# Geometric Algorithms for Object Placement and Planarity in a Terrain

**Rahul Ray**

*Max-Planck Institut für Informatik*

# Geometric Algorithms for Object Placement and Planarity in a Terrain

# Geometric Algorithms for Object Placement and Planarity in a Terrain

**Rahul Ray**

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Saarbrücken
July 27th, 2004

***Max-Planck Institut für Informatik***

| | |
|---|---|
| Tag des Kolloquiums: | 27 July 2004 |
| Dekan: | Prof. Dr. Philipp Slusallek |
| Gutachter: | Prof. Dr. Kurt Mehlhorn |
| | Prof. Dr. Stefan Schirra |

Dedicated to my parents

# Acknowledgements

Many people have taught, inspired, encouraged, supported, helped and advised me during the time in which I worked on this thesis. I wish to express my deepest gratitude to all of them, without which this thesis would never happen.

First of all, I would like to thank my advisor, **Prof. Dr. Kurt Mehlhorn**, for providing me a perfect balance of scientific guidance and scientific freedom. He has always been a great source of motivation whenever I needed. I would like to thank him for creating a wonderful research atmosphere in Max-Planck Institut für Informatik, Saarbrücken. I also wish to thank Dr. Stefan Funke and Dr. Theocharis Malamatos for co-guiding my research work that led to this thesis. They are great persons to work with and I consider myself privileged to have collaborated with them in my research. I share some of the most enjoyable moments in MPI with them which came along during many of our enlighting discussions or exchanges of innumerable emails. They also helped me in proof-reading this thesis manuscript and improving the presentation a lot.

When I go back to my uninteresting days in industry, I shall never forget Prof. Michiel Smid who gave me an opportunity to think about a PhD option. I am grateful to him for infinite number of reasons. He introduced me to the planarity problem in terrain and many other interesting aspects of it while I was in Magdeburg University (Germany). I am thankful to ever smiling Prof. Ulrich Wendt for his wonderful collaboration from material science department in our project. I learned a lot from my co-authors, whose ideas and insightful suggestions were invaluable in discovering many of the algorithms presented in this thesis. My sincere thanks to them.

I thank Prof. Stefan Schirra for kindly agreeing to review the thesis and to act as an examiner.

I would like to thank and express my sincere gratitude to Suman Mukherjee, who pursuaded me to continue this research at a time I thought to abandon it. My friend Arnab Paul has always kept on inspiring me even before I started my research. All my friends in *Tajmahal, Chhannachhara* have been a constant source of inspiration for me to finish this thesis one day. My special thanks to them.

My MPII experience would not be the same without my friends, colleagues and office-mates (categories not mutually exclusive). I thank Naveen Sivadasan, Debapriyo, Hisao, Venkatesh, Tobias, Ingmar, Christian, Holgar, Kerstin, Petra, Arno, Nicola, Kavitha, Sunil and many others who are/were my colleagues in various times in MPII for their friendly help and making my life fun and enjoyable.

I express my gratitude and thanks to Deb, Sudip, Chandrima, Pintu, Raja, Sai, Reddy, Venkat, Fawad, Tarang, Hareesh and many others in Saarcricket Club for providing wonderful social interaction which has tunred my stay in Saarbrücken an unforgettable event in my life.

Finally, I would like to thank those whom I owe the most. My parents have always encouraged me to pursue higher studies even when I was in a kindergarten !! They have greatly influenced my life with love and guidance and most importantly freedom of my soul which allowed me to reach where I am now. My then class mate and now wife, Piyali, was a great emotional support for me, whether it was a time for hard work or for taking it easy. To mark the submission of this thesis, we were blessed with a baby son, Rishav, on 22nd April.

I gratefully acknowledge the fellowship from *International Max-Planck Research School (IMPRS)* , that supported my research at Max-Planck Institut für Informatik, Saarbrücken.

# Abstract

We consider the following *placement problem*: Let $C$ be a compact set in $\mathbb{R}^2$ and let $S$ be a set of $n$ points in $\mathbb{R}^2$. The objective is to compute a translate of $C$ that contains the maximum number, $\kappa^*$, of points of $S$. Motivated by the applications in clustering and pattern recognition, this optimal placement problem has received much attention over the last two decades. It is known that this problem can be solved in a time that is roughly quadratic in $n$. We show for a given $\varepsilon > 0$ how random-sampling and bucketing techniques can be used to develop a near-linear-time Monte Carlo algorithm that computes a placement of $C$ containing at least $(1 - \varepsilon)\kappa^*$ points of $S$ with high probability.

When $C$ is a unit disk, we give an approximation algorithm for the optimal placement problem by approximating the constraining radius of the disk. Here, our algorithm computes a placement of the disk of radius $(1 + \varepsilon)$ which contains at least $\kappa^*$ points, where $\kappa^*$ denotes the maximum number of points covered by any unit disk. The running time of this algorithm is $O(n/\varepsilon^2)$.

Then, we turn to the problem of computing the largest connected region in a triangulated terrain of size $n$ for which the normals of the triangles deviate by at most some small fixed angle. We devise an exact algorithm by adapting dynamic graph connectivity algorithm to compute the largest planar region in $O(n^2 \log n(\log \log n)^3)$ time. We also give a heuristic that can be used to compute sufficiently large planar regions in a terrain much faster. We discuss an implementation of this heuristic, and show some experimental results for terrains representing three-dimensional (topographical) images of fracture surfaces of metals obtained by confocal laser scanning microscopy, which directly motivated our research into this direction.

Since the output of this heuristic comes with no quality assurance, we present a new approximation scheme for the same problem which apart from giving a guarantee on the quality of the produced solution also has been implemented in practice and shows good performance on real-world data representing fracture surfaces. This approximate deterministic algorithm computes in $O(n/\varepsilon^2)$ time the largest approximately connected planar region in the terrain. We also give a variant of the above algorithm with a better dependence on $\varepsilon$ at the cost of an extra poly-logarithmic factor on $n$. For a sufficiently large $n$, both the algorithms consume optimal $O(n)$ space.

# Kurzzusammenfassung

Wir betrachten das folgende *Platzierungsproblem*: Sei $C$ eine kompakte Menge im $\mathbb{R}^2$ and $S$ eine Menge von $n$ Punkten im $\mathbb{R}^2$. Das Ziel ist es, ein Verschiebung von $C$ zu berechnen, welche eine maximale Anzahl $\kappa^*$ an Punkten aus $S$ enthält. Aufgrund zahlreichen Anwendungen im Clustering und in der Mustererkennung wurde dieses Problem in den letzten Jahrzehnten oft betrachtet. Es gibt bekannte Algorithmen, die dieses Problem in fast-quadratischer Zeit lösen. Wir präsentieren einen Monte-Carlo Algorithmus, der für ein gegebenes $\varepsilon > 0$ in fast-linearer Zeit eine Platzierung von $C$ berechnet, welche mindestens $(1 - \varepsilon)\kappa^*$ Punkte aus $S$ mit hoher Wahrscheinlichkeit enthält. Unser Algorithmus basiert auf der Entnahme zufälliger Stichproben und Bucketingtechniken.

Im Falle, dass $C$ eine Einheitskreisscheibe ist, stellen wir einen Approximationsalgorithmus für das Plazierungsproblem vor, wobei wir den Radius der Kreisscheibe relaxieren. So erhalten wir eine Platzierung einer Kreisscheibe mit Radius $(1 + \varepsilon)$ welche mindestens $\kappa^*$ Punkte enthält. Hierbei bezeichnet $\kappa^*$ die maximale Anzahl von Punkten, die von einer Einheitskreisscheibe überdeckt werden können. Die Laufzeit dieses Algorithmus ist $O(n/\varepsilon^2)$.

Danach betrachten wir das Problem, die grösste zusammenhängende Region in einem triangulierten Terrain der Grösse $n$ zu berechnen, für welche die Dreiecksnormalen der Region um nicht mehr als einen vorgegebenen Winkel abweichen. Wir präsentieren einen exakten Algorithmus, der auf einer Datenstruktur zur dynamischen Verwaltung von Zusammenhangskomponenten eines Graphen basiert und das Problem in $O(n^2 \log n(\log \log n)^3)$ Zeit löst. Wir stellen auch eine heuristische Lösung vor, die schnell relativ grosse planare Regionen berechnet. Diese Heuristik wurde implementiert und experimentell evaluiert. Dazu lagen uns dreidimensionale topografische Daten von Bruchoberflächen verschiedenster Metallsorten vor. Diese Anwendung war auch die ursprüngliche Motivation für unsere Forschung.

Da diese Heuristik jedoch keinerlei Qualitätsgarantie für die berechnete Region liefert, stellen wir schliesslich ein neues Approximationsschema vor, welches sowohl eine garantierte Güte liefert, als auch praktisch implementiert wurde und auf unseren Testdaten gute Resultate erzielte. Die Laufzeit dieses Approximationsalgorithmus ist $O(n/\varepsilon^2)$.

# Contents

## Chapter 1

# Introduction

In this thesis we present results in two areas which are geometric in nature.

The importance and applications of computational geometry in various fields of science and engineering are well studied and a growing area of research. Imagine installing a geo-stationary satellite which can cover a finite amount of area down on earth's surface. Now an interesting question is how exactly one would like to place the satellite so that it covers most of the static land-lines. And this turns out to be one of the fundamental problems in geometry to compute a placement of a planar object to cover maximum number of points. We have considered this placement problem and presented several approaches to solve it efficiently in a randomized, deterministic and approximate deterministic settings.

The second problem we have investigated in our thesis was given to us by the material scientists from Magdeburg University. They wanted to quantify the fracture surface topographies given as 3-dimensional images obtained from confocal laser scanning microscope. The objective was to detect large planar regions in the fractured surface, which can give an insight of the cause of fracture. Since the strict definition of plane ( all the normals of a plane are parallel to one another ) does not fit well into the micro-structure of the fracture surfaces, we have introduced the notion of 'approximate' planarity where we allow a small deviation of the normals of an 'approximately' planar region.

We elaborate two main results on locating nearly planar regions in a terrain. The first result focuses mainly on an exact algorithm and a fast grid-based heuristic. The second contribution solves an approximated version of the same problem with a guarantee on the computed output.

The thesis is organized as follows. In Chapter 2, we discuss the placement problem and

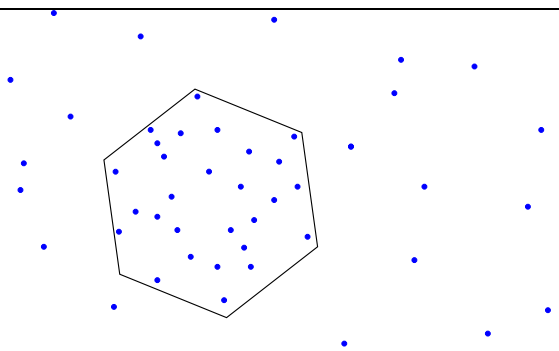

*Fig. 1.1:* Placing a hexagonal object to cover maximum points.

several algorithms to solve it in details. Then we turn to the problem of locating planar regions in a terrain. In Chapter 3, we give an exact algorithm and an efficient heuristic of the problem. The heuristic approach performs very well in the real-world data. In Chapter 4, we present an approximate version of the same problem and show that our algorithms here can compute the largest planar region in terrain with a guarantee.

Now we give an overview of the two problems we have studied in the thesis and the main results we have achieved.

## 1.1 Placement Problem

### 1.1.1 Problem Definition

Let $C$ be a compact set in $\mathbb{R}^2$ and let $S$ be a set of $n$ points in $\mathbb{R}^2$. We define the *optimal-placement* problem to be that of computing a point $t \in \mathbb{R}^2$ for which the translate $C + t$ of $C$ contains the maximum number of points of $S$. Set

$$\kappa^*(C, S) = \max_{t \in \mathbb{R}^2} |S \cap (C + t)|.$$

Since this optimal-placement problem is believed to be of $o(n^2)$ time hard, we have considered the following two approximate versions of the same.

**Approximation on $k^*$:** We call a translate $C + t$ of $C$ an *$\epsilon$-approximate placement* if $|S \cap (C + t)| \geq (1 - \epsilon)\kappa^*(C, S)$, and an algorithm that produces an $\epsilon$-approximate placement is called an *$\epsilon$-approximation* algorithm. We define the *$\epsilon$-optimal-placement* problem to be the one that asks for an $\epsilon$-approximate placement.

**Approximation on the radius of $C$:** Assume $C^1$ is a unit disk. We determine a placement $(x, y)$ of a disc $C^{1+\epsilon}$ of radius $1 + \epsilon$ with $|C^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*(C, S)$.

### 1.1.2 Motivation and State of the Art

A fundamental problem in computational geometry is shape fitting (see [HU87, LSW88]), which has its applications in pattern recognition, computer vision, machine learning and many other areas dealing with the best shape which "fits" a given point set. This problem has received much attention over the last two decades for different application specific geometric shapes of $C$. In pattern recognition, e.g., we are given a set (e.g., a polygon) and want to match it to a given set of objects (see Huttenlocher and Ullman [HU87] and Lamdan *et al.* [LSW88]). The problem we have studied here can be regarded as that of matching a set $C$ to a set $S$ of points. We believe this kind of placement problems investigated here in this thesis would find its place in the standard textbooks of computational geometry in near future.

