their algorithm can be modified to report all intersections in $O(n \log n + k)$ time, where k is the number of intersections, and to count the

number of intersections in $O(n \log n)$ time if the line segments are orthogonal [2]. Their algorithm can also be modified to handle non-orthogonal line segment intersection detection and reporting; but for reporting, the runtime is $O((n+k)\log n)$ [2].

In higher dimensions, intersection among orthogonal boxes can be reported in $O\left(n \log^{d-1} n + k\right)$ time and counted in $O\left(n \log^{d-1} n\right)$ time [28].

For non-orthogonal line segments, reporting can be done in $O(n \log n + k)$ time [13] and counting can be done in $O\left(n^{4/3} \log^{O(n)} n\right)$ time [11].

In the bi-chromatic setting, objects are coloured red or blue, and we only care about intersections between objects of different colours. Given a set of disjoint red line segments and disjoint blue line segments in two dimensions, all red-blue intersections can be reported in $O(n \log n + k)$ time and counted in $O(n \log n)$ time, where n is the total number of line segments [14].

3.6.3 Orthogonal Line Segment Detection

We note that the techniques used in Section 3.2.4 can be applied to produce a randomized algorithm that uses $1n \log n + O\left(n\sqrt{\log n}\right)$ expected comparisons. Our new algorithm is the same as section 3.2.4 but with the following modifications:

- We use the vertical decomposition for line segments as defined in Section 3.5.
- When computing an r-division, we also check if the random sample contains an intersection and terminate if it does.
- When generating the conflict lists in the walking step, also check for intersections.
- We walk using the method described in Subsection 3.5.2.

The new algorithm is as follows:

r-division(S)

1: take a random sample $R \subset S$ of size r
2: check if R contains an intersection
3: build Seidel and Adamy's point location data structure for $VD(R)$
4: **for** each $s \in S$ **do**
5: locate the cell $\Delta \in VD(R)$ that contains the corner of s
6: find all cells intersecting s by walking in $VD(R)$, starting from Δ

7: during the walk, if s crosses a horizontal line of R, terminate
8: **end for**
9: **for** each $\Delta \in VD(R)$ **do**
10: obtain the conflict list $S_\Delta = \{s \in S : s \text{ intersects } \Delta\}$
11: **end for**

SegmentsIntersection(S)
1: build an r-division for S
2: **for** each $\Delta \in VD(R)$ **do**
3: solve the sub-problem for S_Δ inside Δ
4: **end for**

Theorem 3.6.1 (Orthogonal Line Segment Detection in Two Dimensions)**.** *We can determine if a set of n horizontal and vertical line segments intersect in two dimensions using a randomized algorithm that uses $1n \log n + O(n\sqrt{\log n})$ expected number of comparisons.*

3.6.4 General Line Segments

We note that Clarkson-Shor random sampling and Seidel and Adamy's query algorithm can be extended to non-orthogonal line segments. Thus, our algorithm can be extended to the non-orthogonal case. However, instead of comparisons, sidedness tests are used.

Theorem 3.6.2 (General Line Segment Detection in Two Dimensions)**.** *We can determine if a set of n line segments intersect in two dimensions using a randomized algorithm that uses $1n \log n + O(n\sqrt{\log n})$ expected number of sidedness tests.*

3.6.5 Reporting All Intersections

The reporting version of the problem is to return the list of all k intersections. We can modify the approach of Section 3.2.4 to get a randomized algorithm that uses $1n \log n + O\left(n\sqrt{\log n}\right) + O\left(k\right)$ expected number of comparisons.

The vertical decomposition of n intersecting horizontal and vertical line segments is obtained by extending vertical rays from their endpoints. The decomposition has $O\left(n + k\right)$ cells, where k is the number of intersections. Intuitively, we can expect the vertical decomposition of r randomly chosen line segments to have $O\left(r + k\left(\frac{r}{n}\right)^2\right)$ cells; and each cell has a sub-problem of size at most $O\left(\frac{n}{r}\right)$.

Formally, let $VD(R)$ be the vertical decomposition of R, r randomly chosen line segments, by a Clarkson-Shor analysis, we have:

- $E\left[\sum_{\Delta \in VD(R)} \delta_\Delta |S_\Delta|\right] = O\left(r + k\left(\frac{r}{n}\right)^2\right) O\left(\frac{n}{r}\right) = O\left(n + k\frac{r}{n}\right)$, where δ_Δ is the degree of cell Δ.
- $E\left[O\left(\sum_{\Delta \in VD(R)} |S_\Delta| \log |S_\Delta|\right)\right] = O\left(r + k\left(\frac{r}{n}\right)^2\right) O\left(\left(\frac{n}{r}\right) \log\left(\frac{n}{r}\right)\right)$
  $= O\left(n \log \frac{n}{r}\right) + O\left(k\frac{r}{n} \log \frac{n}{r}\right)$.

Our new algorithm is the similar to the previous algorithm for orthogonal line segment detection, but modified to report intersection instead of terminating, it is as follows:

SegmentsIntersectionReporting(L)

1: take a random sample $R \subset L$ of size r
2: report all intersections in R and compute $VD(R)$
3: build a point location data structure for $VD(R)$
4: **for** each $\ell \in L$ **do**
5: locate a cell Δ that contains an endpoint of ℓ
6: starting from Δ walk along $VD(R)$ to find all the cells that intersect ℓ, reporting intersections of ℓ with R as we walk
7: **end for**
8: **for** each $\Delta \in VD(R)$ **do**
9: obtain the conflict list for Δ
10: solve the sub-problem for S_Δ inside Δ
11: **end for**

Where $r = \frac{n}{\log n}$.

Analysis:

Let $k' = k\left(\frac{r}{n}\right)^2$. Lines 2 and 3 uses $O\left((r + k') \log (r + k')\right)$ expected number of comparisons. Line 5 uses $1n \log n + O\left(n\sqrt{\log n}\right)$ comparisons using Seidel and Adamy's query algorithm. Line 6 uses $\sum_{\Delta \in VD(R)} \delta_\Delta |S_\Delta|$ comparisons. Its expected value is $O\left(n + k\frac{r}{n}\right)$. Line 10 uses $O\left(\sum_{\Delta \in VD(R)} |S_\Delta| \log |S_\Delta|\right)$ comparisons; we note that in the orthogonal case we can avoid the $O(k)$ term for reporting. Its expected value is $O\left(n \log \frac{n}{r}\right) + O\left(k\frac{r}{n} \log \frac{n}{r}\right)$ comparisons.

In total, the algorithm uses $1n\log n + O\left(n\sqrt{\log n}\right) + O\left((r+k')\log(r+k')\right) + O\left(n + k\frac{r}{n}\right) + O\left(n\log\frac{n}{r}\right) + O\left(k\frac{r}{n}\log\frac{n}{r}\right)$ comparisons.

Since $r = \frac{n}{\log n}$ and we expect to use the following number of comparisons

$$\begin{aligned}&\leq 1n\log(n+k) + O\left(n\sqrt{\log(n+k)}\right) + O(k)\\ &\leq 1n\log n + O\left(n\sqrt{\log n}\right) + O(k).\end{aligned}$$

If k is small, $1n\log n$ dominates; if k is large, $O(k)$ dominates.

3.6.6 Reducing The Cost of Reporting

If the value of k is known, we can reduce the cost of reporting to $O\left(n\log\left(1+\frac{k}{n}\right)\right)$ comparisons. If $k \in o(n\log n)$ we use our previous reporting algorithm. If $k \in \Omega(n\log n)$ we choose $r = \frac{n^2}{k}$.

In total, the algorithm uses $1n\log n + O\left(n\sqrt{\log n}\right) + O\left((r+k')\log(r+k')\right) + O\left(n + k\frac{r}{n}\right) + O\left(n\log\frac{n}{r}\right) + O\left(k\frac{r}{n}\log\frac{n}{r}\right)$ comparisons.

Since $r = \frac{n^2}{k}$; we have $k' = \frac{n^2}{k}$. We expect the algorithm to use the following number of comparisons:

$$\leq 1n\log n + O\left(n\sqrt{\log n}\right) + O\left(\frac{n^2}{k}\log\frac{n^2}{k}\right) + O(n) + O\left(n\log\left(1+\frac{k}{n}\right)\right).$$

Since $k \in \Omega(n\log n)$, then $\frac{n^2}{k} \in O\left(\frac{n}{\log n}\right)$. Thus, at most $1n\log n + O\left(n\sqrt{\log n}\right) + O\left(n\log\left(1+\frac{k}{n}\right)\right)$ comparisons are used.

3.6.7 Guessing the Number of Intersections

If the value of k is not known in advance, then it is possible to estimate the value of k by taking a random sample. We pick a sample by choosing pairs of lines and testing if they intersect. We choose n pairs from all possible $n_h n_v$ pairs. Let X be the number of such pairs that intersect. Thus $E[X] = n\frac{k}{n_h n_v} = \mu$. Let $\tilde{k} = Xn_h n_v/k$ be our estimator for k; $E[\tilde{k}] = E[X]n_h n_v/k = k\frac{nn_h n_v}{nn_h n_v} = k$.

Furthermore, by a standard Chernoff bound, $X = \Omega(\mu)$, that is $\tilde{k} = \Omega(k)$ with probability at least $1 - e^{-\Omega(\mu)} = 1 - e^{-\Omega(k/n)} > 1 - n^{-\omega(1)}$, if $k \in \omega(n \log n)$. Consequently, $n \log n + \tilde{k} \in \Omega(k)$ with high probability, regardless of the value of k.

We want to minimize $1n \log n + O\left(n\sqrt{\log n}\right) + O\left((r + k') \log (r + k')\right) + O\left(n + k\frac{r}{n}\right) + O\left(n \log \frac{n}{r}\right) + O\left(k\frac{r}{n} \log \frac{n}{r}\right)$, where $k' = k\left(\frac{r}{n}\right)^2$.

We choose $r = \frac{n^2}{n \log n + \tilde{k}}$. Using an analysis that is similar to the previous section's, we conclude that the expected total number of comparisons used is

$$\leq 1n \log n + O\left(n\sqrt{\log n}\right) + O\left(n \log\left(1 + \frac{k}{n}\right)\right).$$

Theorem 3.6.3 (Orthogonal Line Segment Reporting in Two Dimensions)**.** *We can report all intersection among a set of n horizontal and vertical line segments in two dimensions using a randomized algorithm that uses $1n \log n + O\left(n\sqrt{\log n}\right) + O\left(n \log\left(1 + \frac{k}{n}\right)\right)$ expected number of comparisons, where k is the number of intersections.*

Chapter 4

Point Location Data Structures

4.1 Point Location

The point location problem is one of the most fundamental geometric search problems. Given a subdivision of space and a query point, we want to determine the region that contains the query point. It is a well studied problem with many solutions, many variations, and arises naturally in many settings.

For simplicity, we assume a query point intersects a single region, that is a query point is never on the boundary of a region. We focus on the data structure version of the problem, that is, we want to pre-process the objects and create a data structure so that a query can be answered quickly.

In one dimension, the problem is equivalent to predecessor search. In two dimensions, the problem is known as *planar point location* (see Figure 4.1), and the problem naturally extends to higher dimensions.

First, we discuss some previous work, in particular we present Seidel and Adamy's data structure. Second, we discuss binary space partitions and their relationship to point location data structures. Third, we introduce a variant of binary space partitions called multi-way space partitions, and reinterpret Seidel and Adamy's data structure as a multi-way space partition. Fourth, we use multi-way space partitions to construct point location data structures for axis-aligned disjoint boxes and axis-aligned disjoint boxes that form a space-filling subdivision.