Chazelle and Lee [CL86] presented an $O(n^2)$-time algorithm for the case in which $C$ is a circular disk, and Eppstein and Erickson [EE94] proposed an $O(n \log n)$-time algorithm for rectangles. Efrat et al. [ESZ94] developed an algorithm for convex $m$-gons with a running time of $O(n\kappa^* \log n \log m + m)$, where $\kappa^* = \kappa^*(C, S)$, which was subsequently improved by Barequet et al. [BDP97] to $O(n \log n + n\kappa^* \log(m\kappa^*) + m)$.

| Time | Error probability | Schemes |
|---|---|---|
| $n + \gamma^2 n^2$ | $ne^{-\gamma\kappa^*}$ | Random sampling |
| $n \log n$ | $e^{-\sqrt{\kappa^* \log n}}$ | Random sampling with bucketing (Monte-Carlo Alg) |
| $n$ | $ne^{-\sqrt{\kappa^*}}$ | Two levels of random sampling with bucketing |
| $n^{1+\delta}$ | 0 | Approximating $\kappa^*$ |
| $n/\epsilon^2$ | 0 | Approximating radius of $C$ |

*Tab. 1.1:* Worst-case running times (constant factors omitted) and error probabilities of our $\varepsilon$-approximation algorithms if $C$ is a circular disk; $\delta$ and $\varepsilon$ are arbitrarily small positive constants and $\gamma$ is a nonnegative real number, possibly depending on $n$.

### 1.1.3 Main Results

We present two algorithms for the optimal-placement problem and show how bucketing can be used to expedite the running time, especially if $C$ is fat. In particular, let $T(n)$ be the running time of an algorithm for the optimal-placement problem on a set of $n$ points, and let $T_{\mathrm{g}}(n)$ denote the time required to partition $n$ points into the cells of an integer grid. Then the bucketing algorithm can compute an optimal placement in time $O(T_{\mathrm{g}}(n) + nT(\kappa^*)/\kappa^*)$, where $\kappa^* = \kappa^*(C, S)$. Besides being interesting in its own right, this will be crucial for the approximation algorithms.

Subsequently, we show that using random sampling and/or bucketing, we can transform any deterministic algorithm for the optimal-placement problem to a Monte-Carlo algorithm for the $\varepsilon$-optimal-placement problem. Given a parameter $\gamma \geq 0$, the first algorithm in Section 2.5, based on a random-sampling technique, computes an $\varepsilon$-approximate placement in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2\gamma\kappa^*}$, where $s$ is the maximum number of distinct intersections of the boundaries of the two translates $C + t$ and $C + t'$ of $C$ for $t, t' \in S$. The second algorithm combines the random-sampling technique with the bucketing technique and computes an $\varepsilon$-approximate placement in $O(T_{\mathrm{g}}(n)+nQ(m)+nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ time with error probability at most $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$. $Q(m)$ is the time to decide whether $p \in C$. For circular disks and constant $\varepsilon$, the running time becomes $O(n \log n)$ and the error probability is at most $e^{-\sqrt{\kappa^* \log n}}$; see Table 1.1. If $C$ is fat and $m = O(1)$, by combining two levels of random sampling with the bucketing technique, we can compute an $\varepsilon$-approximate placement in $O(n)$ time for constant $\varepsilon$ with error probability at most $ne^{-\sqrt{\kappa^*}}$.

We suggest a deterministic algorithm, based on the cutting algorithm of Chazelle [Cha93], that computes an $\varepsilon$-approximate placement for sets $C$ that satisfy certain conditions. If $C$ has $O(1)$ edges, the algorithm runs in $O(n^{1+\delta} + n/\varepsilon)$ time, for any given constant $\delta > 0$ (where the constant of proportionality depends on $\delta$). If $C$ is a convex $m$-gon, the running time becomes $O((n^{1+\delta} + n/\varepsilon) \log m)$.

Finally, we present an algorithm for placing a unit disk in the plane such that the number

of points contained is maximized. Our algorithm yields a placement of a disk of radius $(1+\epsilon)$ which contains at least $\kappa^*$ points, where $\kappa^*$ denotes the maximum number of points in any unit disk. The running time of this algorithm is $O(n/\epsilon^2)$. We also sketch a more complicated variant running in $O(n + (n/\kappa^*) \cdot (1/\epsilon^3))$ time which is better for $\kappa^* = \omega(1/\epsilon)$.

Observe that for these last algorithms our notion of approximation differs from the one used so far in the previous settings which is the same as in [AHR$^+$02] (they approximate the size of the resulting set) and rather resembles the notion used in [HPM03] where one approximates the constraining radius. Not only are the running times of our algorithms linear in $n$ and the dependencies on $\epsilon$ reasonable, but also the constants involved are small enough to make them relevant in practice. In the Table 1.1 we summarize our results related to the placement problem. All these results have been published in *10th Annual European Symposium of Algorithms* in 2002 and partly in *20th ACM Symposium on Computational Geometry* in 2004.

## 1.2 Planarity in a Terrain

### 1.2.1 Problem Definition

A terrain is a surface in $\mathbb{R}^3$ defined by a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. If $f$ is piecewise linear and the surface consists of a collection of triangles, the terrain is called a *triangulated irregular network (TIN)*. Given a triangulated irregular network $\mathcal{T}$, our goal is to find large, almost planar regions in $\mathcal{T}$. More formally, we want to find a subset of triangles $T$ of $\mathcal{T}$ and a vector $\overrightarrow{r}$ (called the *reference normal*), such that

1. the adjacency graph of the triangles in $T$ is connected,
2. for each triangle $t \in T$, the angle between $\overrightarrow{r}$ and $\overrightarrow{n_t}$ is at most $\delta$, where $\overrightarrow{n_t}$ denotes the normal of triangle $t$ and $\delta$ is a given parameter, and
3. $T$ is chosen such that the total weight of $T$ is maximized, where the weight can be for example the number or the total area of the triangles in $T$ (depending on the application).

**Remark:** The set of triangles $T$ satisfying (1)-(3) above is called the $\delta$-planar set.

### 1.2.2 Motivation

The problem considered here arose in a collaboration between the Departments of Computer Science and Materials Science at the University of Magdeburg. The goal of this project is to design and implement algorithms that can be used for the quantification of fracture surface topographies, given as three-dimensional images obtained by confocal laser scanning microscopy (CLSM).

According to material scientists ( [WLS$^+$02]), surface topographies contain useful information about their generation process and about the influence of crystal structure, microstructure and external loading conditions on these processes. Furthermore, the topography can

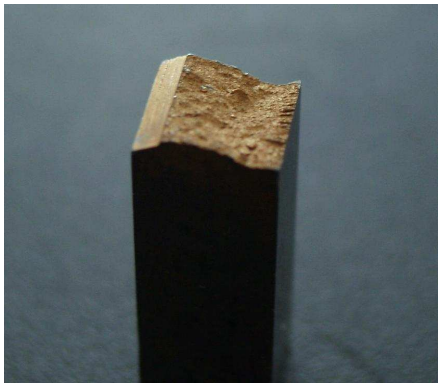

*Fig. 1.2:* A typical fracture surface from Carbon Steel coated with a thin Silver(Au) layer to get a better reflectability when inspected under CLSM

substantially influence the functionality of structural components. Surfaces generated by fracture, wear, corrosion and machining are examined by confocal laser scanning microscopy. The quantification of material topographies can support establishing and understanding of correlations between the processing parameters, microstructure , testing conditions and material properties and gives hints to the interpretation of the mechanical behavior. In practice, quantification of fracture surfaces can give a guideline for the improvement and to the modification of known materials, as well as to new design criteria of materials. This holds for all kind of materials such as metals, polymers, ceramics, and composites. Two types of parameters are generally used in the quantification of topography: global parameters and feature-related (object-specific) parameters. The latter describe the discrete geometrical objects, for example, planar regions like facets in brittle fracture surfaces. A typical fracture surface is shown in Figure 1.2.

The images of the fracture surfaces are given as $512 \times 512$ arrays of pixels, where the grey value stored at each entry is equal to the height of the corresponding terrain point. One goal in our research is to find large connected regions in this image that are approximately planar. The normal vectors and areas of these planar regions give useful information about the fracture surface generating process.

Almost a similar kind of problem in macro-scale has been investigated closely by the community in *Geographic Information Systems* as they are inherently based on retrieving information from the rather simple surface data, for example a TIN or a rasterized image - DEMs *(Digital Elevation Models)*. DEMs can be obtained by scanning any surface and then storing the height values of junctions in a uniform 2D-grid of any desired resolution.

### 1.2.3 Main Results

We show how computational geometry and dynamic graph algorithms can be used to solve the problem described above in $O(n^2 \log n (\log \log n)^3)$ time.

A closely related problem (computing the deepest point in an arrangement of hyperplanes) is shown in [GO95] to be 3SUM hard, implying that it is unlikely that our problem can be solved in sub-quadratic time. Therefore, we describe a simple grid-based heuristic. We have

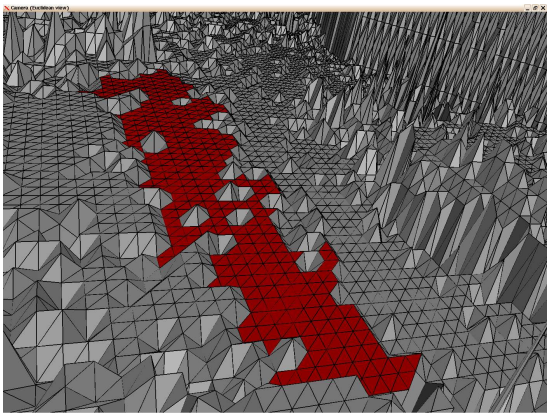

*Fig. 1.3:* An approximately planar region shown in red in the terrain of a fracture surface

implemented this heuristic and discuss some details about it. We also present some experimental results on images of fracture surfaces obtained by confocal laser scanning microscopy. These results show that the heuristic is able to find $\delta$-planar regions whose area is sufficiently large. This work has been published in *Materialwissenschaft und Werkstofftechnik* in 2002 ( [WLS$^+$02]) and *Discrete Applied Mathematics* in April 2004 ( [SRWL04]).

## 1.3 Finding Large Planar Regions in a Terrain with a Guarantee

### 1.3.1 Problem Definition

We look at the same problem as described in the previous section. But here we solve an approximate version of the problem. There are at least two ways to define the notion of an $\epsilon$-approximate $\delta$-planar set $T$. One way would be to require $T$ to be $\delta$-planar and of weight at least $(1-\epsilon)$ times the weight of an optimal $\delta$-planar set. Unfortunately solving this type of approximation seems to be as difficult as solving the problem exactly. We adopt the following notion of approximation: A subset of triangles $T$ of $\mathcal{T}$ is *$\epsilon$-approximate* $\delta$-planar if it is $\delta(1+\epsilon)$-planar and has weight at least as large as an optimal $\delta$-planar set.

### 1.3.2 Main Results

We present an algorithm which, given some parameters $\delta$ and $\epsilon$ produces a connected subterrain and a reference normal such that all triangle normals in the subterrain deviate at most $(1+\epsilon)\cdot\delta$ from the reference normal, and the weight of the subterrain is at least the weight of the optimal subterrain with maximum deviation $\delta$. The running time of this algorithm is $O(n/\epsilon^2)$. We sketch also a variant of this algorithm with a better dependence on $\epsilon$ but an extra poly-logarithmic factor on $n$. For $n$ sufficiently large, both algorithms use optimal $O(n)$ space.

The experimental and perhaps main contribution here is our implementation of the planarity detector which runs in reasonable time on real-world test data consisting of terrains with several hundred thousands triangles. This result has been published in *20th ACM Sympo-*

*sium on Computational Geometry (SoCG)* in 2004 and a journal version of the same has been invited for *International Journal on Computational Geometry and Applications* ( [FMR04]).

## Chapter 2

# Placement Algorithms

## 2.1 Introduction

Let $C$ be a compact set in $\mathbb{R}^2$ and let $S$ be a set of $n$ points in $\mathbb{R}^2$. We define the *optimal-placement* problem to be that of computing a point $t \in \mathbb{R}^2$ for which the translate $C + t$ of $C$ contains the maximum number of points of $S$. Set

$$\kappa^*(C, S) = \max_{t \in \mathbb{R}^2} |S \cap (C + t)|.$$

Motivated by applications in clustering and pattern recognition (see [HU87, LSW88]), the optimal-placement problem has received much attention over the last two decades. Chazelle and Lee [CL86] presented an $O(n^2)$-time algorithm for the case in which $C$ is a circular disk, and Eppstein and Erickson [EE94] proposed an $O(n \log n)$-time algorithm for rectangles. Efrat et al. [ESZ94] developed an algorithm for convex $m$-gons with a running time of $O(n\kappa^* \log n \log m + m)$, where $\kappa^* = \kappa^*(C, S)$, which was subsequently improved by Barequet et al. [BDP97] to $O(n \log n + n\kappa^* \log(m\kappa^*) + m)$.

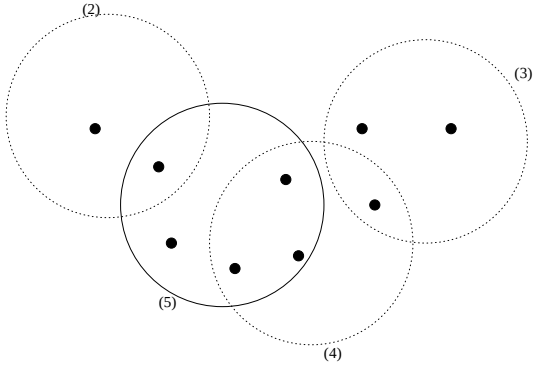


*Fig. 2.1:* Four placements of a disc and their respective containment of a point set

All the algorithms above, except the one for rectangles, require at least quadratic time in the worst case, which raises the question of whether a near-linear approximation algorithm exists for the optimal-placement problem. In this chapter we answer the question in the affirmative by solving the following two approximate versions of the original placement problem.

**Approximation on $k^*$:** We call a translate $C + t$ of $C$ an *$\epsilon$-approximate placement* if $|S \cap (C + t)| \geq (1 - \epsilon)\kappa^*(C, S)$, and an algorithm that produces an $\epsilon$-approximate placement

is called an $\epsilon$-*approximation* algorithm. We define the $\epsilon$-*optimal-placement* problem to be the one that asks for an $\epsilon$-approximate placement. Throughout this chapter we will refer to this approximation scheme except in Section 2.7, where the following approximation is used.

**Approximation on the radius of $C$:** Assume $C$ is a unit disk. We determine a placement $(x, y)$ of a disc $C^{1+\epsilon}$ of radius $1 + \epsilon$ with $|C^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*(C, S)$, where $\kappa^*$ denotes the maximum number of points covered by any unit disk.

## 2.2 Model of Computation

We make the following assumptions about $C$ and $S$:

(A1) The boundary of $C$, denoted by $\partial C$, is connected and consists of $m$ edges, each of which is described by a polynomial of bounded degree. The endpoints of these edges are called the vertices of $C$. We will refer to $C$ as a *disk*.

(A2) For all distinct $t, t' \in S$, the boundaries of the two translates $C+t$ and $C+t'$ of $C$ intersect in at most $s$ points, and the intersections can be computed in $I(m)$ time. By computing an intersection, we here mean determining the two edges, one of each translate, that are involved in the intersection. Moreover, for every point $p \in \mathbb{R}^2$, we can decide in $Q(m)$ time whether $p \in C$.

(A3) $C$ is sandwiched between two axes-parallel rectangles whose widths and heights differ by factors of at most $\alpha$ and $\beta$, respectively, for $\alpha, \beta \geq 1$. We call $C$ *fat* if $\alpha$ and $\beta$ are constants. This condition is detailed below in Definition 1.

(A4) The roots of a bounded-degree polynomial can be computed in $O(1)$ time. This implies that primitive operations on the edges of $C$, e.g., computing the intersection points between two edges of two translates $C + t$ and $C + t'$, with $t, t' \in S$, can be performed in $O(1)$ time.

(A5) For any edge $e$ of $C$ with endpoints $a$ and $b$, and for points $x$ and $y$ on $e$, we can decide in $O(1)$ time if $x$ or $y$ is seen first while walking along $e$ from $a$ to $b$.

Our notion of fatness appears already implicitly in Barequet *et al.* [BDP97], who show that, after an appropriate rotation, every convex polygon is $(2, 4)$-fat, and in Schwarzkopf *et al.* [SFRW98], who show that convex polygons are even $(2, 2)$-fat.

We consider sets $C$, which we refer to as disk, that satisfy the following fatness condition.

**Definition 1.** Let $\alpha$ and $\beta$ be positive integers. We say that $C$ is $(\alpha, \beta)$-*fat* if there are two axes-parallel rectangles $R_0$ and $R_1$ such that

1. $R_0$ is contained in $C$, and $C$ is contained in $R_1$,
2. the length of the horizontal side of $R_1$ is at most $\alpha$ times the length of the horizontal side of $R_0$, and
3. the length of the vertical side of $R_1$ is at most $\beta$ times the length of the vertical side of $R_0$.

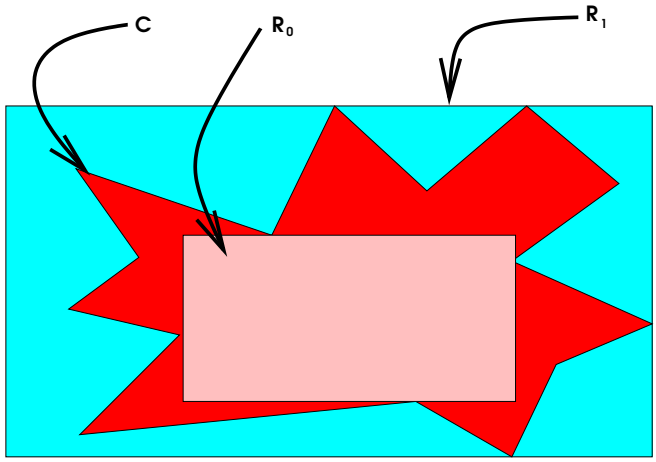


*Fig. 2.2:* $C$ is a fat object

Throughout this chapter, we make the following *general-position* assumption about the point set $S$. We assume that for any two distinct points $p$ and $q$ of $S$, the boundaries of the translates $C + p$ and $C + q$ are disjoint or intersect in a finite number of points. That is, no two edges of $C + p$ and $C + q$ overlap. Also, we assume that for any three distinct points $p$, $q$, and $r$ of $S$, the boundaries of the three translates $C + p$, $C + q$, and $C + r$ do not intersect in a single point. We make these assumptions just to simplify the description. Our algorithms can easily be extended to handle arbitrary sets of points.

## 2.3 Overview of Results

In Section 2.4, we present two algorithms for the optimal-placement problem and show how bucketing can be used to expedite the running time, especially if $C$ is fat. In particular, let $T(n)$ be the running time of an algorithm for the optimal-placement problem on a set of $n$ points, and let $T_{\mathrm{g}}(n)$ denote the time required to partition $n$ points into the cells of an integer grid. Then the bucketing algorithm can compute an optimal placement in time $O(T_{\mathrm{g}}(n) + nT(\kappa^*)/\kappa^*)$, where $\kappa^* = \kappa^*(C, S)$. Besides being interesting in its own right, this will be crucial for the approximation algorithms.

In Section 2.5, we show that using random sampling and/or bucketing, we can transform any deterministic algorithm for the optimal-placement problem to a Monte-Carlo algorithm for the $\varepsilon$-optimal-placement problem. Given a parameter $\gamma \geq 0$, the first algorithm in Section 2.5, based on a random-sampling technique, computes an $\varepsilon$-approximate placement in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2\gamma\kappa^*}$. The second algorithm combines the random-sampling technique with the bucketing technique and computes an $\varepsilon$-approximate placement in $O(T_{\mathrm{g}}(n) + nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ time with error probability at most $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$. For circular disks and constant $\varepsilon$, the running time becomes $O(n \log n)$ and the error probability is at most $e^{-\sqrt{\kappa^* \log n}}$; see Table 2.1. If $C$ is fat and $m = O(1)$, by combining two levels of random sampling with the bucketing technique, we can compute an $\varepsilon$-approximate placement in $O(n)$ time for constant $\varepsilon$ with error probability at most $ne^{-\sqrt{\kappa^*}}$.

| Time | Error probability | Reference |
|---|---|---|
| $n + (\gamma n)^2$ | $ne^{-\gamma\kappa^*}$ | Section 2.5.1 |
| $n \log n$ | $e^{-\sqrt{\kappa^* \log n}}$ | Section 2.5.2 |
| $n$ | $ne^{-\sqrt{\kappa^*}}$ | Section 2.5.2 |
| $n^{1+\delta}$ | 0 | Section 2.6 |
| $n/\epsilon^2$ | 0 | Section 2.7 |

*Tab. 2.1:* Worst-case running times (constant factors omitted) and error probabilities of our $\varepsilon$-approximation algorithms if $C$ is a circular disk; $\delta$ and $\varepsilon$ are arbitrarily small positive constants and $\gamma$ is a nonnegative real number, possibly depending on $n$.

In Section 2.6, we also present a deterministic algorithm, based on the cutting algorithm of Chazelle [Cha93], that computes an $\varepsilon$-approximate placement for sets $C$ that satisfy (A1)–(A5). If $C$ has $O(1)$ edges, the algorithm runs in $O(n^{1+\delta} + n/\varepsilon)$ time, for any given constant $\delta > 0$ (where the constant of proportionality depends on $\delta$). If $C$ is a convex $m$-gon, the running time is $O((n^{1+\delta} + n/\varepsilon) \log m)$.

Finally, in Section 2.7, we briefly describe an algorithm for placing a unit disk in the plane such that the number of points contained is maximized. Our algorithm yields a placement of a disk of radius $(1 + \epsilon)$ which contains at least $\kappa^*$ points, where $\kappa^*$ denotes the maximum number of points in any unit disk. The running time of this algorithm is $O(n/\epsilon^2)$. We also sketch a more complicated variant running in $O(n + (n/\kappa^*) \cdot (1/\epsilon^3))$ time which is better for $\kappa^* = \omega(1/\epsilon)$.

Observe that for these last algorithms in Section 2.7 our notion of approximation differs from the one used so far in this chapter which is the same as in our work [AHR$^+$02] (we approximated the size of the resulting set) and rather resembles the notion used in [HPM03] where one approximates the constraining radius/angle. Not only are the running times of our algorithms linear in $n$ and the dependencies on $\epsilon$ reasonable, but also the constants involved are small enough to make them relevant in practice.

## 2.4 Preliminaries and Exact Algorithms

Let $C$ be a disk satisfying assumptions (A1)–(A5) and let $S$ be a set of $n$ points in $\mathbb{R}^2$. For simplicity, we assume that the origin lies inside $C$. For a point $p \in \mathbb{R}^2$, define $C_p = \{p - c \mid c \in C\}$. Let $\mathcal{C} = \{C_p \mid p \in S\}$. For a point set $R$ and a point $x \in \mathbb{R}^2$, the *depth* of $x$ with respect to $R$, denoted by $d_R(x)$, is the number of points $p \in R$ for which $C_p$ contains $x$. This is the same as $|R \cap (C + x)|$, so that $\kappa^*(C, S) = \max_{x \in \mathbb{R}^2} d_S(x)$. Hence, the problem of computing an optimal placement reduces to computing a point of maximum depth with respect to $S$. This reformulation has been illustrated in Figure 2.3.

Consider the arrangement $\mathcal{A}(\mathcal{C})$ defined by the boundaries of the sets $C_p$, where $p \in S$. Each vertex in $\mathcal{A}(\mathcal{C})$ is either a vertex of $C_p$, for some point $p \in S$ (a *type 1* vertex), or an intersection point of $C_p$ and $C_q$, for two points $p, q \in S$ (a *type 2* vertex). If $p$ is in the interior

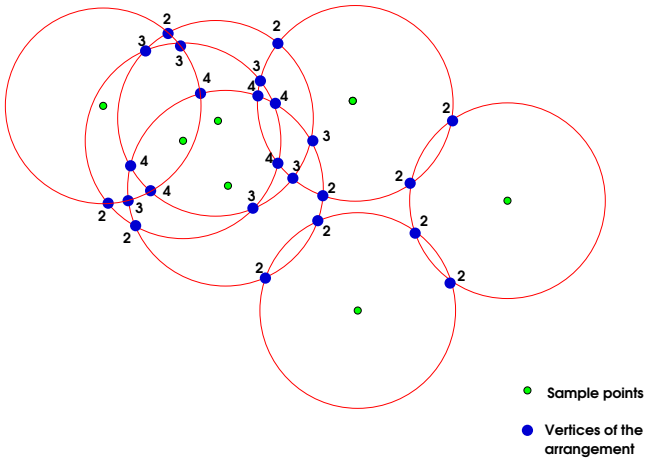


*Fig. 2.3:* Reformulation: Computing the heighest depth in an arrangement solves the optimal placement problem. The numbers in the fi gure indicates the depth of that point.

of $f$ and $q$ is on the boundary of $f$, then our assumption that $C$ is closed implies that the depth of $q$ is greater than or equal to the depth of $p$. The maximum depth of a point is realized by a type 2 vertex of $\mathcal{A}(\mathcal{C})$ (unless the maximum depth is 1). Using this observation, $\kappa^*(C, S)$ can be computed as follows as in Efrat *et al.* [ESZ94]. We give the algorithm here in order to be self-contained.

**Step 1:** For each point $p \in S$, compute the intersections between $\partial C_p$ and all other boundaries $\partial C_q$, where $q \in S$.

**Step 2:** Let $a$ be an arbitrary point on the boundary of $\partial C_q$. Sort all intersections computed in Step 1 according to the order defined by walking along the boundary of $\partial C_q$ in counter-clockwise order, starting in point $a$.

**Step 3:** Let $x_1, x_2, \ldots, x_\ell$ be the sorted sequence computed in Step 2. For $i = 1, \ldots, \ell$, we define $k_i$ to be the number of sets $\partial C_p$, with $p \in S$, that contain $x_i$.

**Step 3.1:** Compute $k_1$.

**Step 3.2:** Walk along the sequence $x_2, \ldots, x_\ell$, and compute $k_i$ for $i = 2, \ldots, \ell$. Observe that $k_i \in \{k_{i-1} - 1, k_{i-1}, k_{i-1} + 1\}$ and that $k_i - k_{i-1}$ depends on whether a set $\partial C_p$ is entered or left when we walk from $x_{i-1}$ to $x_i$.

**Step 4:** Compute an index $i$, with $1 \leq i \leq \ell$, such that $k_i$ is maximum. Return $k_i$ and the corresponding point $x_i$.

It is clear that the value $k_i$ returned by this algorithm is equal to $k_q^*$. The time for Step 1 is bounded by $O(nI(m))$. The number $\ell$ of intersection points computed in this step is less than or equal to $sn$. Hence, Step 2 takes $O(sn \log(sn))$ time. Step 3.1 takes $O(nQ(m))$ time. Finally, Steps 3.2 and 4 take $O(\ell) = O(sn)$ time. Hence, the total running time of the algorithm is $O(n(I(m) + Q(m)) + sn \log(sn))$.

As mentioned above, running this algorithm for each point $q$ of $S$ computing $\kappa^*(C, S)$ in $O(n^2(I(m) + Q(m)) + sn^2 \log(sn))$ time.

Alternatively, we can do the following:

**Step 1:** Using the algorithm of Amato *et al.* [AGR00], compute the arrangement defined by the edges of the boundaries of the sets $\partial C_p$, with $p \in S$.

**Step 2:** For each connected component of the arrangement computed in Step 1, do the following.

**Step 2.1:** Let $u$ be an arbitrary vertex of this component. Compute the number $k_u$ of sets $\partial C_p$, with $p \in S$, that contain $u$.

**Step 2.2:** Starting at $u$, traverse the vertices of the component, e.g., in depth-first order. In a generic step, we walk from a vertex $v$ to a neighboring vertex $w$. At this moment, we know the number $k_v$ of sets $\partial C_p$, with $p \in S$, that contain $v$. As in Step 3.2 of the first algorithm, we use $k_v$ to compute the number $k_w$ of sets $\partial C_p$, with $p \in S$, that contain $w$.