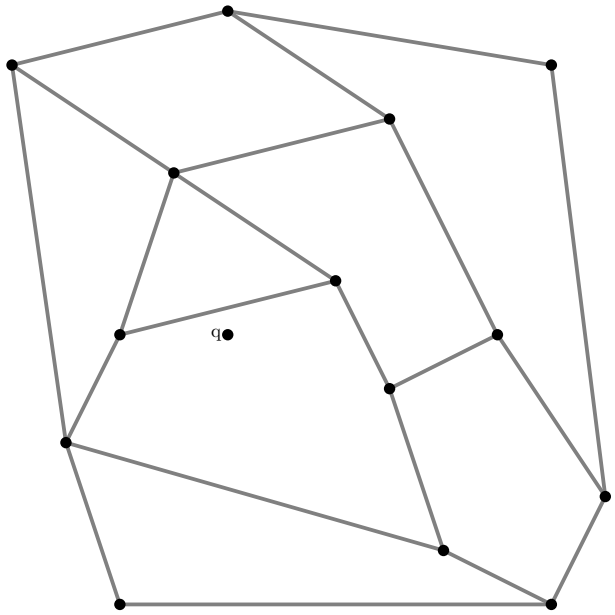

Figure 4.1: Point location in two dimensions.

4.1.1 Previous Algorithms in One Dimension

Predecessor search is solved optimally by binary search on a sorted array. Researchers have investigated the problem in the dynamic setting, where intervals can be added or removed. Self-balancing binary search trees, such as AVL trees and red-black trees [20] solve the problem asymptotically optimally. Search, insert, and delete operations can be done in $O(\log n)$ time. These data structures also use $O(n)$ space and have $O(n \log n)$ preprocessing time.

Randomized search data structures such as treaps [58] and skip-lists [55] can answer queries, perform insertions and deletions in $O(\log n)$ expected time and use $O(n)$ expected space and $O(n \log n)$ expected time to pre-process.

By a standard adversarial argument, any query algorithm needs at least $1 \log n$ comparisons.

4.1.2 Vertical Ray Shooting

Planar point location can be reduced to the related problem of vertical ray shooting. The *vertical ray shooting* problem is defined as follows: Given a set S of n disjoint line segments in the plane and a query point q, determine the first line segment that is immediate above q, that is, the first line segment hit by an upward ray shot from q. This reduction also generalizes to higher dimensions where the line segments become facets on a hyperplane.

4.1.3 Previous Algorithms in Two Dimensions

One of the earliest algorithms is Dobkin and Lipton's slab decomposition method [3]. Vertical lines are drawn at each input vertex, cutting the plane into $O(n)$ vertical slabs (see Figure 4.2). Each slab contains no endpoint and is partitioned into at most $n + 1$ trapezoidal regions. These trapezoidal regions can be linearly ordered by an 'aboveness' relationship. Make a balanced binary search tree for the vertical lines and a balanced binary search tree for each vertical slab. The trees can be stored in $O(n^2)$ total space. Given a query point q, a point location query can be answered using a binary search in the x-axis to determine which slab contains the query point, followed by a binary search on the y-axis within the slab using sidedness tests to determine which trapezoid contains the query point. This query uses $1 \log n + O(1)$ x-comparisons and $1 \log n + O(1)$ sidedness tests.

Lipton and Tarjan reduced the space requirement to $O(n)$ using the planar separator theorem; but, the hidden constants are very large [45]. Other notable approaches include the monotone separating chains with fractional cascading, persistent search trees [56], Preparata's trapezoidal method and Kirkpatrick's triangular refinement [54]. All these approaches use $O(\log n)$ query time and $O(n)$ space. Randomized approaches give data structures that use $O(n)$ expected space, $O(n \log n)$ expected processing time and can answer queries in $O(\log n)$ expected time [3].

In the dynamic setting, there are several notable algorithms. For example, the trapezoidal method can be extended to use $O(\log n)$ query time, $O(\log^2 n)$ update time, and $O(n \log n)$ space [16]. Another algorithm by Goodrich and Tamassia answers queries in $O(\log^2 n)$ time, handles updates in $O(\log n)$ time, and uses $O(n)$ space [34].

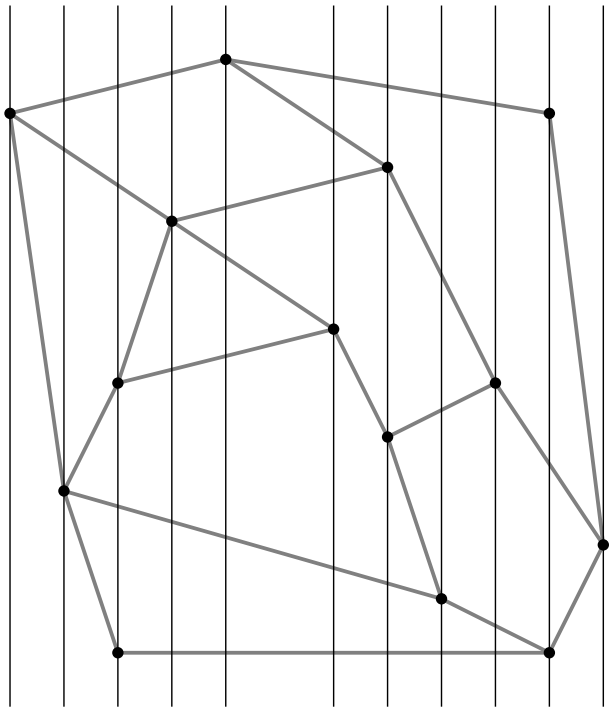

Figure 4.2: Dobkin and Lipton's slab method.

4.1.4 Seidel and Adamy's Point Location Data Structure

As previously mentioned in Section 2.2, Seidel and Adamy presented a planar point location data structure that uses at most $1 \log n + 2\sqrt{\log n} + \frac{1}{2} \log \log n + O(1)$ sidedness tests in the worst case [57]. Many of our results were inspired by their data structure and our data structure for box subdivision use their results as a sub-routine; we discuss their result in the following section.

Weighed Search Trees

The key to Seidel and Adamy's technique is the use of weighted search trees.

A *weighted search tree* is a binary tree where each node i is associated with a weight w_i. The length of a root to leaf path is the sum of the weight of the nodes on path.

Lemma 4.1.1. *Let there be a weighted sequence of n elements, where each element i is associated with weight $w_i \geq 1$, such that $W = \sum_i w_i$. There exists a weighted search*

tree whose leaves are the elements and the length of the root to leaf path for i is at most $\log W - \log w_i + 2$. Such a tree can be constructed in $O(n)$ time.

For a proof see [48].

Seidel and Adamy's Planar Point Location

Similar to Dobkin and Lipton's method they begin with vertical slabs; they place a vertical line on every $\sqrt{n}^{\text{th}}$ endpoint (along the x-axis) making $O(\sqrt{n})$ vertical slabs. They build a balanced binary search tree for the vertical slabs. Thus, locating which slab contains a query point uses $\frac{1}{2}\log n + O(1)$ x-comparisons.

Within a slab, trapezoidal regions are formed by line segments that completely cross a slab. Let e_j be the number of endpoints within the j^{th} trapezoidal region. They assign a weight of $w_j = e_j^2$ to the j^{th} trapezoidal region; and build a weighted search tree, where the trapezoidal regions are the leaves, and the root to leaf paths are of length at most $2 + \log W - \log w_j$. In Figure 4.3, vertical cuts are placed on every third point and free cuts highlighted in black.

Finally, for the j^{th} trapezoidal region (there are e_j endpoints within the j^{th} region), they build the point location data structure by Dobkin and Lipton that uses $2\log e_j + O(1)$ sidedness tests.

Now a query consists of a binary search over the vertical cuts, a binary search in the weighted search tree within a slab, and a point location query within a trapezoid. The total number of sidedness tests used is:

$$
\begin{aligned}
&\leq \frac{1}{2}\log n + O(1) + 2 + \log W - \log w_j + 2\log e_j \\
&\leq \frac{1}{2}\log n + O(1) + \log n - \log e_j^2 + 2\log e_j \quad \text{as} \quad W \leq O(n) \text{ and } w_j = e_j^2 \\
&\leq \frac{1}{2}\log n + O(1) + \log n \\
&\leq \frac{3}{2}\log n + O(1).
\end{aligned}
$$

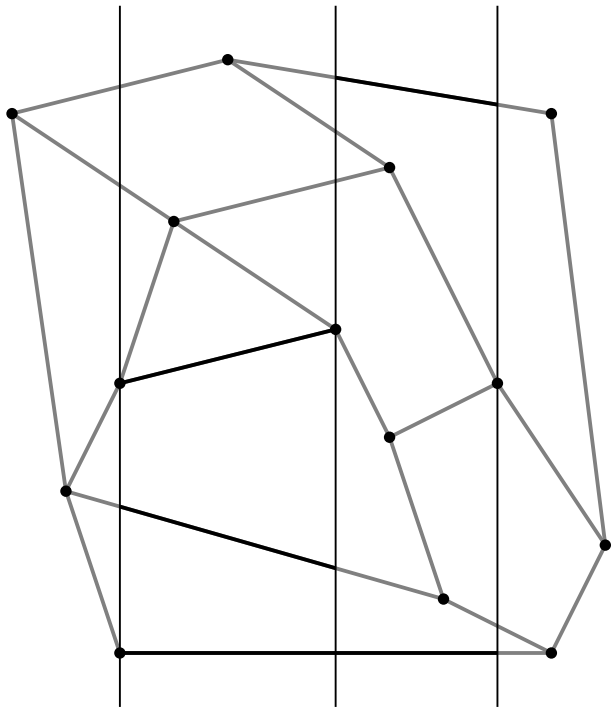

Figure 4.3: An illustration of slabs and free cuts.

Note:

$$
\begin{aligned}
W &\leq \sum_i w_j \\
&\leq \sum_i e_j^2; \quad w_j = e_j^2 \\
&\leq O(\sqrt{n})^2; \quad \sum_i e_j = O(\sqrt{n}), \text{ since there are at most } O(\sqrt{n}) \text{ endpoints in each slab.} \\
&\leq O(n).
\end{aligned}
$$

They note that the number of sidedness tests used within a trapezoid can be reduced from $2\log n + O(1)$ to $\frac{3}{2}\log n + O(1)$ by using their new method instead of Dobkin and Lipton's and a few minor modifications. They place vertical lines on every $n^{\frac{2}{3}\text{th}}$ endpoint to produce $O(n^{\frac{1}{3}})$ slabs each with $n^{\frac{2}{3}}$ endpoints. Similar to above, each slab is divided into trapezoidal regions by free cuts that completely cross the slab. They assign a weight of $w_j = e_j^{\frac{3}{2}}$ to each trapezoidal region and make a weighted search tree. Within each trapezoid

each sub-problem is solved using their previous $\frac{3}{2}\log n + O(1)$ data structure. The total number of sidedness tests used is:

$$\begin{aligned}
&\leq \frac{1}{3}\log n + O(1) + 2 + \log W - \log w_j + \frac{3}{2}\log e_j + O(1) \\
&\leq \frac{1}{3}\log n + O(1) + \log n - \log e_j^{\frac{3}{2}} + \frac{3}{2}\log e_j \quad \text{as} \quad W \leq O(n) \text{ and } w_j = e_j^{\frac{3}{2}} \\
&\leq \frac{1}{3}\log n + O(1) + \log n \\
&\leq \frac{4}{3}\log n + O(1).
\end{aligned}$$

Note:

$$\begin{aligned}
W &\leq \sum_i w_j \\
&\leq \sum_i e_j^{\frac{3}{2}}; \quad w_j = e_j^{\frac{3}{2}} \\
&\leq O(n^{\frac{2}{3}})^{\frac{3}{2}}; \quad \sum_i e_j = O(n^{\frac{2}{3}}), \text{ since there are at most } O(n^{\frac{2}{3}}) \text{ endpoints in each slab.} \\
&\leq O(n).
\end{aligned}$$

They reapply the same technique with different parameters to obtain an even better result. Assume that a data structure of height $\left(1 + \frac{1}{i}\right)\log n + O(i)$ is obtained on the i^{th} iteration. Note when $i = 1$ we use Dobkin and Lipton's data structure. In the $(i+1)^{\text{th}}$ iteration, vertical lines are placed on every $O\left(n^{1-\frac{1}{i+1}}\right)$ endpoint to produce $O\left(n^{\frac{1}{i+1}}\right)$ slabs each with $O\left(n^{1-\frac{1}{i+1}}\right)$ endpoints. Each slab is divided in trapezoidal regions by free cuts that completely cross the slab. Each trapezoid is assigned a weight of $w_j = e_j^{1+\frac{1}{i}}$ and a weighted search tree is made. Within each trapezoid each sub-problem is solved using the previous solution of height $\left(1 + \frac{1}{i}\right)\log n + O(i)$.

The total number of sidedness tests used is:

$$\leq \frac{1}{i+1}\log n + O(1) + 2 + \log W - \log w_j + \left(1+\frac{1}{i}\right)\log e_j + O(i)$$
$$\leq \frac{1}{i+1}\log n + O(i) + \log n - \log e_j^{1+\frac{1}{i}} + \left(1+\frac{1}{i}\right)\log e_j \quad \text{as} \quad W \leq O(n) \text{ and } w_j = e_j^{1+\frac{1}{i}}$$
$$\leq \frac{1}{i+1}\log n + \log n + O(i)$$
$$\leq \frac{1}{i+1}\log n + O(i).$$

Note:

$$\begin{aligned} W &\leq \sum_i w_j \\ &\leq \sum_i e_j^{1+\frac{1}{i}}; \quad w_j = e_j^{1+\frac{1}{i}} \\ &\leq O\left(n^{1-\frac{1}{i+1}}\right)^{1+\frac{1}{i}}; \quad \sum_i e_j = O\left(n^{1-\frac{1}{i+1}}\right), \\ &\qquad\qquad \text{since there are at most } O\left(n^{1-\frac{1}{i+1}}\right) \text{ endpoints in each slab.} \\ &\leq O\left(n^{\frac{i}{i+1}\frac{i+1}{i}}\right) \\ &\leq O(n). \end{aligned}$$

After ℓ iterations, a query structure that uses $\left(1+\frac{1}{\ell}\right)\log n + O(\ell)$ sidedness tests is obtained. By choosing $\ell = O(\sqrt{\log n})$, they get a query structure that uses $\left(1+\frac{1}{\sqrt{\log n}}\right)\log n + O(\sqrt{\log n}) = 1\log n + O(\sqrt{\log n})$ sidedness tests.

Seidel and Adamy's precise upper bound is: $1\log n + 2\sqrt{\log n} + \frac{1}{2}\log\log n + O(1)$ sidedness tests.

Linear Space and a Lower Bound

Seidel and Adamy gave a lower bound of $1\log n + 2\sqrt{\log n} - \frac{1}{2}\log\log n - O(1)$ sidedness tests for the query complexity. Their lower bound matches their upper bound in the first two terms.

The space usage and preprocessing time for their data structure is $O(n2^{\sqrt{\log n}})$, by using sub-structure sharing the space usage can be brought down to $O(n)$ and preprocessing time to $O(n \log n)$ expected time. However, their query cost increases to $1 \log n + 2\sqrt{\log n} + O(\log^{\frac{1}{4}} n)$ sidedness tests.

We note that Kirkpatrick and Seidel removed the $\frac{1}{2} \log \log n$ term from both the upper and lower bounds; thus, their algorithm is optimal in the worst case up to an additive constant [unpublished].

4.1.5 Previous Work in Three and Higher Dimensions

The point location problem has been examined in three dimensions. By using persistence, dynamic planar point location data structures can be used to solve the static three-dimensional problem. We can treat a three-dimensional subdivision as several two-dimensional parallel slices. Goodrich and Tamassia's algorithm can be made persistent; their queries use $O(\log^2 n)$ time and the data structure uses $O(n \log n)$ space and $O(n \log n)$ preprocessing time.

Dobkin and Lipton's slab method can be generalized to three dimensions by projections. The edges of a subdivision are projected into a plane, this forms an arrangement of line segments. Each cell corresponds to a column in three dimensions, each column has potentially n sub-cells ordered by an 'above' relation. A query first locates the cell of the projection containing the query point and then a binary search within the column of the cell finds the sub-cell containing the query point. The query algorithm uses at most $2n \log n + O(\sqrt{\log n})$ sidedness tests, if Seidel and Adamy's query algorithm is used. The space requirement is $O(n^2)$. In higher dimensions, Dobkin and Lipton's method uses $O(2^d d^{O(1)} \log n)$ time to answer a query and $O(n^{2^d})$ space.

In the special case where the regions are axis-aligned disjoint boxes, the trivial data structure uses $d \log n + O(d)$ comparisons and $O(n^d)$ space. At each vertex, we place d axis-aligned hyperplanes, one for each dimension. This divides the space into an $O(n^d)$ sized grid. Since the boxes are axis-aligned, each cell is completely contained by a box. A query consists of d binary searches, one on each of the dimensions. (See Figure 4.4.)

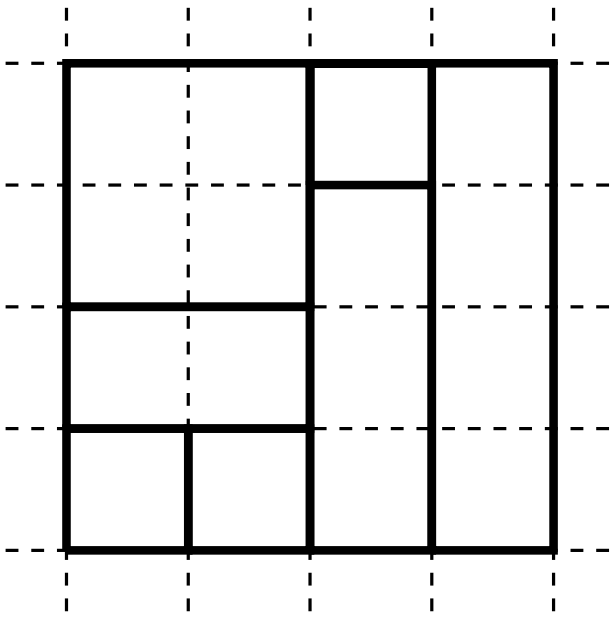

Figure 4.4: The trivial grid solution for axis-aligned disjoint boxes.

4.2 Binary Space Partitioning

In this section, we present binary space partitions and show how constitution a point location data persecution with fast query time reduces to making a binary space partition with small height.

A binary space partition (BSP) is a subdivision of space obtained by recursively dividing space by hyperplane cuts; the subdivision is represented by a tree data structure. Formally, a *binary space partitioning* tree is a binary tree where:

- a *cell* is convex region of $\mathbb{R}^d$ defined by the hyperplane cuts
- each node is associated with a cell;
- a cell u has two children that are obtained by cutting u with a hyperplane;
- the root node of the tree is associated with the entire space;
- the leaves of the tree from a subdivision of the space.

Given a set of objects S, we want the binary space partitioning to subdivide the space so that each leaf intersects at most one object. See Figure 4.5 for an example.

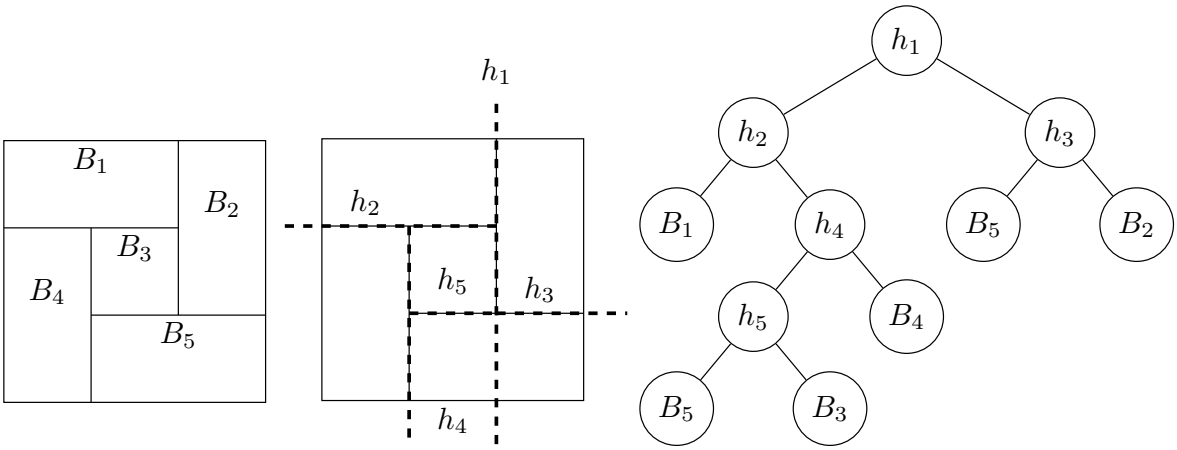

Figure 4.5: An orthogonal subdivision, a BSP, and the corresponding binary tree. This figure is based on one from [37].

4.2.1 Size of a BSP

Binary space partitions are heavily used in computer graphics as they have applications to hidden surface removal, ray tracing, collision detection, to name a few. In this setting, researchers have sought to minimize the size of the binary space partition, that is, the number of nodes in the tree. Minimizing the size allows BSPs to be stored efficiently.

4.2.2 Previous Work

The study of binary space partitioning was initiated by Paterson and Yao. In two dimensions, given a set of n disjoint line segments, Paterson and Yao gave an algorithm to construct a BSP of $O(n \log n)$ size and $O(\log n)$ height [51]. The height of such a BSP on line segments is bounded by $\Omega(\log n)$. Tóth showed that there is a BSP of size at most $O(n\frac{\log n}{\log \log n})$; he also showed a matching lower bound [62, 61]. In the special case of orthogonal line segments, Paterson and Yao gave a BSP of $\Theta(n)$ size [52].

In three dimensions, given n planar facets, Paterson and Yao gave a BSP of size $O(n^2)$ and showed a matching lower bound of $\Omega(n^2)$ [51]. In the special case of orthogonal line

segments or rectangles, Paterson and Yao gave a BSP of size $\Theta(n^{\frac{3}{2}})$ [52]. Hershberger and Suri gave a matching lower bound for the size [36].

In the special case where the objects are orthogonal boxes that form a space-filling subdivision, Hershberger, Suri, and Tóth gave a BSP of size $O(n^{\frac{4}{3}})$ [37]. In three dimensions, the size of such a BSP has a lower bound of $\Omega(n^{\frac{4}{3}})$. Their construction can be extended to higher dimensions; in d dimensions, their BSP is of size $O(n^{\frac{d+1}{3}})$. They also gave a lower bound of $\Omega(n^{\beta(d)})$, where $\beta(d)$ converges to $(1+\sqrt{5})/2$, the golden ratio.

4.2.3 Relationship to Point Location

In the orthogonal setting, we observe that there is a relationship between constructing a data structure for point location and constructing a binary space partitioning.

Observation 4.2.1. *For a set S of orthogonal objects, there exists a point location data structure with a query algorithm that can locate the object containing a query point in at most H comparisons if and only if there exists a binary space partition tree for S with height at most H.*

Proof. The decision tree associated the query algorithm is precisely an orthogonal BSP tree and vice versa. □

Thus, the problem of making a point location data structure with a fast query algorithm reduces to constructing a binary space partitioning tree with a small height.

4.2.4 BSPs with Small Height

There is a large body of work examining the size of binary space partitions. In contrast, there is little work examining the height of BSPs. A lower bound on the size of BSPs immediately implies a lower bound on the height of BSPs.

Lemma 4.2.2. *Any BSP with n nodes has a height of at least $\log n$.*

Proof. A BSP is a binary tree. □

Paterson and Yao's lower bound of $\Omega(n^{\frac{3}{2}})$ on the size of BSPs for orthogonal line segments and rectangles in three dimensions implies a lower bound of $\frac{3}{2}\log n$ on the height. Hershberger, Suri, and Tóth's lower bound of $\Omega(n^{\frac{4}{3}})$ on the size of BSPs for orthogonal subdivisions implies a lower bound of $\frac{4}{3}\log n$ on the height.

4.2.5 Free cuts

In a binary space partitioning, a *free cut* is a hyperplane cut that can divide the set of input objects without dividing any input object into fragments. (In Figure 4.6 the gray face is a free cut.) Free cuts are very useful in constructing BSPs with small size as they do not increase the size complexity of a BSP. However, free cuts do increase the height of a BSP. Many algorithms minimizing the size of a BSP rely heavily on free cuts; as a result, they have poor height bounds. A key observation is that we can construct weight-balanced trees on the parallel free cuts to keep the height small.