**Step 3:** Compute a vertex $v$ of the arrangement for which $k_v$ is maximum. Return $k_v$ and the corresponding point $v$.

In this alternative approach, we compute $\mathcal{A}(\mathcal{C})$ with the algorithm of Amato et al. [AGR00] and use a standard graph-traversal algorithm, such as depth-first search, to compute a type 2 vertex of maximum depth (with respect to $S$). Since $\mathcal{A}(\mathcal{C})$ has $O(mn + sn^2)$ vertices, this algorithm takes $O(mn \log(mn) + nQ(m) + sn^2)$ time. Hence, we obtain the following.

**Theorem 1.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). The value of $\kappa^*(C, S)$ can be computed in*

$$O\left(n^2(I(m) + Q(m)) + sn^2 \log(sn)\right)$$

*or*

$$O\left(mn \log(mn) + nQ(m) + sn^2\right)$$

*time.*

If $m = O(1)$, then $s, Q(m) = O(1)$, and if $C$ is convex, then $s = 2$ [KLPS86] and $I(m), Q(m) = O(\log m)$ [PS85]. (For the upper bounds on $I(m)$ and $Q(m)$, an $O(m)$-time preprocessing step is needed.) Therefore Theorem 1 implies the following.

**Corollary 1.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). Then $\kappa^*(C, S)$ can be computed in $O(n^2)$ time if $C$ has $O(1)$ edges, and in $O(n^2 \log(mn) + m)$ or $O(mn \log(mn) + n^2)$ time if $C$ is a convex $m$-gon.*

*Proof.* In the first case, $m$, $s$, $I(m)$ and $Q(m)$ are bounded by a constant, and the claim follows from the first result of Theorem 1. Assume that $C$ is a convex polygon with $m$ edges. Using Dobkin and Kirkpatrick's hierarchical representation of $C$ (see [DK83]), we can decide in $O(\log m)$ time whether any given point is contained in $C$. Hence, we have $Q(m) = O(\log m)$. It takes only $O(m)$ time to construct the hierarchical representation of $C$. Kedem *et al.* [KLPS86] have shown that, under our general-position assumption, the boundaries of any two translates of $C$ intersect at most twice. Furthermore, Barequet *et al.* [BDP97] have shown that, after an $O(m)$-time preprocessing, these intersection points can be computed in $O(\log m)$ time. Hence, we have $s = 2$ and $I(m) = O(\log m)$. □

### 2.4.1 Bucketing and Estimating $\kappa^*(C, S)$.

For any two positive real numbers $r$ and $r'$, we denote by $\mathcal{G}_{r,r'}$ the two-dimensional grid through the origin whose cells have horizontal and vertical sides of lengths $r$ and $r'$, respectively. Hence, each cell of $\mathcal{G}_{r,r'}$ is of the form $B_{ij} = [ir, (i+1)r) \times [jr', (j+1)r')$ for some integers $i$ and $j$. We call the pair $(i, j)$ the *index* of $B_{ij}$.

We need an algorithm that groups the points of $S$ according to the cells of some grid $\mathcal{G}_{r,r'}$, i.e., stores $S$ in a list such that for each grid cell $B$, all points of $S$ in $B$ occur together in a contiguous sublist. This operation is similar to a sorting of the elements of $S$ by their associated grid cells, but does not require the full power of sorting. Let $T_{\mathrm{g}}(n)$ denote the time needed to perform such a grouping of $n$ points according to some grid and assume that $T_{\mathrm{g}}$ is nondecreasing and *smooth* in the sense that $T_{\mathrm{g}}(O(n)) = O(T_{\mathrm{g}}(n))$ (informally, a smooth function grows polynomially). The following lemma is straightforward.

**Lemma 1.** *Let $S$ be a set of $n$ points in $\mathbb{R}^2$ and let $C$ be a disk satisfying assumptions (A1)–(A5). Let $a, b > 0$ be such that $R_0 \subseteq C \subseteq R_1$ for axes-parallel rectangles $R_0$ of width $a$ and height $b$ and $R_1$ of width $\alpha a$ and height $\beta b$. Let $M$ be the maximum number of points of $S$ contained in any cell of the grid $\mathcal{G}_{a,b}$. Then $M \leq \kappa^*(C, S) \leq (\alpha + 1)(\beta + 1)M$.*

*Proof.* Let $B_{\ell_1\ell_2}$ be a grid cell that contains $M$ points of $S$. Let $x \in \mathbb{R}^2$ be the point such that the translated rectangle $(R_0)_x$ coincides with $B_{\ell_1\ell_2}$. Then the set $C_x$ contains $(R_0)_x$ and, hence, $C_x$ contains at least $M$ points of $S$. This proves that $k^* \geq M$.

To prove the second inequality, let $x \in \mathbb{R}^2$ be a point such that the set $C_x$ contains $k^*$ points of $S$. Consider the translated rectangle $(R_1)_x$. The set $C_x$ is contained in $(R_1)_x$. It is easy to see that the number of grid cells that overlap $(R_1)_x$ is at most $(\alpha + 1)(\beta + 1)$. Since all these grid cells together contain at least $k^*$ points of $S$, one of them contains at least $k^*/((\alpha + 1)(\beta + 1))$ points of $S$. □

Lemma 1 shows that an approximation $M$ to $\kappa^* = \kappa^*(C, S)$ with $M \leq \kappa^* \leq (\alpha + 1)(\beta + 1)M$ can be computed in $O(T_{\mathrm{g}}(n))$ time. Let us see how the grouping of $S$ can be implemented. It is clear that once each point $p$ of $S$ has been mapped to the index $(i, j)$ of the cell containing $p$ of a grid under consideration, $S$ can be grouped with respect to the grid in $O(n \log n)$ time by sorting the pairs $(i, j)$ lexicographically. The mapping of points to grid indices uses the nonalgebraic floor function. To avoid this, we can replace the grid by the *degraded grid* introduced in [DLSS95, LS95], which can be constructed in $O(n \log n)$ time without using the floor function, and for which Lemma 1 also holds. Given any point $p \in \mathbb{R}^2$, the cell of the degraded grid that contains $p$ can be found in $O(\log n)$ time, so that the grouping can be completed in $O(n \log n)$ time.

In a more powerful model of computation, after mapping $S$ to grid indices, we can carry out the grouping by means of hashing. Combining the universal class of Dietzfelbinger et al. [DHKP97] with a hashing scheme of Bast and Hagerup [BH91], we obtain a Las Vegas grouping algorithm that finishes in $O(n)$ time with probability at least $1 - 2^{-n^\mu}$ for some fixed $\mu > 0$.

### 2.4.2 A Bucketing Algorithm

We can use Lemma 1 and Theorem 1 to obtain a faster algorithm for computing $\kappa^* = \kappa^*(C, S)$ in some cases. Suppose we have an algorithm $\mathcal{A}$ that computes $\kappa^*(C, S)$ in time $T(n)$.

The idea of the transformed algorithm is as follows. Let $x \in \mathbb{R}^2$ be a point such that the set $C_x$ contains $k^*$ points of $S$, and let $B_0$ be the cell of the grid $\mathcal{G}_{\alpha a,\beta b}$ that contains $x$. Then the set $C_x$ is contained in the union of $B_0$ and its eight neighboring cells. Let $\mathcal{G}$ be the grid that arises by translating $\mathcal{G}_{3\alpha a,3\beta b}$ such that when the cell $B$ of $\mathcal{G}$ that contains $x$ is divided into nine nonoverlapping subcells with horizontal and vertical side lengths $\alpha a$ and $\beta b$, respectively, then $B_0$ is the center subcell. Then the set $C_x$ is contained in $B$. Hence, by running algorithm $\mathcal{A}$ on the set $S \cap B$, we find a translate of $C$ that contains $k^*$ points of $S$. Observe that $S \cap B$ contains at most $9\alpha\beta M \le 9\alpha\beta k^*$ points. Our transformed algorithm will run algorithm $\mathcal{A}$ on each cell $B'$ that contains at least $M$ points of $S$.

The transformed algorithm uses a *semisorting algorithm* that does the following. Given the set $S$ and two positive real numbers $x$ and $y$, the algorithm returns a list $L$ containing the points of $S$ such that all points that belong to the same cell of the grid $\mathcal{G}_{x,y}$ form a contiguous sublist of $L$. We denote the running time of this semisorting algorithm by $T_{\mathrm{s}}(n)$. In Section 2.5.4, we will describe two different semisorting algorithms.

The transformed algorithm does the following.

**Step 1:** Compute the value $M$, defined as the maximum number of points of $S$ that are contained in any cell of the grid $\mathcal{G}_{a,b}$.

**Step 2:** For any two integers $i$ and $j$ with $0 \le i \le 2$ and $0 \le j \le 2$, do the following.

**Step 2.1:** Run the semisorting algorithm on $S$, based on the grid $\mathcal{G} := \mathcal{G}^{ij}_{3\alpha a,3\beta b}$, which is obtained by shifting $\mathcal{G}_{3\alpha a,3\beta b}$ by the vector $(i\alpha a, j\beta b)$.

**Step 2.2:** For each cell $B$ of $\mathcal{G}$ that contains at least $M$ points of $S$, run $\mathcal{A}$ on the set $S \cap B$ and let $k_B$ and $x_B$ be its output. Hence, the set $C_{x_B}$ contains $k_B$ points of $S \cap B$.

**Step 3:** Compute indices $i$ and $j$ and a cell $B$ of $\mathcal{G}^{ij}_{3\alpha a,3\beta b}$ such that $k_B$ is maximum. Return $k_B$ and the corresponding point $x_B$.

Let us analyze the running time. For each of $0 \le i, j \le 2$, the algorithm spends $T_{\mathrm{g}}(n)$ time to partition $S$ among the grid cells. Since at most $n/M$ cells of $\mathcal{G}^{ij}$ contain at least $M$ points of $S$ and no cell contains more than $9\alpha\beta M$ points, the total running time is $O(T_{\mathrm{g}}(n) + nT(\alpha\beta M)/M) = O(T_{\mathrm{g}}(n) + nT(\alpha\beta\kappa^*)/\kappa^*)$, where in the last step we used the relation $M \le \kappa^*$ and the assumption that $T(n)/n$ is nondecreasing. We thus obtain the following result.

**Theorem 2.** *Let $S$ be a set of $n$ points in the plane, let $C$ be a disk satisfying assumptions (A1)–(A5), and let $\mathcal{A}$ be an algorithm that computes $\kappa^* = \kappa^*(C, S)$ in time $T(n)$. Then $\kappa^*$ can be computed in $O(T_{\mathrm{g}}(n) + nT(\alpha\beta\kappa^*)/\kappa^*)$ time.*

**Corollary 2.** *Let $C, S$, and $T(\cdot)$ be as in Theorem 2. The value of $\kappa^* = \kappa^*(C, S)$ can be computed in $O(T_{\mathrm{g}}(n) + n\kappa^*)$ time if $C$ is fat and has $O(1)$ edges, and in $O(T_{\mathrm{g}}(n) + n\kappa^* \log(m\kappa^*) + m)$ or $O(T_{\mathrm{g}}(n) + mn \log(m\kappa^*) + n\kappa^*)$ time if $C$ is a convex $m$-gon.*

## 2.5 Monte-Carlo Algorithms

In this section we present Monte-Carlo algorithms for the $\varepsilon$-optimal-placement problem. These algorithms use one of the deterministic algorithms described in Theorem 1 as a subroutine. We will refer to this algorithm as $\mathcal{A}$ and to its running time as $T(n)$. We assume that $T(n)$ and $T(n)/n$ are nondecreasing and that $T$ is smooth.

### 2.5.1 A Random Sampling Approach

We first present an algorithm based on the *random-sampling* technique. We carry out the probabilistic analysis using a variant of the well-known Chernoff bounds (see, e.g., Hagerup and Rüb [HR90]).

Let $\epsilon$ and $\gamma$ be real numbers with $0 < \epsilon < 1$ and $0 < \gamma \leq 1$. The value of $\epsilon$ gives a trade-off between the approximation ratio and the error probability of the transformed algorithm, whereas $\gamma$ gives a trade-off between the running time and the error probability.

The transformed algorithm does the following when given a set $S$ of $n$ points in the plane.

**Step 1:** Generate a $\gamma$-*sample* $S'$ of $S$, i.e., a random sample obtained by choosing each point of $S$ with probability $\gamma$ and independently of the other points.

**Step 2:** If $S' = \emptyset$ or $|S'| \geq 2\gamma n$, then let $y$ be a point on the boundary of one of the sets $C_p$, with $p \in S$. Otherwise, use $\mathcal{A}$ to compute a point $y \in \mathbb{R}^2$ such that $C_y$ contains the maximum number of points of $S'$.

**Step 3:** Compute the number $k_y$ of points of $S$ contained in $C_y$. Return $k_y$ and the point $y$.

It is clear that the running time of the transformed algorithm is $O(T(\gamma n) + nQ(m))$.

Consider the value $k_y$ and the point $y$ returned by the algorithm and note that $k_y$ and $y$ are random variables. The error probability of our algorithm is $\Pr(k_y < (1-\epsilon)k^*)$. In the rest of this section, we prove an upper bound on this quantity.

Recall that $k^*$ is the maximum depth of any point in the arrangement defined by the boundaries of the sets $C_p$, with $p \in S$.

We will use the following variants of the Chernoff bounds. The proof of the first claim can be found in Hagerup and Rüb [HR90]. The proof of the second claim is given here.

**Lemma 2.** *Let $Y$ be a binomially distributed random variable and let $0 \leq \lambda \leq 1$.*

1. *For every $t \leq E(Y)$, $\Pr[Y \leq (1-\lambda)t] \leq e^{-\lambda^2 t/2}$.*
2. *For every $t \geq E(Y)$, $\Pr[Y \geq (1+\lambda)t] \leq e^{-\lambda^2 t/3}$.*

*Proof.* Let $N$ be a positive integer and let $\gamma$ be a real number such that $T$ can be interpreted as the number of heads in a sequence of $N$ coin flips, each one coming up heads with probability $\gamma$.

Let $t \geq E(T)$. If $t > N$, then $\Pr(T \geq (1+\delta)t) = 0$, and the claim clearly holds. So assume that $t \leq N$. Let $\gamma' := t/N$ and let $T'$ be binomially distributed with parameters $N$ and $\gamma'$. Then $E(T') = \gamma' N = t$. A variant of the Chernoff bounds (see Hagerup and Rüb [HR90]) states that

$$\Pr(T' \geq (1+\delta)t) \leq e^{-\delta^2 t/3}.$$

We have $\gamma = E(T)/N \leq t/N = \gamma'$. The proof of the second claim in Lemma 2 follows from the fact that

$$\Pr(T \geq (1+\delta)t) \leq \Pr(T' \geq (1+\delta)t). \tag{2.1}$$

To prove (2.1), consider two coins. The first one comes up heads with probability $\gamma/\gamma'$, whereas the second one comes up heads with probability $\gamma'$. For each $i$, $1 \leq i \leq N$, we do the following experiment: Flip both coins and define two random variables $U_i$ and $U_i'$. The value of $U_i$ is one if both coins come up heads and zero otherwise. The value of $U_i'$ is one if the second coin comes up heads and zero otherwise. We have $\Pr(U_i = 1) = \gamma$ and $\Pr(U_i' = 1) = \gamma'$. Since $U_i \leq U_i'$ for all $i$, we have

$$\Pr\left(\sum_{i=1}^{N} U_i \geq (1+\delta)t\right) \leq \Pr\left(\sum_{i=1}^{N} U_i' \geq (1+\delta)t\right).$$

This implies that (2.1) holds, because $\sum_{i=1}^{N} U_i$ and $\sum_{i=1}^{N} U_i'$ have the same probability distributions as $T$ and $T'$, respectively. □

**Theorem 3.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). For arbitrary $\varepsilon, \gamma > 0$, an $\varepsilon$-approximate solution to the optimal-placement problem can be computed in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2\gamma\kappa^*}$, where $\kappa^* = \kappa^*(C, S)$.*

*Proof.* Let $\bar{\varepsilon} = \min\{\varepsilon, 1/2\}$ and $\bar{\gamma} = \min\{288\gamma, 1\}$. The algorithm first draws a $\bar{\gamma}$-sample $S'$ of $S$, i.e., includes every point of $S$ in $S'$ with probability $\bar{\gamma}$ and independently of all other points. If $|S'| > 2\bar{\gamma}n$ (the sampling *fails*), the algorithm returns an arbitrary point. Otherwise it uses $\mathcal{A}$ to return a point $y$ of maximum depth with respect to $S'$.

Since $\bar{\gamma} = O(\gamma)$ and $T$ is smooth, it is clear that the algorithm can be executed in $O(n + T(\gamma n))$ time. By Lemma 2, the sampling fails with probability at most $e^{-\bar{\gamma}n/3}$. If $\varepsilon \geq 1$ or $\bar{\gamma} = 1$, the output is obviously correct. Assume that this is not the case and that the sampling succeeds.

Let us write $d$ for $d_S$ and $d'$ for $d_{S'}$ and let $x \in \mathbb{R}^2$ be a point with $d(x) = \kappa^*$. Informally, our proof proceeds as follows. Let $Z = \{z \in \mathbb{R}^2 \mid d(z) < (1-\varepsilon)\kappa^*\}$ be the set of "bad" points. The error probability is equal to $\Pr[y \in Z]$. We first show that $d'(y)$ is likely to be large, where "large" means at least $(1-\bar{\varepsilon}/2)\bar{\gamma}\kappa^*$. Subsequently we show that for every $z \in Z$, $d'(z)$ is not likely to be large. Combining the two assertions shows that except with small probability, $y \notin Z$.

The first part is easy: Since $E(d'(x)) = \bar{\gamma}\kappa^*$ and $d'(y) \geq d'(x)$, Lemma 2 implies that

$$\Pr[d'(y) < (1-\bar{\varepsilon}/2)\bar{\gamma}\kappa^*] \leq e^{-\bar{\varepsilon}^2\bar{\gamma}\kappa^*/8}.$$

Now fix $z \in Z$. Since $1 - \bar{\varepsilon}/2 \geq (1+\bar{\varepsilon}/2)(1-\bar{\varepsilon})$ and $E(d'(z)) = \bar{\gamma}d(z) < (1-\varepsilon)\bar{\gamma}\kappa^* \leq (1-\bar{\varepsilon})\bar{\gamma}\kappa^*$, we have

$$\Pr[d'(z) \geq (1-\bar{\varepsilon}/2)\bar{\gamma}\kappa^*] \leq \Pr[d'(z) \geq (1+\bar{\varepsilon}/2)(1-\bar{\varepsilon})\bar{\gamma}\kappa^*] \leq e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12}.$$

The preceding argument applies to a fixed $z \in Z$. A priori, we have to deal with an infinite

number of candidate points $z \in Z$. However, using the fact that the arrangement defined by the boundaries of the sets $C_p$, with $p \in S$, has $O(sn^2)$ vertices of type 2, it is not difficult to see that there is a set $X$ with $|X| = O(sn^2)$ such that for every $z \in Z$, there is a $\hat{z} \in X \cap Z$ with $d'(\hat{z}) = d'(z)$. Therefore the probability that $d'(z) \geq (1 - \bar{\varepsilon}/2)\bar{\gamma}\kappa^*$ for some $z \in Z$ is $O(sn^2 e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12})$. The other failure probabilities identified above are no larger. Now, $\bar{\varepsilon} \geq \varepsilon/2$, $1 - \bar{\varepsilon} \geq 1/2$, and $\bar{\gamma} = 3 \cdot 8 \cdot 12\gamma$. Moreover, we can assume that $e^{\varepsilon^2\gamma\kappa^*} \geq sn$, since otherwise the theorem claims nothing. But then $sn^2 e^{-\bar{\varepsilon}^2(1-\bar{\varepsilon})\bar{\gamma}\kappa^*/12} \leq sn^2 e^{-3\varepsilon^2\gamma\kappa^*} \leq (1/s)e^{-\varepsilon^2\gamma\kappa^*}$. Therefore, except if $n$ is bounded by some constant (in which case we can use a deterministic algorithm), the failure probability is at most $sne^{-\varepsilon^2\gamma\kappa^*}$. □

Combining Theorem 3 with Corollary 1, we obtain the following result.

**Corollary 3.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). For arbitrary $\varepsilon, \gamma > 0$, an $\varepsilon$-approximate placement can be computed with error probability at most $ne^{-\varepsilon^2\gamma\kappa^*}$ in $O(n + (\gamma n)^2)$ time if $C$ has $O(1)$ edges, and in $O(n + (\gamma n)^2 \log(\gamma mn) + m)$ or $O(n + \gamma mn \log(\gamma mn) + (\gamma n)^2)$ time if $C$ is a convex $m$-gon.*

Our random-sampling approach can also be used to solve related problems. As an example, consider the problem of computing a point of maximum depth in a set $H$ of $n$ halfplanes. If we denote this maximum depth by $\kappa^*$, then $\kappa^* \geq n/2$. By computing and traversing the arrangement defined by the bounding lines of the halfplanes, one can compute $\kappa^*$ in $O(n^2)$ time. Since a corresponding decision problem is 3SUM-hard (see Gajentaan and Overmars [GO95]), it is unlikely that it can be solved in subquadratic time. If we apply our random-sampling transformation with $\gamma = c/\sqrt{n}$ for a suitable constant $c > 0$, we obtain the following result.

**Theorem 4.** *Let $H$ be a set of $n$ halfplanes. For arbitrary constant $\varepsilon > 0$, in $O(n)$ time we can compute a point in $\mathbb{R}^2$ whose depth in $H$ is at least $(1 - \varepsilon)\kappa^*$, except with probability at most $e^{-\sqrt{n}}$.*

### 2.5.2 Bucketing and Sampling Combined

We now present a transformed Monte Carlo algorithm that combines Theorem 3 with the bucketing algorithm described in Section 2.4.

First compute $M$, as defined in Lemma 1. Next, for each $(i, j) \in \{0, 1, 2\}^2$, consider the grid $\mathcal{G}^{ij}$ as in Section 2.4. Fix a parameter $\gamma \geq 0$. For each cell $B$ of $\mathcal{G}^{ij}$ with $|S \cap B| \geq M$, run the algorithm described in Section 2.5.1 on the set $S \cap B$ to obtain a point $y_B$ and compute the value $k_B = d_{S\cap B}(y_B)$. Finally return a point $y_B$ for which $k_B$ is maximum.

The above algorithm immediately leads to the following results.

**Theorem 5.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). For arbitrary $\varepsilon, \gamma > 0$, an $\varepsilon$-approximate placement of $C$ can be computed in $O(T_g(n) + nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ time by a Monte Carlo algorithm with error probability at most $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$.*

**Corollary 4.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a disk satisfying assumptions (A1)–(A5). For arbitrary constant $\varepsilon > 0$, an $\varepsilon$-approximate placement of $C$ can be computed in $O(n \log n)$ time with probability of error at most $e^{-\sqrt{\kappa^* \log n}}$ if $C$ is fat and has $O(1)$ edges,*

*and in $O(n\log(mn)+m)$ time with probability of error at most $e^{-\sqrt{\kappa^*\log(mn)/\log(m\kappa^*)}}$ if $C$ is a convex $m$-gon.*

*Proof.* To prove the first claim, let $\mathcal{A}$ be the first algorithm of Corollary 1. Then $T(n) = O(n^2)$. Applying Theorem 5 to $\mathcal{A}$, we obtain an $\varepsilon$-approximation algorithm for the optimal-placement problem with running time $O(n\log n+\gamma^2 n\kappa^*)$ and error probability $O(\kappa^* e^{-\varepsilon^2\gamma\kappa^*})$. If we choose $\gamma = (2/\varepsilon^2)\sqrt{(\log n)/M}$, where $M$ is as in Lemma 1, the running time and the error probability are as claimed for $n$ larger than some constant.

For the proof of the second claim, we take $\mathcal{A}$ to be the second algorithm of Corollary 1. Then $T(n) = O(n^2\log(mn)+m)$. If we apply Theorem 5 to $\mathcal{A}$, we obtain an $\varepsilon$-approximation algorithm for the optimal-placement problem with running time

$$O(n\log(mn) + n\gamma^2\kappa^*\log(m\gamma\kappa^*) + m)$$

and error probability

$$O(\kappa^* e^{-\varepsilon^2\gamma\kappa^*})$$

. We choose $\gamma = c\sqrt{\log(mn)/(M\log(mM))}$, where $M$ is as in Lemma 1 and $c$ is a sufficiently large constant. Except if $\gamma > 1$, in which case we can use the second algorithm of Corollary 2, this gives the running time and the error probability claimed for $n$ sufficiently large. □

### 2.5.3 Linear Time Algorithm

Here we show that an additional random-sampling step reduces the running time in Theorem 5 at the expense of a larger error probability.

Let $\mathcal{A}$ be any deterministic algorithm that computes $\kappa^*$ and let $T$ denote its worst-case running time. As before, we assume that $T(n)$ and $T(n)/n$ are non-decreasing and that $T$ is smooth. For any real number $0 < \gamma \leq 1$, let $\mathcal{A}_\gamma$ be the Monte Carlo approximation algorithm obtained by applying Theorem 3 to $\mathcal{A}$.

Let $S$ be a set of $n$ points in the plane and let $0 < \gamma \leq 1$, $0 < \gamma' \leq 1$ and $0 < \epsilon \leq 1$ be real numbers. Our new algorithm does the following. First, it generates a $\gamma'$-sample $S'$ of $S$. Then it runs a slightly altered version of the algorithm of Theorem 5 on the set $S'$, using the parameter $\gamma$. To be more precise, the algorithm makes the following steps.

**Step 1:** Generate a $\gamma'$-sample $S'$ of $S$.

**Step 2:** If $S' = \emptyset$ or $|S'| \geq 2\gamma' n$, then let $y$ be a point on the boundary of one of the sets $C_p$, with $p \in S$. Compute the number $k_y$ of points of $S$ contained in $C_y$. Return $k_y$ and $y$, and terminate. Otherwise, proceed with Step 3.

**Step 3:** Compute the value $M'$, defined as the maximum number of points of $S'$ that are contained in any cell of the grid $\mathcal{G}_{a,b}$.

**Step 4:** For any two integers $i$ and $j$ with $0 \leq i \leq 2$ and $0 \leq j \leq 2$, do the following.

**Step 4.1:** Run the semisorting algorithm on $S'$, based on the grid $\mathcal{G} := \mathcal{G}^{ij}_{3\alpha a,3\beta b}$, which is obtained by shifting $\mathcal{G}_{3\alpha a,3\beta b}$ by the vector $(i\alpha a, j\beta b)$.

**Step 4.2:** For each cell $B$ of $\mathcal{G}$ that contains at least $M'/(18\alpha\beta)$ points of $S'$, run $\mathcal{A}_\gamma$ on the set $S' \cap B$. Let $k'_B$ and $x_B$ be its output. Hence, the set $C_{x_B}$ contains $k'_B$ points of $S' \cap B$.

**Step 5:** Compute indices $i$ and $j$ and a cell $B$ of $\mathcal{G}^{ij}_{3\alpha a, 3\beta b}$ such that $k'_B$ is maximum. Compute the number $k_B$ of points of $S$ contained in $C_{x_B}$. Return $k_B$ and $x_B$.

The algorithm described above suggests the following improvements.