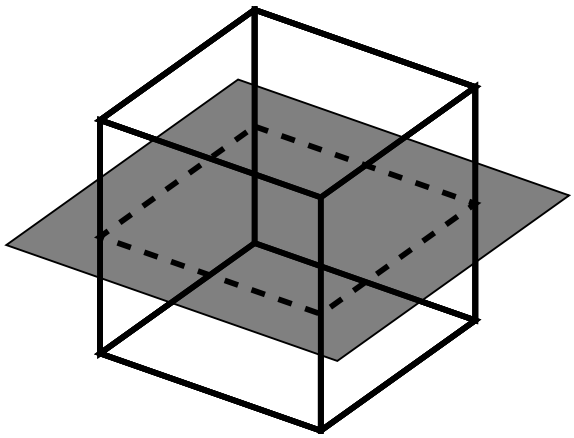

Figure 4.6: A free cut.

4.3 Multi-way Space Partitioning

Inspired by Seidel and Adamy's point location data structure, we describe a new technique called multi-way space partitioning (MSP). The problem of designing BSPs with small height reduces to the problem of designing MSP. In this section, we briefly describe the key to Seidel and Adamy's results and reformulate their idea as multi-way space partitioning. Finally, to illustrate the usefulness our new technique, we re-derive their result using an MSP.

4.3.1 Seidel and Adamy's Point Location

Seidel and Adamy's point location data structure [57] was described in detail in Section 4.1.4. The key to their result was the use of weighted search trees (WST); namely, Lemma 4.1.1. The parameters of the weighted search trees were carefully chosen to make the cost of a query small. Using this method, they achieved a query time of $1 \log n + O\left(n\sqrt{\log n}\right)$ comparisons.

4.3.2 Multi-way Space Partitioning

An MSP is a BSP, except that an MSP tree may have nodes with degrees larger than two. Formally, a *multi-way space partitioning* tree over a set of objects is a multi-way tree, where the degree of a node may be large, satisfying the following conditions:

- each node is associated with a cell;
- a cell u of degree k has k children that are obtained using $k-1$ hyperplane cuts that are disjoint within the cell of u; in the orthogonal setting these cuts are axis-aligned and parallel with each other;
- the root node of the tree is associated with the entire space;
- the leaves of the tree form a subdivision of the space;
- each leaf intersects with at most one object.

Figure 4.7 is an example of a set of disjoint line segments, figure 4.8 is an MSP for the line segments, and figure 4.9 is the corresponding MSP tree.

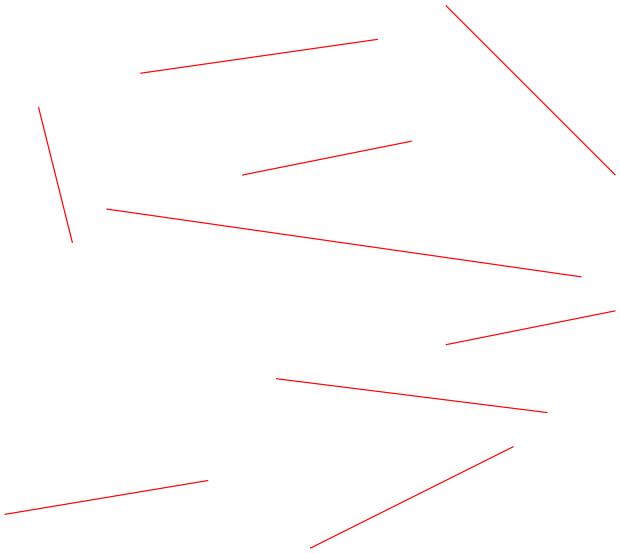

Figure 4.7: A set of disjoint line segments.

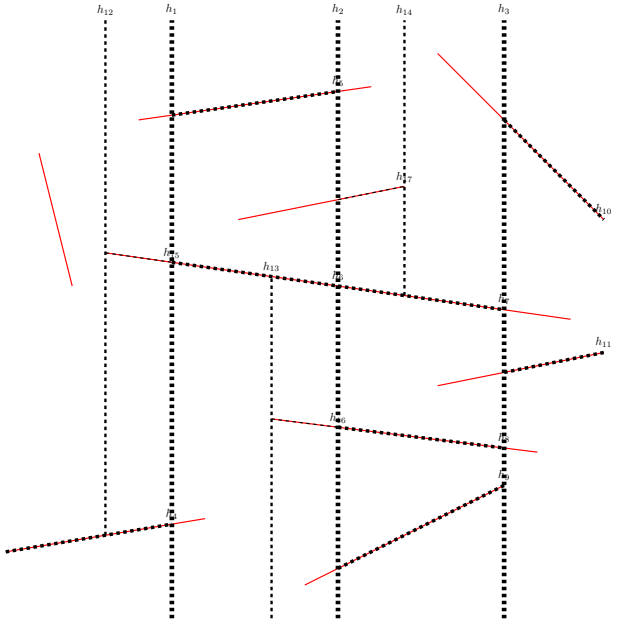

Figure 4.8: An MSP for disjoint line segments.

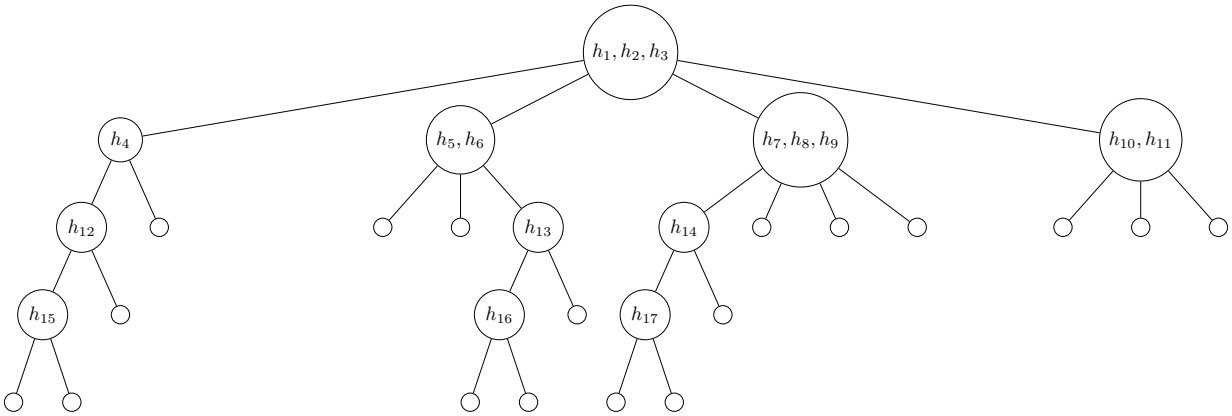

Figure 4.9: The corresponding tree.

Lemma 4.3.1. *Given an MSP tree with W leaves and height H, the MSP tree can be converted to a BSP tree with the same W leaves and height at most* $\log W + O(H)$.

Proof. The key observation is that each node of the MSP tree can be expanded to a binary search tree, while maintaining its balance using a weight-balanced tree. Let u be a node of the MSP tree and $W(u)$ be the number of leaves under u. Denote the children of u as $v_1, v_2, \ldots, v_k$, then $W(u) = W(v_1) + W(v_2) + \cdots + W(v_k)$. By the Lemma 4.1.1, we know there is a WST with root u, leaves $v_1, v_2, \ldots, v_k$ in that order such that the root to leaf v_i path has length at most $\log W(u) - \log W(v_i) + 2$. We replace the outgoing edges of u with the WST, and do this for all internal nodes of the MSP to obtain a binary tree. This binary tree corresponds to a BSP. Figure 4.10 is an example of a WST to binary tree expansion.

Consider a root to leaf path $u_1, u_2, \ldots, u_\ell$ in the MSP. This path is transformed to a path in the BSP with length bounded by

$$\begin{aligned}
&\leq \sum_{j=1}^{\ell} \left(\log\left(W(u_j)\right) - \log W(u_{j+1}) + 2\right) \\
&\leq \log\left(W(u_1)\right) - \log\left(W(u_\ell)\right) + 2\ell \quad \text{by a telescoping sum} \\
&\leq \log W + O(H).
\end{aligned}$$

□

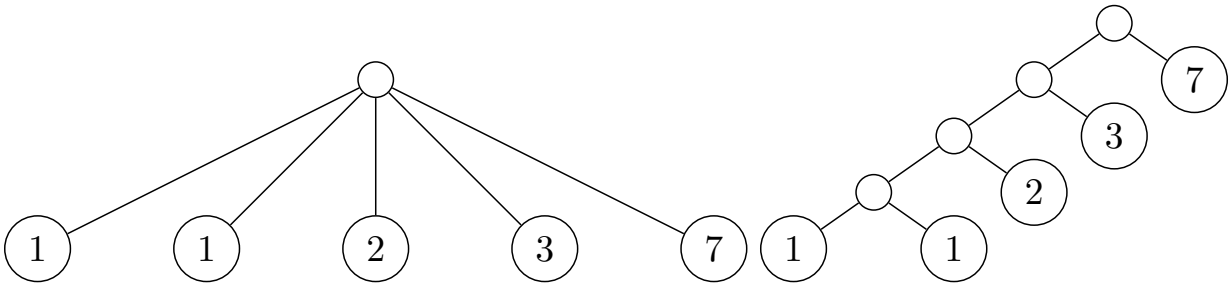

Figure 4.10: A multi-way tree to weight-balanced binary tree expansion.

4.3.3 Seidel and Adamy's Point Location Data Structure as an MSP

We illustrate the usefulness of MSPs by re-deriving Seidel and Adamy's point location data structure. The construction is a simple divide-and-conquer approach.

SeidelAdamy-MSP(S)

1: divide the cell Δ into $r+1$ sub-cells by making r vertical cuts such that each sub-cell has $O\left(\frac{n}{r}\right)$ endpoints; make a node of degree $r+1$
2: divide each sub-cell into sub-sub-cells by applying free cuts, make a node over the free cuts
3: for each sub-problem within the sub-sub-cells, recursively build an MSP tree for that sub-problem

Note: The vertical cuts made in line 1 can be linearly ordered on the x-axis. Similarly, within each sub-cell, the line segments that completely cross a cell can be linearly ordered by an 'aboveness' relation.

Analysis:

We reduce the analysis to a recurrence and then solve it. Let n_i be the number of endpoints in the i^{th} sub-problem, $W(n)$ be the number of leaves of the MSP tree, and $H(n)$ be height of the MSP tree. Each iteration creates at most $O(nr)$ empty leaves; hence, $W(n) \leq \sum_i W(n_i) + O(nr)$. Each iteration adds two levels to the tree; thus, $H(n) \leq \max_i H(n_i) + 2$. At each iteration, r vertical cuts divide the problem in $O(r)$ sub-cells of size at most $O\left(\frac{n}{r}\right)$, thus $\max_i n_i = O\left(\frac{n}{r}\right)$. Each line segment contributes two endpoints, thus $\sum_i n_i \leq 2n$.

Therefore we have

$$W(n) \leq \sum_i W(n_i) + O(nr) \quad \text{and} \quad H(n) \leq \max_i H(n_i) + 2$$

such that $\max_i n_i = O\left(\frac{n}{r}\right)$, $\sum_i n_i \leq 2n$, $W(1) = 1$, and $H(1) = 0$.

The recurrence for $H(n)$ is easy to solve:

$$\begin{aligned} H(n) &\leq \max_i H(n_i) + 2 \\ &\leq H\left(\frac{n}{r}\right) + 2 \\ &\leq O(\log_r n) \\ &\leq O\left(\frac{\log n}{\log r}\right). \end{aligned}$$

The recurrence for $W(n)$ is harder. The critical case is when the sub-problems are equal-sized.

$$\begin{aligned} W(n) &\leq \sum_i W(n_i) + O(nr) \\ &\leq 2rW\left(\frac{n}{r}\right) + O(nr) \\ &\leq 4r^2W\left(\frac{n}{r^2}\right) + O(2nr) \\ &\leq 2^i r^i W\left(\frac{n}{r^i}\right) + O(2^i nr) \\ &\leq 2^\ell r^\ell W(1) + O(2^\ell nr), \quad \text{where } \ell = \left\lceil \frac{\log n}{\log r} \right\rceil \\ &\leq O\left(nr2^{\frac{\log n}{\log r}}\right). \end{aligned}$$

Formally, we guess that the solution is $W(n) \leq c_0 nr2^{c\frac{\log n}{\log r}}$, for some constants c_0 and

c. We show that it is correct by induction.

$$\begin{aligned}
W(n) &\leq \sum_i W(n_i) + O(nr) \\
&\leq \sum_i c_0 n_i r 2^{c\frac{\log n_i}{\log r}} + O(nr) \\
&\leq (\sum_i n_i) c_0 r 2^{c\frac{\log \frac{n}{r}}{\log r}} + O(nr), \quad \text{since } \max_i n_i = O\left(\frac{n}{r}\right) \\
&\leq 2nc_0 r 2^{c\frac{\log \frac{n}{r}}{\log r}} + O(nr), \quad \text{since } \sum_i n_i \leq 2n \\
&\leq c_0 nr 2^{c\frac{\log n}{\log r} - c + 1} + O(nr) \\
&\leq c_0 nr 2^{c\frac{\log n}{\log r}}, \quad \text{for a sufficiently large } c.
\end{aligned}$$

We want to minimize $\log W(n) + O(H(n)) = \log\left(nr2^{\frac{\log n}{\log r}}\right) + O\left(\frac{\log n}{\log r}\right)$, so we choose $r = 2^{\sqrt{\log n}}$.

$$\begin{aligned}
\log W(n) + O(H(n)) &\leq \log\left(nr2^{\frac{\log n}{\log r}}\right) + O\left(\frac{\log n}{\log r}\right) \\
&\leq \log n + \log r + \log 2^{\frac{\log n}{\log r}} + O\left(\frac{\log n}{\log r}\right) \\
&\leq \log n + \log r + O\left(\frac{\log n}{\log r}\right) \\
&\leq \log n + \log 2^{\sqrt{\log n}} + O\left(\frac{\log n}{\log 2^{\sqrt{\log n}}}\right) \\
&\leq \log n + O\left(\sqrt{\log n}\right) + O\left(\frac{\log n}{\sqrt{\log n}}\right) \\
&\leq \log n + O\left(\sqrt{\log n}\right).
\end{aligned}$$

4.4 Point Location Among Axis-Aligned Disjoint Boxes

In this section, we present our new data structures for point location among n axis-aligned disjoint boxes. Our query algorithm uses $\frac{d}{2}\log n + O(\log\log n)$ comparisons in d dimensions. Figure 4.11 is an example of disjoint boxes.

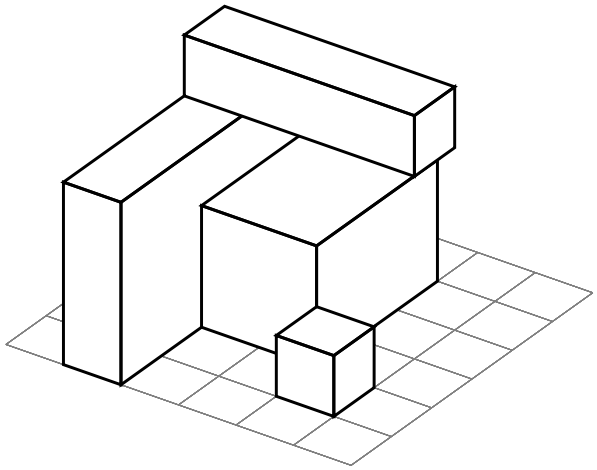

Figure 4.11: Axis-aligned disjoint boxes in three dimensions.

First, we introduce our partitioning scheme, for clarity, we first illustrate the scheme in three dimensions then extend it to d dimensions. Second, we show how the scheme can be reinterpreted as an MSP that can be converted to a BSP with the desired bounds.

4.4.1 Our Partitioning Scheme

Our partitioning scheme is as follows:

Lemma 4.4.1. *Given a set B of n axis-aligned disjoint boxes in d dimensions and a parameter r, we can divide the space into $O\left(r^d\right)$ cells so that the number of j-dimensional facets (j-faces) intersecting a sub-cell is at most $O\left(\frac{n}{r^{d-j}}\right)$, for $j < d$.*

A simple construction was given by Overmars and Yap [50], for the $j = d - 2$ case. Dumitrescu, Mitchell, and Sharir rediscovered and generalized the construction to all j's (see Lemma 7.a in [26]). Chan restated the lemma in its above form as Lemma 4.6 in [7]. For convenience, we provide a brief description of the proof presented in [7].

For clarity, we first illustrate the technique in three dimensions, and then show that it can be easily extended to higher dimensions.

In Three Dimensions

In three dimensions Lemma 4.4.1 becomes:

Lemma 4.4.2. *Given a set B of n axis-aligned disjoint boxes in three dimensions, and a parameter r, we can divide the space into $O(r^3)$ cells so that each sub-cell intersects at most $O(\frac{n}{r})$ facets, $O(\frac{n}{r^2})$ edges, and $O(\frac{n}{r^3})$ vertices.*

Proof. Consider the following partitioning algorithm:

Partition(B)

1: project the 3D boxes into the xy-plane
2: project the 2D plane into the x-axis
3: partition the x-axis with $O(r)$ cuts so that each interval contains at most $O\left(\frac{n}{r}\right)$ vertices
4: lift the x-axis and the cuts back into the xy-plane ▷ this results in $O(r)$ columns
5: **for** each column γ **do**
6: partition γ using $O(r)$ line segments (orthogonal to the y-axis) so that each cell contains at most $O\left(\frac{n}{r^2}\right)$ vertices
7: partition γ using $O(r)$ additional line segments (orthogonal to the y-axis) so that each cell intersects at most $O\left(\frac{n}{r}\right)$ horizontal edges (edges orthogonal to the y-axis)
8: **end for**
9: lift the xy-plane back into 3D ▷ this results in $O(r^2)$ columns
10: **for** each column γ **do**
11: partition γ using $O(r)$ planes (orthogonal to the z-axis) so that each cell contains at most $O\left(\frac{n}{r^3}\right)$ vertices
12: partition γ using $O(r)$ additional planes (orthogonal to the z-axis) so that each cell intersects at most $O\left(\frac{n}{r^2}\right)$ horizontal edges (edges orthogonal to the z-axis)
13: partition γ using $O(r)$ additional planes (orthogonal to the z-axis) so that each cell intersects at most $O\left(\frac{n}{r}\right)$ xy-facets (facets orthogonal to the z-axis)
14: **end for**

Clearly, the cuts produce $O(r^3)$ cells. It is also clear that each cell contains at most $O(\frac{n}{r^3})$ vertices.

After line 6 executes, each cell intersects at most $O(\frac{n}{r})$ horizontal edges (orthogonal to y). Observe that each vertical edge (orthogonal to x) was lifted from a single vertex in the x-axis; thus, the number of vertical edges intersecting a cell is bounded by the number of vertices in the column. The number of vertices is bounded by $O(\frac{n}{r})$, due to the cuts in line 3. Therefore, the number of edges intersecting a cell is bounded by $O(\frac{n}{r})$.

Similarly, after line 9 executes, each cell has at most $O(\frac{n}{r^2})$ horizontal edges (orthogonal to z). Observe that vertical edges (parallel to z) were lifted from endpoints in the xy-plane; there are at most $O(\frac{n}{r^2})$ endpoints in a column. Therefore, the number of edges intersecting a cell is bounded by $O(\frac{n}{r^2})$.

A similar argument holds for facets. Line 10 reduces the number of xy-facets to $O(\frac{n}{r})$. The xz-facts and yz-facets were lifted from edges in the xy-plane. The number of such edges in a column is bounded by $O(\frac{n}{r})$. Thus, each cell has at most $O(\frac{n}{r})$ facets.

□

We note that within each round, the cuts are parallel to each other; thus, the partitioning scheme produces an MSP of height 3.

Figure 4.12 is an example of a series of projections, figure 4.13 is an example of a series of cuts in two dimensions, and figure 4.14 is an example of a cut in three dimensions.

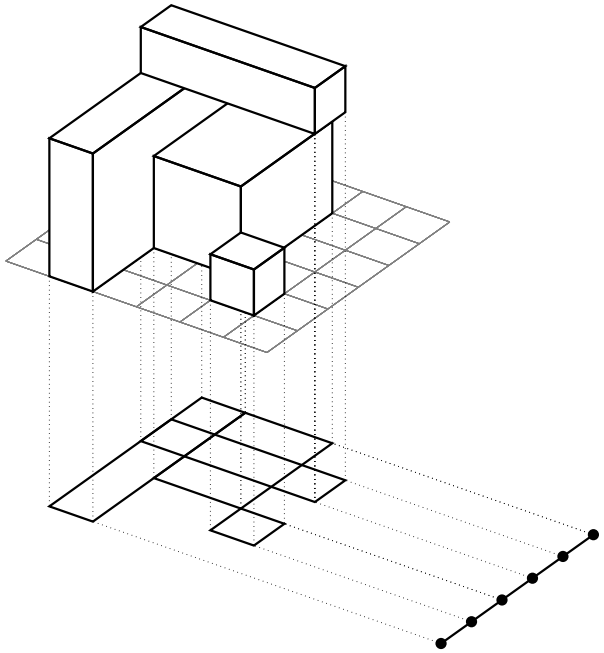

Figure 4.12: Projection of the 3D boxes into the xy-plane and the x-axis.

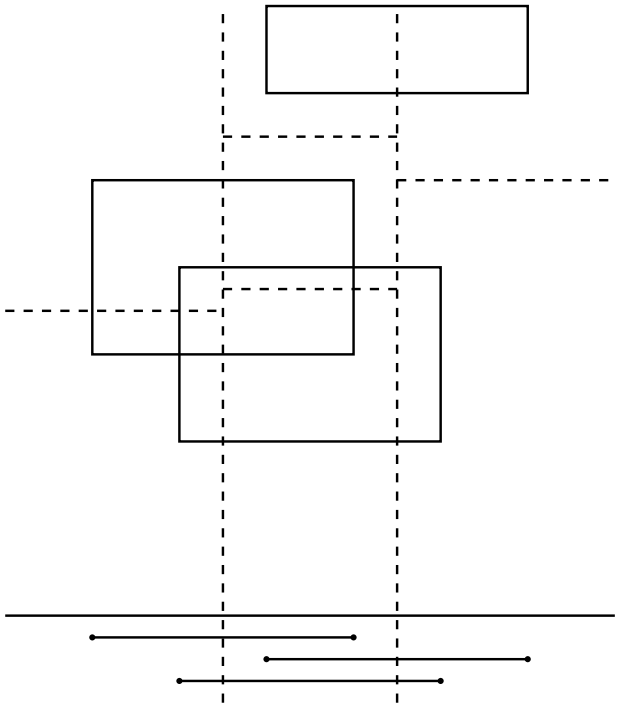

Figure 4.13: Cuts in Two Dimensions.

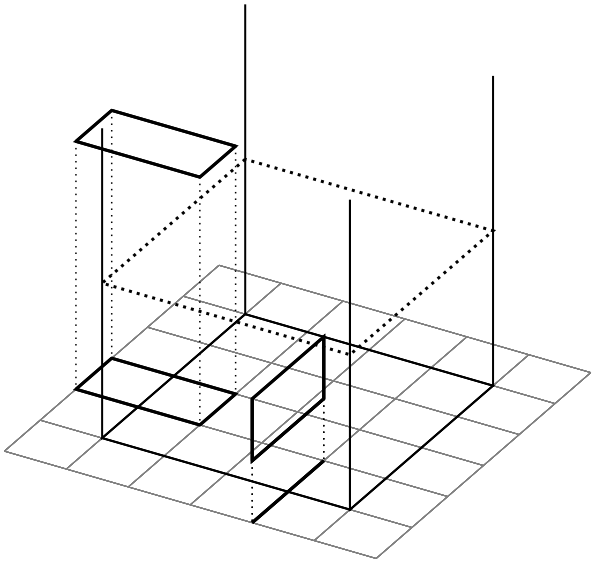

Figure 4.14: A cut in Three Dimensions.

In Higher Dimensions

We begin by proving Lemma 4.4.1.