**Theorem 6.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a fat disk satisfying assumptions (A1)–(A5) and having $O(1)$ edges. For arbitrary constant $\varepsilon > 0$, an $\varepsilon$-approximate placement of $C$ can be computed in $O(n)$ time with error probability at most $ne^{-\sqrt{\kappa^*}}$.*

*Proof.* The algorithm described above, can be thought of as sampling followed by bucketing followed by sampling: Essentially the algorithm draws a random $\gamma$-sample $S'$ of $S$, where $\gamma = \min\{L/\log n, 1\}$ for a constant $L > 0$ to be chosen below. If $|S'| > 2\gamma n$, the algorithm returns an arbitrary point. Otherwise we apply the first algorithm of Corollary 4 to $S'$, but with approximation parameter $\varepsilon/4$, rather than $\varepsilon$, and return the point returned by that algorithm. The overall running time is clearly $O(n)$.

As in the proof of Theorem 3, we write $d$ for $d_S$ and $d'$ for $d_{S'}$. Assume that $\varepsilon \leq 1$ and that $|S'| \leq 2\gamma n$ and let $\kappa'$ be the maximum value of $d'(x)$ over all points $x$ in the plane. By Lemma 2, $\Pr[\kappa' < (1-\varepsilon/4)\gamma\kappa^*] \leq e^{-c_1\gamma\kappa^*}$ for some constant $c_1 > 0$. Moreover, the analysis in the proof of Theorem 3 shows the probability that $d'(z) \geq (1-\varepsilon/2)\gamma\kappa^*$ for some $z \in \mathbb{R}^2$ with $d(z) < (1-\varepsilon)\kappa^*$ to be $O(ne^{-c_2\gamma\kappa^*})$ for some other constant $c_2 > 0$. Finally, the probability that the algorithm of Corollary 4 returns a point $y$ with $d'(y) < (1-\varepsilon/4)\kappa'$ is at most $e^{-\sqrt{\kappa' \log n}}$. Observe that $\kappa' \geq (1-\varepsilon/4)\gamma\kappa^*$ and $d'(y) \geq (1-\varepsilon/4)\kappa'$ imply $d'(y) \geq (1-\varepsilon/2)\gamma\kappa^*$. Therefore, for some constant $c > 0$, the answer is correct, except with probability $O(ne^{-c\gamma\kappa^*} + e^{-c\sqrt{\gamma\kappa^* \log n}})$. If $\gamma < 1$, the probability under consideration is $O(ne^{-cL\kappa^*/\log n} + e^{-c\sqrt{L\kappa^*}})$. We can assume that $\kappa^* \geq (\log n)^2$, since otherwise there is nothing to prove. But then it is clear that for $L$ chosen sufficiently large and for $n$ larger than some constant, the error probability is at most $ne^{-\sqrt{\kappa^*}}$. □

### 2.5.4 Implementing the Semisorting Algorithm

Let $x$ and $y$ be two positive real numbers. A first implementation of the semisorting algorithm, based on the grid $\mathcal{G}_{x,y}$, is as follows. For each point $p = (p_1, p_2)$ of $S$, compute the index $(\ell_1^p, \ell_2^p)$ of the grid cell that contains $p$. Note that $\ell_1^p = \lfloor p_1/x \rfloor$ and $\ell_2^p = \lfloor p_2/y \rfloor$. Then sort all indices $(\ell_1^p, \ell_2^p)$, with $p \in S$, lexicographically. It is clear that in the sorted sequence, all points of $S$ that are in the same grid cell form a contiguous subsequence. The running time of this algorithm is bounded by $O(n \log n)$. The algorithm uses the non-algebraic floor function to compute indices of grid cells. We can replace the grid $\mathcal{G}_{x,y}$ by the *degraded grid* of [DLSS95, LS95]. This is basically a collection of $O(n)$ horizontal and vertical lines that partition $\mathbb{R}^2$ into axes-parallel rectangles. The horizontal and vertical sides of any such rectangle have lengths at least $x$ and $y$, respectively. Moreover, if the rectangle contains one or more points of $S$, then these side lengths are equal to $x$ and $y$, respectively. In [DLSS95, LS95], it is shown how such a degraded grid can be computed in $O(n \log n)$ time without using the floor function. Moreover, given any point $p \in \mathbb{R}^2$, we can find the cell of the degraded grid that contains $p$ in $O(\log n)$ time. Hence, using this degraded grid instead of the grid $\mathcal{G}_{x,y}$, we

obtain a semisorting algorithm with a running time of $O(n \log n)$ and working in the algebraic computation-tree model. Note that Lemma 1 also holds for the degraded grid.

In a more powerful computation model, we use the grid $\mathcal{G}_{x,y}$ and implement the semisorting algorithm using hashing. In this extension of the algebraic computation-tree model, each of the following operations can be performed in unit-time: Computing the floor and logarithm of any real number, computing $2^m$ for any positive integer $m$, and indirect addressing.

Consider again the sequence $(\ell_1^p, \ell_2^p)$, $p \in S$, of indices. Let $M_1 := \max_{p \in S} |\ell_1^p|$, $M_2 := \max_{p \in S} |\ell_2^p|$, and $M := 2\max(M_1, M_2) + 1$. Let $I$ be the set consisting of the integers $\ell_1^p + \ell_2^p M$. We realize the semisorting through a hash function that maps the set $I$ injectively to a set of non-negative integers that are bounded by $O(n)$. After the hashing has been completed, the semisorting can be carried out in $O(n)$ time.

One complication for the hashing is that the universe containing the keys to be hashed is not fixed in advance, but data-dependent. For this reason hashing schemes that embed the universe in a finite field—such as the celebrated scheme of Fredman *et al.* [FKS84]—may be problematic due to the necessity of computing large primes. We avoid this problem by combining the universal class of Dietzfelbinger *et al.* [DHKP97] with a hash function of Bast and Hagerup [BH91]. The entire semisorting runs in at most $cn$ time, for some constant $c$, with probability at least $1 - 2^{-n^\delta}$, where $\delta$ is a positive constant.

In the algorithms of Sections 2.5.2 and 2.5.3, we run the semisorting algorithm ten times. If one of these runs does not terminate within $cn$ time, then we stop the algorithm, take an arbitrary point $y \in \mathbb{R}^2$, compute the number $k_y$ of points of $S$ that are contained in $C_y$, and return $k_y$ and $y$. In this way, the error probability increases by an additive factor of $10 \cdot 2^{-n^\delta}$.

## 2.6 A Deterministic Approximation Algorithm

In this section we present a deterministic algorithm for the $\varepsilon$-optimal-placement problem. For simplicity, we assume that $C$ is convex and has $O(1)$ edges. The algorithm can be extended to more general cases. For example, at the cost of an extra $O(\log m)$-factor in the running time, the algorithm can be extended to arbitrary convex $m$-gons. Throughout this section, $S$ denotes a set of $n$ points in the plane, $\mathcal{C}$ denotes the set of $n$ translates $C_p$, with $p \in S$, $\mathcal{A}(\mathcal{C})$ denotes the arrangement defined by the boundaries of the elements of $\mathcal{C}$, and $\kappa^*$ denotes $\kappa^*(C, S)$.

### 2.6.1 Cuttings

We will refer to a simply connected region with at most four edges (left and right edges being vertical segments and top and bottom edges being portions of the boundaries of translates of $-C$) as a *pseudo-trapezoid*. For technical reasons, we will also regard vertical segments and portions of the boundaries of translates of $-C$ as 1-dimensional pseudo-trapezoids. For a pseudo-trapezoid $\Delta$ and a set $\mathcal{F}$ of translates of $-C$, we will use $\mathcal{F}_\Delta \subseteq \mathcal{F}$ to denote the set of all elements of $\mathcal{F}$ whose boundaries intersect $\Delta$. The vertical decomposition $\mathcal{A}^\parallel(\mathcal{C})$ of $\mathcal{A}(\mathcal{C})$ partitions the plane into pseudo-trapezoids. Let $\mathcal{F}$ be a subset of $\mathcal{C}$ and let $\Delta$ be a pseudo-trapezoid. We will denote by $\chi(\mathcal{F}, \Delta)$ the number of pairs of elements of $\mathcal{F}$ whose boundaries intersect inside $\Delta$. If $\Delta$ is the entire plane, we use the notation $\chi(\mathcal{F})$ to denote $\chi(\mathcal{F}, \Delta)$, for brevity. Given a parameter $r \geq 1$, a partition $\Xi$ of $\Delta$ into a collection of pairwise

openly-disjoint pseudo-trapezoids is called a $(1/r)$-*cutting* of $(\mathcal{F}, \Delta)$ if $|\mathcal{F}_\tau| \leq |\mathcal{F}|/r$ for every pseudo-trapezoid $\tau \in \Xi(\mathcal{F}, \Delta)$. The *conflict list* of a pseudo-trapezoid $\tau$ in $\Xi$ is the set $\mathcal{F}_\tau$.

We state the following technical result in full generality, since it is of independent interest and may find additional applications. We apply it here only with $\Delta$ being the whole plane.

**Theorem 7.** *Let $\Delta$ be a pseudo-trapezoid, let $r \geq 1$, and let $\delta > 0$ be an arbitrarily small constant. A $(1/r)$-cutting of $(\mathcal{C}, \Delta)$ of size $O(r^{1+\delta} + \chi(\mathcal{C}, \Delta)r^2/n^2)$, along with the conflict lists of its pseudo-trapezoids, can be computed in $O(nr^\delta + \chi(\mathcal{C}, \Delta)r/n)$ time, where the constants of proportionality depend on $\delta$.*

We prove this theorem by adapting Chazelle's cutting algorithm [Cha93] to our setting. We call a subset $\mathcal{F}$ of $\mathcal{C}$ a $(1/r)$-*approximation* if, for every pseudo-trapezoid $\Delta$,

$$\left| \frac{|\mathcal{C}_\Delta|}{|\mathcal{C}|} - \frac{|\mathcal{F}_\Delta|}{|\mathcal{F}|} \right| < \frac{1}{r}.$$

A result by Brönnimann et al. [BCM99] implies that a $(1/r)$-approximation of $\mathcal{C}$ of size $O(r^2 \log r)$ can be computed in time $nr^{O(1)}$. Moreover, an argument similar to the one in [Cha93] implies the following:

**Lemma 3.** *Let $\mathcal{F}$ be a $(1/r)$-approximation of $\mathcal{C}$. For every pseudo-trapezoid $\Delta$,*

$$\left| \frac{\chi(\mathcal{C}, \Delta)}{|\mathcal{C}|^2} - \frac{\chi(\mathcal{F}, \Delta)}{|\mathcal{F}|^2} \right| < \frac{1}{r}.$$

Next, we call a subset $\mathcal{H}$ of $\mathcal{C}$ a $(1/r)$-*net* of $\mathcal{C}$ if, for any pseudo-trapezoid $\Delta$, $|\mathcal{C}_\Delta| > |\mathcal{C}|/r$ implies that $\mathcal{H}_\Delta \neq \emptyset$. A $(1/r)$-net $\mathcal{H}$ is called *sparse* for $\Delta$ if

$$\frac{\chi(\mathcal{H}, \Delta)}{\chi(\mathcal{C}, \Delta)} \leq 4 \left( \frac{|\mathcal{H}|}{n} \right)^2.$$

As in [Cha93], we can prove the following.

**Lemma 4.** *Given a pseudo-trapezoid $\Delta$ and a parameter $r \geq 1$, we can compute, in $n^{O(1)}$ time, a $(1/r)$-net of $\mathcal{C}$ having size $O(r \log n)$ and that is sparse for $\Delta$.*

**Proof of Theorem 7.** Using Lemmas 3 and 4, we compute a $(1/r)$-cutting of $(\mathcal{C}, \Delta)$ as follows. Let $c$ be a sufficiently large constant whose value will be chosen later. For every $0 \leq j \leq \lceil \log_c r \rceil$, we compute a $(1/c^j)$-cutting $\Xi_j$ of $(\mathcal{C}, \Delta)$. The final cutting is a $(1/r)$-cutting of $(\mathcal{C}, \Delta)$. While computing $\Xi_j$, we also compute the conflict lists of the pseudo-trapezoids in $\Xi_j$.

$\Xi_0$ is $\Delta$ itself, and $\mathcal{C}$ is the conflict list of $\Delta$. We compute $\Xi_j$ from $\Xi_{j-1}$ as follows. For each pseudo-trapezoid $\tau \in \Xi_{j-1}$, if $|\mathcal{C}_\tau| \leq n/c^j$, then we do nothing in $\tau$. Otherwise, we first compute a $(1/2c)$-approximation $\mathcal{F}_\tau$ of $\mathcal{C}_\tau$ of size $O(c^2 \log c)$ and then a $(1/2c)$-net $\mathcal{H}_\tau$ of $\mathcal{F}_\tau$ of size $O(c \log c)$ that is sparse for $\tau$. Note that $\mathcal{H}_\tau$ is a $(1/c)$-net of $\mathcal{C}_\tau$. We then compute the vertical decomposition $\mathcal{A}^{\shortparallel}(\mathcal{H}_\tau)$ of $\mathcal{A}(\mathcal{H}_\tau)$ within $\tau$. $\mathcal{A}^{\shortparallel}(\mathcal{H}_\tau)$ consists of $O(|\mathcal{H}_\tau| + \chi(\mathcal{H}_\tau, \tau))$ cells. We replace $\tau$ with the pseudo-trapezoids of $\mathcal{A}^{\shortparallel}(\mathcal{H}_\tau)$. Repeating this for all $\tau \in \Xi_{j-1}$, we form $\Xi_j$ from $\Xi_{j-1}$. It is easy to see that $\Xi_j$ is a $(1/c^j)$-cutting of $(\mathcal{C}, \Delta)$.