Proof. The following was adapted from Lemma 4.6 in [7].

partition(S,d)

1: project the d-dimensional input into $d-1$ dimensions
2: partition(S,$d-1$)
3: the partition of the $(d-1)$-dimensional projection has $O\left(r^{d-1}\right)$ cells; each cell has at most $O\left(\frac{n}{r^{d-j}}\right)$ j-faces, for $j < d-1$
4: **for** each cell Δ of the partition **do**
5: lift Δ back into d dimensions, this results in a vertical column γ
6: partition γ with $O(r)$ hyperplane cuts orthogonal to the d^{th} axis, so that in each interval the number of j-faces orthogonal to the d^{th} dimension is reduced by a factor of $O(r)$, for all $j < d-1$
7: **end for**
8: ▷ now, we have $O\left(r^d\right)$ cells where each cell has at most $O\left(\frac{n}{r^{d-j}}\right)$ j-faces, for $j < d$

Let n_i^d be the maximum number of i-faces intersecting each cell during the d-dimensional round. If an i-face is orthogonal to the d^{th}-axis then the vertical projection is an i-flat; in the d^{th}-round the number of i-faces of this type is reduced by a factor of r. If an i-flat is not orthogonal to the d^{th} axis, then its vertical projection is a $(i-1)$-flat. Therefore, we have $n_i^d \leq n_{i-1}^{d-1} + \frac{n_i^{d-1}}{r}$; by induction, $n_i^d \in O\left(\frac{n}{r^{d-i}}\right)$.

□

4.4.2 The MSP

The construction process of Lemma 4.4.1 uses many parallel cuts, thus it can be reinterpreted as an MSP. Our MSP is constructed is as follows:

MSP(S)

1: apply Lemma 4.4.1 dividing the space into cells; make an MSP tree for these cuts
2: **for** each cell Δ **do**
3: **for** each $(d-1)$-face f that completely crosses Δ **do**
4: apply a free cut along f
5: **end for**
6: make a multi-way node for these free cuts

7: **end for**
8: recursively build an MSP tree within each cell

Analysis:

Since the boxes are disjoint, the free cuts are parallel to each other, so they form an MSP.

There are at most $O\left(\frac{n}{r}\right)$ such free cuts in a cell. Applying these free cuts increases the number of cells to at most $O\left(r^d \frac{n}{r}\right) = O\left(nr^{d-1}\right)$.

In order for a box B to intersect the i^{th} cell Δ_i, either: A d-face of B completely covers Δ_i and no $(d-1)$-face of B intersects Δ_i, in which case the sub-problem is trivial. A $(d-1)$-face of B intersects Δ_i and no $(d-2)$-face of B intersects Δ_i, in which case the $(d-1)$-face of B was used as a free cut, thus this case is not possible. The final possibility is a $(d-2)$-face of B intersects Δ_i. Thus, the number of boxes intersecting a cell is bounded by $O\left(\frac{n}{r^2}\right)$, the number of $(d-2)$-faces.

Free cuts do not partition any box, so the total number of boxes remains bounded by $O\left(r^d\right) O\left(\frac{n}{r^2}\right) = O\left(nr^{d-2}\right)$.

Let $W(n)$ be the number of leaves in the MSP, $H(n)$ be the height of the MSP, and n_i be the number of boxes in the i^{th} sub-problem.

Each iteration constructs an MSP of height $O(1)$ and $O\left(nr^{d-1}\right)$ leaves. Thus,

$$W(n) \leq \sum_i W(n_i) + O\left(nr^{d-1}\right) \quad \text{and} \quad H(n) \leq \max_i H(n_i) + O(1)$$

such that $\max_i n_i = O\left(\frac{n}{r^2}\right)$, $\sum_i n_i = O\left(r^d \frac{n}{r^2}\right)$, $W(1) = 1$ and $H(1) = 0$.

We choose $r = n^\epsilon$.

The recurrence for height is easy to solve:

$$\begin{aligned} H(n) &\leq \max_i H(n_i) + O(1) \\ &\leq H\left(\frac{n}{r^2}\right) + O(1) \\ &\leq H\left(n^{1-2\epsilon}\right) + O(1) \\ &\leq O(\log \log n). \end{aligned}$$

The recurrence for the number of leaves is harder to solve; we first sketch a proof for the case when the sub-problems are equal-sized to provide intuition.

$$\begin{aligned} W(n) &\leq \sum_i W(n_i) + O\left(nr^{d-1}\right) \\ &\leq O\left(r^d\right) W\left(\frac{n}{r^2}\right) + O\left(nr^{d-1}\right). \end{aligned}$$

If r is a large constant, then $W(n) \in O\left(n^{\frac{d}{2}+\epsilon}\right)$, by the "master method" [20]. For $r = n^{\epsilon}$, the bound becomes slightly better.

Formally, we guess that the solution is $W(n) \leq c_0 n^{\frac{d}{2}} \log^c n$, for some constants c_0 and c. We show that it is correct by induction.

$$\begin{aligned} W(n) &\leq \sum_i W(n_i) + O\left(nr^{d-1}\right) \\ &\leq \sum_i c_0 n_i^{\frac{d}{2}} \log^c n_i + O\left(nr^{d-1}\right) \\ &\leq \sum_i c_0 n_i \left(n_i^{\frac{d}{2}-1}\right) \log^c n_i + O\left(nr^{d-1}\right) \\ &\leq \sum_i c_0 n_i \left(\frac{n}{r^2}\right)^{\frac{d}{2}-1} \log^c \left(\frac{n}{r^2}\right) + O\left(nr^{d-1}\right), \quad \text{since } \max_i n_i = O\left(\frac{n}{r^2}\right) \\ &\leq c_0 c' \left(r^d \frac{n}{r^2}\right) \left(\frac{n}{r^2}\right)^{\frac{d}{2}-1} \log^c \left(\frac{n}{r^2}\right) + O\left(nr^{d-1}\right), \quad \text{since } \sum_i n_i \leq c' r^d \frac{n}{r^2} \text{ for some } c' \\ &\leq c_0 c' n^{\frac{d}{2}} \log^c \left(\frac{n}{r^2}\right) + O\left(nr^{d-1}\right) \\ &\leq c_0 c' n^{\frac{d}{2}} \log^c \left(n^{1-2\epsilon}\right) + O\left(nr^{d-1}\right) \\ &\leq \left(c_0 c' n^{\frac{d}{2}} \log^c n\right) (1-2\epsilon)^c + o\left(n^{d/2}\right) \\ &\leq c_0 n^{\frac{d}{2}} \log^c n, \quad \text{for a sufficiently large } c. \end{aligned}$$

By applying Lemma 4.3.1, we produce a BSP of height

$$\begin{aligned} H &\leq \log\left(W\left(n\right)\right) + O\left(H\right) \\ &\leq \log\left(n^{\frac{d}{2}}\log^{c} n\right) + O\left(\log\log n\right) \\ &\leq \frac{d}{2}\log n + O\left(\log\left(\log^{c} n\right)\right) + O\left(\log\log n\right) \\ &\leq \frac{d}{2}\log n + O\left(\log\log n\right). \end{aligned}$$

Theorem 4.4.3 (Point Location Among Axis-Aligned Disjoint Boxes)**.** *Given a set of n axis-aligned disjoint boxes, there is a point location data structure that has a query algorithm that uses $\frac{d}{2}\log n + O\left(\log\log n\right)$ comparisons.*

4.4.3 Lower Bounds

Hershberger and Suri showed that the lower bound of the size of a BSP for axis-aligned disjoint boxes in three dimensions is $\Omega\left(n^{\frac{3}{2}}\right)$ [36]. Thus, the height of any such BSP is atleast $\frac{3}{2}\log n - \Theta(1)$. Observation 4.2.1 implies that $\frac{3}{2}\log n - \Theta(1)$ comparisons is a lower bound for any point location query algorithm. Thus, our query algorithm is optimal in the leading constant in three dimensions.

4.5 Point Location Among Axis-Aligned Disjoint Boxes that Form A Space-Filling Subdivision

In the special case where the boxes form a space-filling subdivision, the query algorithm can be improved to use at most $\frac{d+1}{3}\log n + O\left(\log\log n\right)$ comparisons in d dimensions. The extra ingredient is to use a technique by Hersberger, Suri, and Tóth [36] as an additional base case combined by our MSP reinterpretation of Seidel and Adamy's result.

First, we introduce our partitioning scheme, for clarity we first illustrate the scheme in three dimensions then extend it to higher dimensions. Second, we show how the scheme can be reinterpreted as an MSP that can be converted to a BSP with the desired bounds.

4.5.1 Our Partitioning Scheme

We reuse the partitioning scheme from the previous section; in particular we use Lemma 4.4.1 and apply free cuts along faces that completely cross a cell. In addition, when a cell has no vertices in its interior, we use the following lemma by Hershberger, Suri and Tóth to make a smaller partition.

Lemma 4.5.1. *Given a subdivision S of n axis-aligned disjoint boxes inside a cell Δ where no vertices intersect the interior of Δ, there is an MSP for S with $n2^{O\left(\sqrt{\log n}\right)}$ leaves and height $O\left(\sqrt{\log n}\right)$.*

The idea behind the proof is due to Hershberger, Suri and Tóth, who stated that in this case, there is a BSP of $O(n)$ size. Hershberger, Suri and Tóth did not focus on the height of the BSP. We observe that their construction can be modified to give an MSP with the stated bounds. We note that the MSP can be converted to a BSP with a size of $O(n)$ and height of $1 \log n + O(\sqrt{\log n})$.

The proof is lengthy; first we define some useful terms and prove a few related lemmas. For clarity, we first illustrate the proof in three dimensions, then show that it can be extended to d dimensions.

A box B in a cell Δ is called a *k-rod* of Δ if the extent of B is the same as the extent of Δ in exactly k dimensions; these k dimensions are the orientation of B. That is, the interval spanned by B is exactly the same as the interval spanned by Δ in exactly k dimensions. (See Figure 4.15.)

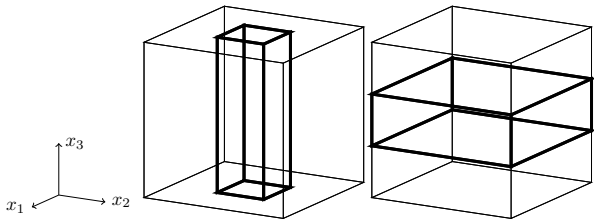

Figure 4.15: A 1-rod of orientation $\{x_3\}$ on the left and a 2-rod of orientation $\{x_1, x_2\}$ on the right. This figure is based on one from [36].

Three Dimensions

Lemma 4.5.2. *In a three-dimensional subdivision, if a cell Δ consists of 1-rods then it cannot have 1-rods of all three orientations.*

Proof. This lemma was originally presented by Hershberger and Suri in [36] as Lemma 3.3; for clarity we restate their proof.

By contradiction, suppose A is a 1-rod with orientation $\{1\}$, B is a 1-rod with orientation $\{2\}$, and C is a 1-rod with orientation $\{3\}$. Let ℓ_A be the line through the center of A, ℓ_B be the line through the center of B, and ℓ_C be the line through the center of C. Let P_{AC} be the axis-aligned plane containing ℓ_A intersecting ℓ_C, P_{CB} be the axis-aligned plane containing ℓ_C intersecting ℓ_B, and P_{BA} be the axis-aligned plane containing ℓ_B intersecting ℓ_A. The three panes are not parallel with each other thus intersect at a point $p = P_{AC} \cap P_{CB} \cap P_{BA}$. Consider the point p, every 1-rod with orientation $\{1\}$ through p intersects C, every 1-rod with orientation $\{2\}$ through p intersects A, and every 1-rod with orientation $\{3\}$ through p intersects B. Thus, it is not possible for any 1-rod to contain p, since S is a subdivision of rods, this is a contradiction. (See Figure 4.16.) □

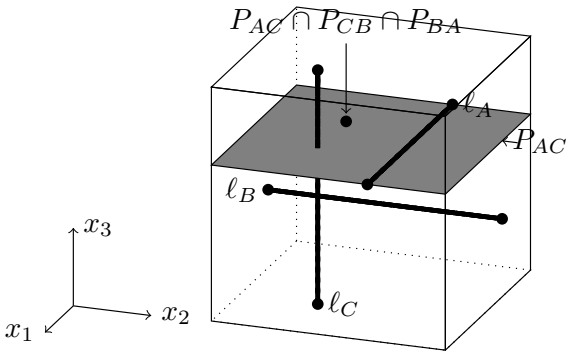

Figure 4.16: 1-rods with three different orientations imply the existence of an interior vertex. This figure is based on one from [36].

Lemma 4.5.3. *If a cell Δ consists of 1-rods of two possible orientations, then there exist parallel free cuts that partition the cell into sub-cells where each sub-cell consists of 1-rods that are singly oriented.*

Proof. This lemma was originally presented by Hershberger and Suri in [36] as Lemma 3.4. Their focus was on producing a BSP with small size. For clarity, we modify their proof to suit our purposes.

It is sufficient to show that there exists at least one free cut that partitions the cell into two smaller sub-cells. Within each sub-cell, the process can be applied recursively until the sub-cells consist of singly oriented rods.

Let rods A and B be two rods of different orientations d_1 and d_2, respectively, that share a common face f. Consider the plane p containing f, p is parallel to dimensions d_1 and d_2 and orthogonal to dimension d_3. Suppose there is a rod r whose interior intersects plane p; r cannot have orientation d_3 by Lemma 4.5.2. If r has orientation d_1, it intersects B, if r has orientation d_1 it intersects A. Thus, r cannot exist, and p is a free cut. (See Figure 4.17.) □

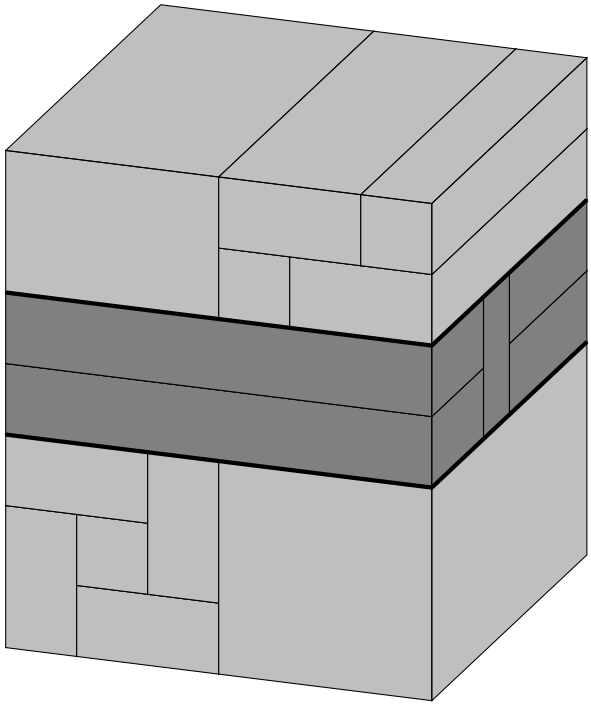

Figure 4.17: 1-rods with two different orientations imply the existence of a free cut. This figure is based on one from [36].

We are now ready to prove Lemma 4.5.1 in three dimensions.

Proof. If there are no vertices in the interior of a cell Δ, all the boxes of Δ are rods. There are three types of rods: 1-rods, 2-rods, and 3-rods. If a cell has a 3-rod, the 3-rod covers the entire cell and the problem is trivial. If a cell Δ contains a 2-rod, r, the faces of r that are interior of Δ are free cuts, these free cuts further reduce the problem. We repeatedly apply free cuts along 2-rods until the sub-problems consist solely of 1-rods. If a cell Δ consists of 1-rods, then by Lemmas 4.5.2 and 4.5.3 we can apply free cuts until the sub-problems have

a single orientation. We project the singly oriented sub-problems down to two dimension and apply Seidel and Adamy's results.

We construct the MSP tree as follows: We make an MSP node over the free cuts along 2-rods and those of Lemma 4.5.3, until each sub-cell consists of singly oriented 1-rods. For those sub-cells of singly oriented 1-rods, we make an MSP using Seidel and Adamy's data structure. In the worst case, this gives an MSP tree of $n2^{O\left(\sqrt{\log n}\right)}$ leaves and height $O\left(\sqrt{\log n}\right)$.

□

In Higher Dimensions

We now prove Lemma 4.5.1 in d dimensions.

Lemma 4.5.4. *Let B_1 and B_2 be any two rods of the subdivision, let their orientations be D_1 and D_2 respectively, $D_1, D_2 \subset \{1, 2, \ldots, d\}$. The following is not possible: $D_1 \cup D_2 = \{1, 2, \ldots, d\}$.*

Proof. This lemma was originally presented by Hershberger and Suri in [36] as Lemma 4.1; for clarity, we restate their proof.

Let ℓ_i be the extent of dimension i that is common to both B_1 and B_2. Since for every dimension, at least one rod extends the entire dimension, ℓ_i is non-empty for all dimensions. Consider the rod $b = \ell_1 \times \ell_2 \times \cdots \times \ell_d$; b lies in the interior of B_1 by construction and b lies in the interior of B_2 by construction. Therefore, B_1 and B_2 intersect; thus, the rods are not a subdivision. □

Consider the set system of the orientations of $(d-2)$-rods in a subdivision. Denote the system as $S = (D, \mathcal{F})$ where the dimensions $D = \{1, 2, \ldots, d\}$ form the base set, and $\mathcal{F}$ are the orientations of $(d-2)$-rods. We note that for all $F \in \mathcal{F}$, $|F| = d-2$. Observe that the complement of the set system is a graph $G = (D, E)$, where $E = \{D - F \mid F \in \mathcal{F}\}$. It is easier to reason with G than S.

Lemma 4.5.5. *In a box-subdivision consisting of $(d-2)$-rods, the graph $G = (D, E)$ is a star-graph. A* star-graph *is a graph where all edges share one common vertex v.*

Proof. This lemma was originally presented by Hershberger and Suri in [36] as Lemma 4.2; for clarity, we restate their proof with some slight modifications.

Let B_1 and B_2 be any two $(d-2)$-rods, by lemma 4.5.4, their orientations cannot cover D, thus the complements of their orientations have a common intersection. This implies that in G every pair of edges must have a vertex in common. Clearly, if $|E| > 3$ then the graph is a star-graph. If $|E| = 1$ or 2 then G is a star-graph.

If $|E| = 3$, if G is not a star-graph, it must be a 3-cycle by Lemma 4.5.4. We name the dimensions d_1, d_2, and d_3. Consider all the dimensions not in the 3-cycle, all the rods of the subdivisions extent the entire range of the dimensions. We project the subdivision down to d_1, d_2, and d_3. This gives us a subdivision consisting of 1-rods of all three orientations contradicting Lemma 4.5.2. Thus the graph must be a star-graph. (See Figure 4.18 and Figure 4.19.) □

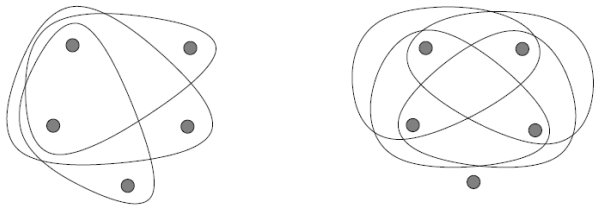

Figure 4.18: A set system in a cycle configuration (left) and a set system in a star configuration (right). This figure is based on one from [36].

Figure 4.19: A cycle graph (left) and a star graph (right).

Lemma 4.5.6. *Let G be the graph for a box subdivision consisting of $(d-2)$-rods. If G is a star-graph with central vertex v and at least two edges, then there is a free cut along dimension v.*

Proof. This lemma was originally presented by Hershberger and Suri in [36] as Lemma 5.1. Their focus was on producing a BSP with small size. For clarity, we modify their proof to suit our purposes.

There are at least 2 edges, so there are that least three orientations. Choose two $(d-2)$-rods B_1 and B_2 be with different orientations that intersect at a face f. Denote $\{v, v_1\}$ and $\{v, v_2\}$ as the complement of the orientations of B_1 and B_2 respectively. Let H be the hyperplane through f; H is orthogonal to the v dimension.

The two $(d-2)$-rods B_1 and B_2 with different orientations that intersect at a face f must exist. Suppose they do not exist, then all faces have rods of the same orientation on both sides. Consider all the faces that are orthogonal to dimension v, all these faces are adjacent to boxes of the same orientation. Thus all rods in the cell are of the same orientation, contradicting the requirement that there are rods of at least two different orientations.

Without loss of generality, let v be dimension 1, v_1 be dimension 2, and v_2 be dimension 3. We can write B_1 as $([a_1, a_2], [b_1, b_2], *, *, \cdots, *)$ and B_2 as $([a_3, a_4], *, [b_3, b_4], *, \cdots, *)$, where $*$ indicates that the rod extends the entire dimension. Without loss of generality, let $a_1 < a_4$. Then, B_1 and B_2 intersect at face f, so $a_2 = a_3$.

Let p be any point on H. First, we show that p cannot be interior to any rod. By contradiction, let p be interior to rod B_3. By Lemma 4.5.5, B_3 must have the same orientation as B_1 or B_2, or B_3's orientation is the complement of $\{v, v'\}$, where $v' \neq v_1$ and $v' \neq v_2$.

Suppose B_3 has the same orientation as B_1; we write $B_3 = ([a_5, a_6], [b_5, b_6], *, *, \cdots, *)$. Since p is interior to B_3, we have $a_5 < a_3 < a_6$. We compare B_3 to B_2:

$$\begin{aligned} B_2 &= ([a_3, a_4], * \quad\quad, [b_3, b_4], *, \cdots, *) \\ B_3 &= ([a_5, a_6], [b_5, b_6], * \quad\quad, *, \cdots, *) \end{aligned}$$

We see that they intersect at $([a_3, \min\{a_4, a_6\}], [b_5, b_6], [b_3, b_4], *, \cdots, *)$.

Similarly, if B_3 has the same orientation as B_2, we write $B_3 = ([a_5, a_6], *, [b_5, b_6], *, \cdots, *)$. We compare B_3 to B_1:

$$\begin{aligned} B_1 &= ([a_1, a_2], [b_1, b_2], * \quad\quad, *, \cdots, *) \\ B_3 &= ([a_5, a_6], * \quad\quad, [b_5, b_6], *, \cdots, *) \end{aligned}$$

We see that they intersect at $([\max\{a_1, a_5\}, a_2], [b_1, b_2], [b_5, b_6], *, \cdots, *)$.

Similarly, if B_3's orientation is the complement of $\{v, v'\}$, we write $B_3 = ([a_5, a_6], *, \cdots, *, [b_5, b_6], *, \cdots, *)$. We compare B_3 to B_1:

$$\begin{aligned} B_1 &= ([a_1, a_2], [b_1, b_2], *, \cdots, *) \\ B_3 &= ([a_5, a_6], * \quad\quad, \cdots, *, [b_5, b_6], *, \cdots, *) \end{aligned}$$

We see that they intersect at $([\max\{a_1, a_5\}, a_2], [b_1, b_2], *, \cdots, *, [b_5, b_6], *, \cdots, *)$.

Thus, no point on H is internal to any rod, so H is a free cut. □

By recursively applying Lemma 4.5.6, we can find a series of free cuts along dimension v that partition the cell into sub-problems, where each sub-problem consists of singly orientated 1-rods. Each sub-problem can be projected down to two dimensions where Seidel's and Adamy's results can be applied.

To construct the MSP, we make a node for the free cuts along dimension v. These free cuts do not increase the number of leaves but increase the height by $O(1)$. For each sub-problem, we make an MSP using Seidel and Adamy's algorithm. In the worst case, we have $W(n) \leq n2^{O\left(\sqrt{\log n}\right)}$ and $H(n) \leq O\left(\sqrt{\log n}\right)$. This completes the proof of Lemma 4.5.1.

4.5.2 The MSP

Our MSP is constructed as follows:

MSP(S)

```
1: apply Lemma 4.4.1 dividing the space into sub-cells; make an MSP for these cuts
2: for each sub-cell Δ do
3:     for each (d − 1)-face f the completely cross Δ do
4:         apply a free cut along f dividing Δ into sub-sub-cells
5:     end for
6:     make an multi-way node for these free cuts
7: end for                          ▷ sub-cells without free cuts are also sub-sub-cells
8: for each sub-sub-cell Δ do
9:     if Δ has no internal vertices then
10:        apply Lemma 4.5.1 to Δ
11:    else
12:        recursively build an MSP tree for Δ
13:    end if
14: end for
```

Analysis:

Let $W(n)$ be the number of leaves in the MSP, $H(n)$ be the height of the MSP, n_i be the number of boxes in the i^{th} sub-problem, m be the number of $(d-3)$-flats intersecting the interior of Δ, and m_i be the number of $(d-3)$-flats intersecting the i^{th} sub-sub-cell.