By an analysis similar to the one in [Cha93], it can be shown that by choosing the constant $c$ sufficiently large (but depending on $\delta$), the size of the final cutting is $O(r^{1+\delta}+\chi(\mathcal{C},\Delta)r^2/n^2)$ and that the running time of the algorithm is $O(nr^\delta+\chi(\mathcal{C},\Delta)r/n)$. This completes the proof of Theorem 7.

By a result of Sharir [Sha91], $\chi(\mathcal{C},\Delta)=O(n\kappa^*)$, so Theorem 7 implies the following.

**Corollary 5.** *Let $r\geq 1$ and let $\delta>0$ be an arbitrarily small constant. A $(1/r)$-cutting $\Xi(\mathcal{C})$ of size $O(r^{1+\delta}+\kappa^* r^2/n)$, along with the conflict lists, can be computed in $O(nr^\delta+\kappa^* r)$ time.*

### 2.6.2 The Approximation Algorithm

Let $\delta,\varepsilon>0$ be real numbers. We now present a deterministic $\varepsilon$-approximation algorithm. We will need the following lemma.

**Lemma 5.** *Let $r\geq 1$ and let $\Xi$ be a $(1/r)$-cutting of $\mathcal{C}$. Then we can compute a point of depth (with respect to $S$) at least $\kappa^*-n/r$ in $O(|\Xi|n/r)$ time.*

*Proof.* Let $\Delta$ be a pseudo-trapezoid of $\Xi$. The maximum depth (with respect to $S$) of any point inside $\Delta$ is at most $n/r$ plus the depth (with respect to $S$) of any vertex of $\Delta$. It thus suffices to return a vertex of (a pseudo-trapezoid of) $\Xi$ of maximum depth (with respect to $S$). We can compute the depths of all vertices of $\Xi$ by following an Eulerian tour on the dual graph of the planar subdivision induced by $\Xi$; see, e.g., [AASS93]. As shown in [AASS93], the time spent in this step is proportional to the total size of all the conflict lists in $\Xi$. Since the size of each conflict list is at most $n/r$, the claim follows. □

Our algorithm works in two stages. In the first stage, we estimate the value of $\kappa^*$ to within a factor of 9, and then we use Lemma 5 to compute an $\varepsilon$-approximation of $\kappa^*$.

I. Using the bucketing algorithm of Section 2.4, we obtain a coarse estimate $k_0$ of $\kappa^*$ that satisfies $k_0/9\leq\kappa^*\leq k_0$. (Since we assume that $C$ is convex, we have $\alpha=\beta=2$, as follows from [SFRW98], which leads to the constant 9 in the estimate above.)

II. We set $r=\min\left\{\frac{9n}{\varepsilon k_0},n\right\}$, compute a $(1/r)$-cutting $\Xi$ of $\mathcal{C}$, and return a point of maximum depth (with respect to $S$) among the vertices of $\Xi$. Denote this maximum depth by $k$.

By Lemma 5, and assuming that $r=\frac{9n}{\varepsilon k_0}$,

$$k\geq\kappa^*-\frac{n}{r}=\kappa^*-\frac{n}{9n/(\varepsilon k_0)}=\kappa^*-\frac{\varepsilon k_0}{9}\geq(1-\varepsilon)\kappa^*.$$

If $r=n$, then $k\geq\kappa^*-n/r=\kappa^*-1$, which is at least $(1-\varepsilon)\kappa^*$, since we may assume that $\varepsilon\geq 1/\kappa^*$.

As shown in Section 2.4, Step I can be implemented in $O(n\log n)$ time. Using Theorem 7 and Lemma 5, Step II takes time

$$O\left(nr^\delta+\frac{\kappa^* n}{\varepsilon k_0}\right)=O\left(n^{1+\delta}+\frac{n}{\varepsilon}\right).$$

Hence, we conclude the following.

**Theorem 8.** *Let $S$ be a set of $n$ points in the plane and let $C$ be a convex disk with $O(1)$ edges. Assume that assumptions (A1)–(A5) are satisfied and let $\varepsilon > 0$ be a real number. An $\varepsilon$-approximate placement of $C$ can be computed in $O(n^{1+\delta} + n/\epsilon)$ time, for any given constant $\delta > 0$.*

## 2.7 Deterministic Algorithm by Approximating the Radius of the Disk

We present a different approximation scheme here. Instead of, approximating the number of points as done so far in this chapter, we approximate the radius of the unit disk $C$. We give a simple algorithm which in $O(n/\epsilon^2)$ time determines a placement $(x, y)$ of a disc $C^{1+\epsilon}$ of radius $1 + \epsilon$ with $|C^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$. Here $\kappa^*$ denotes the maximal number of points of $S$ contained in a unit disk.

### 2.7.1 Simple Approximation Algorithm

The idea of the algorithm is first to get a rough estimate of $\kappa^*$ by putting a grid of width two over the point set and then reexamine the interesting regions. Let $k_{(i,j)} = |\{p \in S : 2i \leq p_x < 2(i+1) \text{ and } 2j \leq p_y < 2(j+1)\}|$ be the number of points contained in grid cell $(i, j)$ of the point set. Let $k = \max_{(i,j)} k_{(i,j)}$. The algorithm proceeds as follows:

1. Locate each point $p \in S$ in a grid of width 2 centered at the origin.
2. Let $U = \{(i, j) : \sum_{g=i-1}^{i+1} \sum_{h=j-1}^{j+1} k_{(g,h)} \geq k/4\}$ be the grid cells which have a 'well-occupied' neighborhood.
3. For each cell $(i, j) \in U$,
   (a) Place a grid of width $\epsilon$ over $(i, j)$.
   (b) Check each of the $O(1/\epsilon^2)$ grid points as center of a potential disc $C^{(1+\epsilon)}$ by counting the points of the neighborhood that would fall into that disc.
4. Report the best disc encountered during Step 3.

We will first argue about the correctness and show that it computes indeed a disc of radius $(1 + \epsilon)$ containing at least $\kappa^*$ points.

**Lemma 6.** *Assume point $(x^*, y^*)$ is the center of an optimal placement for the unit disc, then there is a grid point $(x', y')$ inspected during the algorithm such that $C_{(x^*,y^*)} \subset C^{1+\epsilon}_{(x',y')}$.*

*Proof.* Let us first show that the center of an optimal placement of the unit disc falls inside a cell $u \in U$. Assume otherwise, then there are less than $k/4$ points in the neighborhood of this optimal center that could be possibly covered. But using our grid approximation we can cover at least $k/4$ points since any grid cell can be covered by four unit discs and hence one of these discs must contain at least $k/4$ points, which is a contradiction.

So we know that $(x^*, y^*)$ falls in a cell $u \in U$. Let $(x', y')$ be the grid point inside $c$ which is closest to $(x^*, y^*)$. This has distance at most $\epsilon/\sqrt{2}$ hence the disc centered at $(x', y')$ with radius $1 + \epsilon$ contains $C_{(x^*,y^*)}$. □

Using the previous lemma we obtain the main result of this section:

**Theorem 9.** *Given a set of points $S$ in the plane and some $\epsilon > 0$, we can determine in $O(n/\epsilon^2)$ time a placement $(x, y)$ of a disc $C^{1+\epsilon}$ of radius $(1 + \epsilon)$ with $|C^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$, where $\kappa^*$ denotes the maximal number of points in $S$ contained in a unit disc.*

*Proof.* First observe that we have $k/4 \leq \kappa^* \leq 4k$, since any grid cell can be fully covered by four unit discs and any unit disc intersects at most four grid cells.

Locating and counting all points in their respective grid cells can be done in $O(n)$ expected time using a standard scheme for perfect hashing, which also allows to perform the second step within the same time bounds. Observe that $|U| = O(n/k)$, since any cell with at least $k/16$ points appears in at most 8 neighborhoods. Step 3 requires for each cell in $C$ the inspection of $O(1/\epsilon^2)$ potential centers. Each of these inspections takes $O(k)$ time by brute-force, yielding a running time of $O(n/\epsilon^2)$ for Step 3 which dominates the overall running time. □

### 2.7.2 A Variant for Large $\kappa^*$

Our algorithm is clearly optimal in terms of the dependence on $n$, still in some settings, in particular for $\kappa^* = \omega(1/\epsilon)$, one can achieve a better dependence on $\epsilon$. In the following we sketch an improvement for this case. The basic idea of the approach is to replace the brute-force neighborhood exploration for each grid-point by a query to a suitable data structure for approximate weighted range counting [AM00].

For each cell $u = (i, j) \in U$, we put a grid of width $\epsilon$ not only covering $(i, j)$ but also its neighbors $\{(i', j') : i-1 \leq i' \leq i+1, j-1 \leq j' \leq j+1\}$. For each of the resulting $O(1/\epsilon^2)$ mini-cells we count the number of points contained and associate it with a representative which is located at the center of each mini-cell and has weight according to the number of points in the cell. For these representatives we construct a data structure for $\epsilon$-approximate weighted range counting in $O((1/\epsilon^2)\log(1/\epsilon^2))$ time. Now each grid point contained in cell $c$, instead of using the $O(k)$ brute-force exploration of its neighborhood like before, queries this data structure with a $(1+3\epsilon)$ query, which takes $O(\log(1/\epsilon^2)+1/\epsilon)$ time. The weight returned corresponds to a set of points that can surely be enclosed in a disc of radius $(1+5\epsilon)$. Furthermore all points within distance $(1 + \epsilon)$ are guaranteed to be accounted for. So correctness follows from the same arguments as in the previous algorithm and we obtain the following result which is an improvement over our previous algorithm for $\kappa^* = \omega(1/\epsilon)$.

**Theorem 10.** *Given a set of points $S$ in the plane and some $\epsilon > 0$, we can determine in time $O(n+(n/\kappa^*)\cdot(1/\epsilon^3))$ a placement $(x, y)$ of a disc $C^{1+\epsilon}$ of radius $(1+\epsilon)$ with $|C^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$, where $\kappa^*$ denotes the maximal number of points in $S$ contained in a unit disc.*

## 2.8 Concluding remarks

We have given a unified approach to solve the problem of computing a translate of a closed and bounded set $C$ such that it contains a largest subset of a given set of $n$ points. We started with two basic algorithms. Then we presented two transformations, based on random sampling and bucketing, respectively. By combining these transformations with any of the basic algorithms,

we obtained different algorithms, either solving the exact optimal placement problem, or the approximation version of this problem. In the latter case, the algorithm is of the Monte Carlo type, with a guaranteed upper bound on the error probability. We also presented a deterministic algorithm for the approximation problem using an adaptation of Chazelle's cutting algorithm.

We leave open the problem of improving the error probabilities of our Monte Carlo approximation algorithms for small values of $k^*$. Dickerson and Scharstein [DS98] have considered the optimal placement problem if, besides translations, it is also allowed to rotate the set $C$. For the case when $C$ is a convex polygon with $m$ vertices, they give an algorithm for computing an optimal placement in $O(n^2k^*m^2\log(mn))$ time. In the worst case, this at least cubic in $n$. It would be interesting to know if our techniques can be extended to obtain a faster approximation algorithm.

## Chapter 3

# Planarity in a Terrain

## 3.1 Introduction

A *terrain* is a 2-dimensional surface in 3-dimensional space with a special property: every vertical line intersects in a point, if it intersects it all. Mathematically, it is a surface in $\mathbb{R}^3$ defined by a function $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. If $f$ is piecewise linear and the surface consists of a collection of triangles, the terrain is called a *triangulated irregular network (TIN)*.

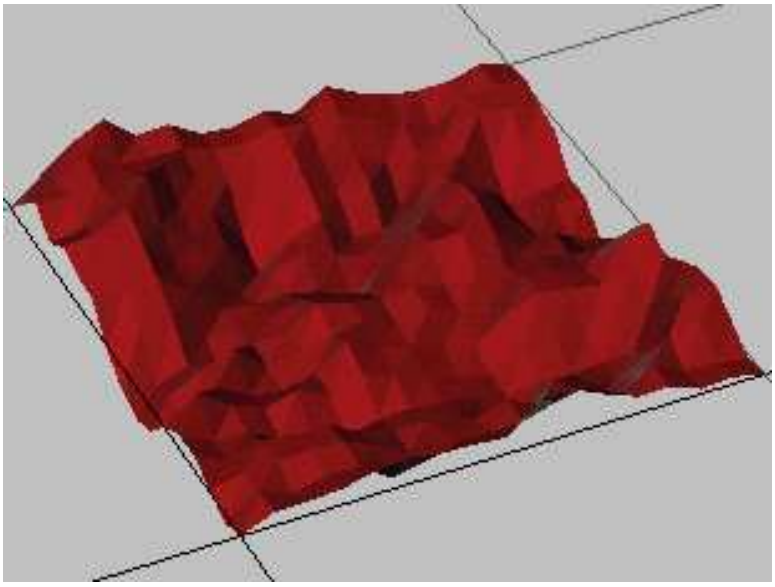

*Fig. 3.1: A* Triangulated Irregular Network*(TIN).*

Given a triangulated irregular network $\mathcal{T}$, our goal is to find large, almost planar regions in $\mathcal{T}$. More formally, we want to find a subset of triangles $T$ of $\mathcal{T}$ and a vector $\overrightarrow{r}$ (called the *reference normal*), such that

1. the adjacency graph of the triangles in $T$ is connected,
2. for each triangle $t \in T$, the angle between $\overrightarrow{r}$ and $\overrightarrow{n_t}$ is at most $\delta$, where $\overrightarrow{n_t}$ denotes the normal of triangle $t$ and $\delta$ is a given parameter, and
3. $T$ is chosen such that the total weight of $T$ is maximized, where the weight can be for example the number or the total area of the triangles in $T$ (depending on the application).

If the set of triangles $T$ satisfies (1) and (2) above, we say that $T$ is *$\delta$-planar* with respect to the reference normal $\overrightarrow{r}$ . We now show how this notion can be used to reformulate our problem.