By a direct application of Lemma 4.4.1, $\max_i m_i = O\left(\frac{n}{r^3}\right)$, but we want $\max_i m_i$ to be dependant on m and independent of n. We duplicate each $(d-3)$-flat $\left\lceil \frac{n}{m} \right\rceil$ times before applying Lemma 4.4.1 and after applying Lemma 4.4.1 we de-duplicate the $(d-3)$-flats that are repeated within a cell. The number of $(d-3)$-flats remaining is bounded by $O(n)$ but $\max_i m_i = O\left(\frac{n}{r^3}/\frac{n}{m}\right) = O\left(\frac{m}{r^3}\right)$.

The recurrence is:

$$W(n,m) \leq \sum_i W(n_i, m_i) + O\left(nr^{O(1)}\right) \quad \text{and} \quad H(n,m) \leq \max_i H(n_i, m_i) + O(1)$$

such that $\max_i n_i = O\left(\frac{n}{r^2}\right)$, $\sum_i n_i = O\left(r^d \frac{n}{r^2}\right) = O\left(nr^{d-2}\right)$, $\max_i m_i = O\left(\frac{m}{r^3}\right)$, and $\sum_i m_i = O\left(r^d \frac{m}{r^3}\right) = O\left(mr^{d-3}\right)$, with the bases cases of $W(1) = 1$, $H(1) = 0$,

$$W(n,0) \leq n2^{O\left(\sqrt{\log n}\right)} \quad \text{and} \quad H(n,0) \leq O\left(\sqrt{\log n}\right).$$

We choose $r = m^\epsilon$. The solution for $H(n,m)$ is simple:

$$\begin{aligned} H(n,m) &\leq \max_i H(n_i, m_i) + O(1) \\ &\leq H\left(n, \frac{m}{r^{3\epsilon}}\right) + O(1) \\ &\leq H\left(n, m^{1-3\epsilon}\right) + O(1) \\ &\leq H(n,0) + O(\log\log m) \\ &\leq O\left(\log\log m + \sqrt{\log n}\right). \end{aligned}$$

The recurrence for the number of leaves is harder to solve. The solution is $W(n,m) \leq c_0 n m^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n}} \log^c m$, for some constants c_0, c_1, and c. We show that it is correct by induction.

$$
\begin{aligned}
W(n,m) &\leq \sum_i W(n_i,m_i) + O\left(nr^{O(1)}\right) \\
&\leq \sum_i c_0 n_i m_i^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n_i}} \log^c m_i + O\left(nr^{O(1)}\right) \\
&\leq c_0 c' \left(r^d \frac{n}{r^2}\right)\left(\frac{m}{r^3}\right)^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n}} \log^c\left(\frac{m}{r^3}\right) + O\left(nr^{O(1)}\right), \quad \text{since } \sum_i n_i = O\left(r^d\frac{n}{r^2}\right) \\
&\qquad\qquad \text{and } \max_i m_i = O\left(\frac{m}{r^3}\right) \\
&\leq c_0 c' n m^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n}} \log^c\left(m^{1-3\epsilon}\right) + O\left(nm^{O(1)\epsilon}\right) \\
&\leq \left(c_0 c' n m^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n}} \log^c m\right)(1-3\epsilon)^c + o\left(nm^{\frac{d-2}{3}}\right) \\
&\leq c_0 n m^{\frac{d-2}{3}} 2^{c_1\sqrt{\log n}} \log^c m, \quad \text{for a sufficiently large } c.
\end{aligned}
$$

Thus, $H(n,n) \leq O\left(\sqrt{\log n}\right)$ and $W(n,n) \leq O\left(n^{\frac{d+1}{3}} 2^{O(\sqrt{\log n})} \log^c n\right)$. By Lemma 4.3.1 we can produce a BSP of height

$$
\begin{aligned}
&\leq \log W(n,n) + O(H(n,n)) \\
&\leq \log O\left(n^{\frac{d+1}{3}} 2^{O(\sqrt{\log n})} \log^c n\right) + O\left(\sqrt{\log n}\right) \\
&\leq \frac{d+1}{3}\log n + O\left(\sqrt{\log n}\right).
\end{aligned}
$$

Theorem 4.5.7 (Point Location Among a Subdivision of Axis-Aligned Disjoint Boxes)**.** *Given a set of n axis-aligned disjoint boxes that form a space-filling subdivision, there is a point location data structure that has a query algorithm that uses $\frac{d+1}{3}\log n + O\left(\sqrt{\log n}\right)$ comparisons.*

4.5.3 Lower Bound

Hershberger, Suri and Tóth showed that the lower bound for the size of any BSP for box subdivisions is $\Omega\left(n^{\frac{4}{3}}\right)$ in three dimensions. Thus, the height of any such BSP is atleast $\frac{4}{3}\log n - \Theta(1)$. Observation 4.2.1 implies that $\frac{4}{3}\log n - \Theta(1)$ comparisons is a lower bound for any point location query algorithm. Thus, our query algorithm is optimal in the leading constant in three dimensions.

Chapter 5

Conclusion and Open Problems

In this thesis, we have studied various classical problems in computational geometry. We presented several new comparison-based algorithms that improve the constant factors on the leading term for the number of comparisons needed. We conclude with some open problems.

- We have presented new randomized algorithms for maxima in three dimensions, line segment intersection, and vertical decomposition of line segments. Are there deterministic algorithms that use the same number of comparisons or less?
- In d dimensions, the maxima can be computed using $dn \log n + O(dn)$ comparisons. For $d \geq 4$, is it possible to use less than $dn \log n$ comparisons? We note that the number of comparisons used may not correlate to the actual running time. Currently, the best algorithm for d-dimensional maxima has a running time of $O(n \log^{d-3} n)$ [9].
- What is the exact worst case complexity for computing the closest pair of points in d-dimensions? We note that the notion of comparisons may need to be extended to include more complicated predicates such as computing the distance between two points. A similar question can be asked where distance is measured using the L_1 or L_∞ metric.
- What is the exact complexity of the counting version of the orthogonal line segment intersection problem?
- We note that the finding a bridge between two convex hulls (see Section 3.3) reduces to linear programming. What is the exact worst case complexity for linear programming in two and three variables?

- What is the exact worst case complexity of the counting, reporting, decision, and dominance versions of orthogonal range query? The trivial grid solution uses $d \log n + O(1)$ comparisons. We note that Dubé examined the one-reporting case (see Section 2.2) [25]; his algorithm is complex, can it be be simplified?
- Is the $O(\sqrt{\log n})$ lower-order term in Seidel and Adamy's query algorithm optimal in the special case where all line segments are orthogonal?
- Can one prove a lower bound for point location among d-dimensional disjoint boxes with a constant factor that converges to infinity as a function of d?
- Can one prove optimality of the lower-order terms for any of our results?
- Can one improve the lower-order terms for any of our results?

References

[1] P. Afshani, J. Barbay, and T. M. Chan. Instance-optimal geometric algorithms. In *Proc. 50th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 129–138. IEEE, 2009.

[2] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, 100(9):643–647, 1979.

[3] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational Geometry*. Springer, 2000.

[4] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput. Geom.*, 16(4):361–368, 1996.

[5] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16(4):369–387, 1996.

[6] T. M. Chan. Three problems about dynamic convex hulls. *Internat. J. Comput. Geom. and Appl.*, 22(04):341–364, 2012.

[7] T. M. Chan. Klee's measure problem made easy. In *Proc. 54th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 410–419. IEEE, 2013.

[8] T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.

[9] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 1–10. ACM, 2011.

[10] T. M. Chan, J. Snoeyink, and C. K. Yap. Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams. *Discrete Comput. Geom.*, 18(4):433–454, 1997.

[11] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(1):145–158, 1993.

[12] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10(1):377–409, 1993.

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

[14] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994.

[15] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Comput. Geom. Theory Appl.*, 5(1):27–32, 1995.

[16] Y.-J. Chiang and R. Tamassia. Dynamization of the trapezoid method for planar point location in monotone subdivisions. *Internat. J. Comput. Geom. and Appl.*, 2(3):311–333, 1992.

[17] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2(1):195–222, 1987.

[18] K. L. Clarkson. More output-sensitive geometric algorithms. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 695–702. IEEE, 1994.

[19] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4(1):387–421, 1989.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press Cambridge, 2001.

[21] W. Cunto and J. I. Munro. Average case selection. *J. ACM*, 36(2):270–279, 1989.

[22] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241–253, 1983.

[23] D. Dor and U. Zwick. Selecting the median. *SIAM J. Comput.*, 28(5):1722–1758, 1999.

[24] D. Dor and U. Zwick. Median selection requires $(2 + \epsilon)n$ comparisons. *SIAM J. Discrete Math.*, 14(3):312–325, 2001.

[25] T. Dubé. Dominance range-query: The one-reporting case. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 208–217, 1993.

[26] A. Dumitrescu, J. S. B. Mitchell, and M. Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. *Discrete Comput. Geom.*, 31(2):207–227, 2004.

[27] M. Dyer. Linear time algorithms for two-and three-variable linear programs. *SIAM J. Comput*, 13(1):31–45, 1984.

[28] H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Inform. Process. Lett.*, 13(4):177–181, 1981.

[29] J. Ford, R. Lester, and S. M. Johnson. A tournament problem. *Amer. Math. Monthly*, pages 387–389, 1959.

[30] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted rows and columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982.

[31] H. Gabow, J. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annu. ACM Sympos. Theory Comput.*, pages 135–143. ACM, 1984.

[32] M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11(6):501–524, 1994.

[33] G. H. Gonnet and J. I. Munro. Heaps on heaps. *SIAM J. Comput.*, 15(4):964–971, 1986.

[34] M. T. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. *SIAM J. Comput.*, 28(2):612–636, 1998.

[35] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1(4):132–133, 1972.

[36] J. Hershberger and S. Suri. Binary space partitions for 3d subdivisions. In *Proc. 14th ACM-SIAM Sympos. on Discrete Algorithms*, pages 100–108. SIAM, 2003.

[37] J. Hershberger, S. Suri, and C. D. Tóth. Binary space partitions of orthogonal subdivisions. *SIAM J. Comput.*, 34(6):1380–1397, 2005.

[38] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, 2(1):18–21, 1973.

[39] K. Kaligosi, K. Mehlhorn, J. I. Munro, and P. Sanders. Towards optimal multiple selection. In *Proc. 32nd Int. Colloq. Automata, Languages and Programming*, pages 103–114. Springer, 2005.

[40] D. G. Kirkpatrick and R. Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 89–96. ACM, 1985.

[41] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm. *SIAM J. Comput.*, 15(1):287–299, 1986.

[42] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[43] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[44] Z. Li and B. A. Reed. Heap building bounds. *Algorithms and Data Structures*, pages 14–23, 2005.

[45] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[46] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(1):157–182, 1993.

[47] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Comput*, 12(4):759–776, 1983.

[48] K. Mehlhorn. *Data Structures and Algorithm 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1981.

[49] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981.

[50] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.

[51] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5(1):485–503, 1990.

[52] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13(1):99–113, 1992.

[53] I. Pohl. A sorting problem and its complexity. *Commun. ACM*, 15(6):462–464, 1972.

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

[55] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[56] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.

[57] R. Seidel and U. Adamy. On the exact worst case complexity of planar point location. *J. Algorithms*, 37(1):189–217, 2000.

[58] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4):464–497, 1996.

[59] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 17th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 208–215. IEEE, 1976.

[60] P. K. Stockmeyer and F. F. Yao. On the optimality of linear merge. *SIAM J. Discrete Math.*, 9(1):85–90, 1980.

[61] C. D. Tóth. A note on binary plane partitions. *Discrete Comput. Geom.*, pages 3–16, 2003.

[62] C. D. Tóth. Binary plane partitions for disjoint line segments. *Discrete Comput. Geom.*, 45(4):617–646, 2011.