The *unit sphere*, i.e., the boundary of the three-dimensional ball centered at the origin and having radius one, is denoted by $\mathbb{S}^2$. The *upper hemisphere* is defined as $\mathbb{S}^2_+ := \mathbb{S}^2 \cap \{(x, y, z) \in \mathbb{R}^3 : z > 0\}$. We can regard the normal vector of any triangle in a terrain as a point on $\mathbb{S}^2_+$. The lower hemisphere $\mathbb{S}^2_-$ is of no interest to us because, due to the imaging principle, each vertical line can intersect the surface described by the image in only one point. Hence, no normal can have a negative $z$-coordinate.

Let $\mathcal{T}$ be a TIN, and let $S \subseteq \mathbb{S}^2_+$ be the set of normal vectors of the triangles in $\mathcal{T}$. Let $G = (S, E)$ be the undirected graph having vertex set $S$ and in which any two vertices are connected by an edge if and only if the corresponding triangles in $\mathcal{T}$ share an edge. (Actually, $S$ is a multiset because different triangles in $\mathcal{T}$ may have the same normal. Equal normals are treated as different vertices in $G$.) Observe that each vertex of $G$ has degree at most three. We give each vertex $p$ of $G$ a weight $wt(p)$ which is equal to the area of the triangle that gives rise to $p$. The weight $wt(C)$ of any subset $C$ of $S$ is defined as $wt(C) := \sum_{p \in C} wt(p)$.

For any point $x \in \mathbb{S}^2_+$, let $D_x$ denote the spherical disk of radius $\delta$ centered at $x$. That is, $D_x$ is the set of all points $y \in \mathbb{S}^2$ such that the angle between the vectors $\mathbf{x}$ and $\mathbf{y}$ is less than or equal to $\delta$. Furthermore, let $G_x$ denote the subgraph of $G$ having $S \cap D_x$ as its vertex set and whose edge set is the set of all edges $(p, q) \in E$ for which $p$ and $q$ are both contained in $D_x$. Finally, we define $W_x$ to be the maximum weight of any connected component of the graph $G_x$. Using this terminology, our problem can be formally stated as follows.

**Problem 1. Given a graph $G$ as above having $n$ vertices, and a real constant $\delta > 0$, compute a point $x \in \mathbb{S}^2_+$ such that $W_x$ is maximum.**

Let $(p, q)$ be any edge of the graph $G$. If the angle between the vectors $\mathbf{p}$ and $\mathbf{q}$ is larger than $2\delta$, then it is clear that $(p, q)$ can be ignored when solving Problem 1. Therefore, we may assume without loss of generality that the angle between the endpoints (when regarded as vectors) of any edge of $G$ is at most $2\delta$.

### 3.1.1 Main results

In Section 3.2, we will show how computational geometry and dynamic graph algorithms can be used to solve Problem 1 in $O(n^2 \log n (\log \log n)^3)$ time.

It is unlikely that Problem 1 can be solved in subquadratic time. In fact, it seems that even the problem of computing a point $y \in \mathbb{S}^2_+$ such that $W_y$ approximates the optimal solution cannot be solved in subquadratic time. Therefore, in Section 3.4, we describe a simple grid-based heuristic. We have implemented this heuristic and discuss some details about it in Section 3.5. We also present some experimental results on images of fracture surfaces obtained by confocal laser scanning microscopy. These results show that the heuristic is able to find $\delta$-planar regions whose area is sufficiently large.

### 3.1.2 Related work

The problem considered here is related to the *terrain simplification problem*. In this problem, we want to approximate a polyhedral terrain by a "smaller" terrain, i.e., one having the minimum number of vertices. Although this problem has been studied in, for example, the computer graphics community [SB95], the main reference we are aware of that considers this problem from a complexity point of view is Agarwal and Suri [AS98]. They give evidence that the terrain simplification problem is hard by proving that a strongly related problem is NP-hard. They also give a polynomial-time algorithm for approximating the minimum terrain.

The problem 3SUM is defined as follows: Given three sets $A$, $B$, and $C$, each consisting of $n$ real numbers, decide whether there are elements $a \in A$, $b \in B$, and $c \in C$ such that $a + b = c$. This problem can be solved in $O(n^2)$ time, and it is widely believed that it cannot be solved in subquadratic time. A problem $\mathcal{P}$ is called 3SUM-*hard* if 3SUM can be reduced to $\mathcal{P}$ in subquadratic time; see Gajentaan and Overmars [GO95].

If we consider Problem 1 for the case when the graph $G$ is complete and all vertices have equal weights, then we get the problem of computing a placement of a spherical disk that contains the largest subset of a given set $S$ of $n$ points on $\mathbb{S}^2$. This problem has been considered by Chazelle and Lee [CL86] for the case when $S$ is a set of points in the Euclidean plane. They showed that the problem can be solved in $O(n^2)$ time. The related problem of computing the deepest point in an arrangement of halfplanes is 3SUM-hard, see [GO95]. This indicates that it is unlikely that the disk placement problem can be solved in subquadratic time. In the previous chapter and in [AHR$^+$02] & [FMR04] we gave an an alternative $O(n^2)$-time algorithm for the optimal disk placement problem, as well as randomized approximation algorithms whose running times are close to linear. Our algorithm in Section 3.2 for solving Problem 1 has been inspired by the $O(n^2)$-time algorithm developed by us to solve the *placement problem*.

## 3.2 Solving Problem 1

In this section, we give an algorithm that solves Problem 1. Consider the graph $G = (S, E)$, and consider the spherical disks $D_p$ centered at the points $p$ of $S$. Let $\mathcal{A}$ be the arrangement on $\mathbb{S}^2$ defined by the boundaries of the disks $D_p$, where $p \in S$. That is, $\mathcal{A}$ is the subdivision of $\mathbb{S}^2$ into vertices, edges and faces defined by the overlay of the boundaries of the disks $D_p$, $p \in S$. Since $p \in D_x$ if and only if $x \in D_p$, we have $W_x = W_y$ for any two points $x$ and $y$ that are in the interior of the same face $f$ of $\mathcal{A}$. Also, for each vertex $z$ of $f$, we have $W_z \geq W_x$. This proves the following lemma.

**Lemma 7.** *To solve Problem 1, it suffices to consider points $x \in \mathbb{S}^2_+$ that are vertices of the arrangement $\mathcal{A}$.*

Throughout this section, we make the following *general-position* assumption about the set $S$. We assume that the elements of $S$ are pairwise distinct. Moreover, we assume that for any two distinct points $p$ and $q$ of $S$, the spherical disks $D_p$ and $D_q$ are either disjoint or have an intersection of positive area (hence, $D_p$ and $D_q$ do not touch each other). Finally, for any three distinct points $p$, $q$, and $r$ of $S$ the spherical disks $D_p$, $D_q$, and $D_r$ do not intersect in a

single point. We make this assumption only to simplify the description of our algorithm. This algorithm can easily be extended to handle arbitrary sets of points.

The discussion above leads to the following preliminary algorithm for solving Problem 1.

**Step 1:** Compute the arrangement $\mathcal{A}$.

**Step 2:** Let $W := 0$. For each vertex $x$ of $\mathcal{A}$, do the following.

- Compute the graph $G_x$.
- Compute the connected components of $G_x$, together with their weights. Let $W_x$ be the maximum weight of any of these connected components.
- Set $W := \max(W, W_x)$.

**Step 3:** Return $W$.

It is clear that this algorithm correctly solves Problem 1. Let us analyze its running time. Recall that $n$ denotes the number of elements of the point set $S$. For any $p \in S$, let $deg(p)$ denote the degree of $p$ in $G$. Observe that $\sum_{p \in S} deg(p)$ is equal to twice the number of edges of $G$, and that $deg(p) \leq 3$ for any $p \in S$. Therefore, $G$ has at most $3n/2$ edges.

Step 1, i.e., computing the arrangement $\mathcal{A}$, takes $O(n^2)$ time using, e.g., the algorithm of Amato *et al.* [AGR00]. Consider any vertex $x$ of $\mathcal{A}$. The graph $G_x$ can clearly be computed in time proportional to the number of vertices and edges of $G$; hence, $G_x$ can be computed in $O(n)$ time. Given $G_x$, its connected components and the value $W_x$ can be computed in $O(n)$ time; see, e.g., the book by Cormen *et al.* [CLR90]. Hence, for each vertex $x$ of $\mathcal{A}$, $O(n)$ time is spent in Step 2. Since $\mathcal{A}$ has $O(n^2)$ vertices, the overall time for Step 2 is $O(n^3)$. Hence, the entire algorithm takes $O(n^3)$ time.

## 3.3 Improving Running Time

We now show how to improve the running time considerably. Note that the bottleneck of the previous algorithm is Step 2. The idea of the improved algorithm is to traverse the arrangement $\mathcal{A}$ and maintain the connected components of $G_x$ in a data structure. Consider what happens when we walk along an edge of $\mathcal{A}$ from a vertex $x$ to a vertex $y$. Assume that we know the connected components of the graph $G_x$. Our goal is to compute the connected components of $G_y$ as fast as possible. Observe that walking from $x$ to $y$ means that we move the spherical disk $D_x$ along an edge of $\mathcal{A}$ to the position $D_y$. During this move, at most one point of $S$ can enter or leave the spherical disk. (Here we use our general-position assumption.) Since the graph $G$ has degree three, it follows that the graph $G_y$ can be obtained from $G_x$ by performing at most a constant number of edge insertions and deletions.

Assume that we have a data structure $CC$ that stores the connected components, together with their weights, of a given graph, and that supports edge insertions and deletions, and queries of the form "report the maximum weight of any connected component". (We will specify this data structure later. For the moment, we use it as a black box.) For any point $x \in \mathbb{S}^2_+$, we denote by $CC_x$ the instance of this data structure for the graph $G_x$.

Our improved algorithm does the following.

**Step 1:** Compute the arrangement $\mathcal{A}$.

**Step 2:** Let $x$ be an arbitrary vertex of $\mathcal{A}$.

- Compute the graph $G_x$.
- Compute the connected components of $G_x$, together with their weights. Compute $W_x$ as the maximum weight of any connected component of $G_x$.
- Set $W := W_x$.
- Construct the data structure $CC_x$.

**Step 3:** Starting at $x$, traverse the vertices of the arrangement $\mathcal{A}$, e.g., in depth-first order. In a generic step, we walk from a vertex $y$, along an edge of $\mathcal{A}$, to a neighboring vertex $z$. At the moment when we leave $y$, we have the data structure $CC_y$, storing the connected components of the graph $G_y$, together with their weights. The graph $G_z$ can be obtained by inserting and deleting at most a constant number of edges in the current graph $G_y$. Hence, we obtain the data structure $CC_z$ by performing these updates in the data structure $CC_y$. Afterwards, we query $CC_z$ to find the value of $W_z$, and set $W := \max(W, W_z)$.

**Step 4:** Return $W$.

The correctness of this algorithm is clear. Steps 1 and 4 take $O(n^2)$ and $O(1)$ time, respectively. The times for Steps 2 and 3 depend on the data structure $CC$. Let $P(n)$, $U(n)$, and $Q(n)$ denote the preprocessing time, update time, and query time of this data structure, respectively. Then Step 2 takes $O(n + P(n))$ time. In Step 3, we spend $O(U(n) + Q(n))$ time for each edge of $\mathcal{A}$. Since this arrangement has $O(n^2)$ edges, it follows that the total running time of the algorithm is

$$O\left(P(n) + n^2(U(n) + Q(n))\right).$$

It remains to specify the data structure $CC$. In [Tho00], Thorup gives a data structure for maintaining a spanning forest of a graph under insertions and deletions of edges, in $O(\log n(\log\log n)^3)$ amortized time per update, where $n$ denotes the number of vertices of the graph. Given any two vertices of this graph, it can be decided in $O(\log n/\log\log\log n)$ time if they are in the same connected component. (A simpler but theoretically slightly less efficient data structure was given by Holm *et al.* [HdLT98].) This data structure can easily be extended so that it maintains the weights of all connected components within the same time bound. If we store these weights in a heap, then we can extract the weight of the largest connected component in $O(1)$ time. Moreover, this heap can be updated in $O(\log n)$ time per operation. The data structure can be built by successively inserting all edges into an initially empty graph. Hence, we have $P(n) = O(n\log n(\log\log n)^3)$, $U(n) = O(\log n(\log\log n)^3)$, and $Q(n) = O(1)$. Thus, we have proved the following result.

**Theorem 11.** *Problem 1 can be solved in $O(n^2\log n(\log\log n)^3)$ time.*

It is not clear if Problem 1 can be solved in subquadratic time; see Section 3.1.2. Instead, one can ask about the time complexity for approximating the optimal solution. That is, let $\epsilon$ be a real number such that $0 < \epsilon < 1$, and let $x \in \mathbb{S}^2_+$ be a point for which $W_x$ is maximum.

In the approximation version of Problem 1, we have to compute a point $y \in \mathbb{S}^2_+$ such that $W_y \geq (1-\epsilon)W_x$.

Consider the following example. Let $D$ be a spherical disk of radius $\delta$, and let $p$ and $q$ be two diametrally opposite points on the boundary of $D$. Let $m$ be a large integer. For each $i$, $1 \leq i \leq m$, let $a_i := p$ and $b_i := q$. Consider the edges $(a_i, b_i)$, $1 \leq i \leq m$, and $(b_i, a_{i+1})$, $1 \leq i < m$. Note that these edges form a path between $a_1$ and $b_m$ that alternates between the points $p$ and $q$. Let $G$ be a graph containing the points $a_i$ and $b_i$, $1 \leq i \leq m$, as vertices, and the above edges. All other vertices of this graph are "far" away from $p$ and $q$, and have "large" distances among each other. We assume that all vertices of $G$ have unit-weight. (It is easy to construct a TIN for which $G$ is the corresponding graph.) For this graph, the center of $D$ gives the optimal solution to Problem 1. If we move the disk $D$, then either the point $p$ or the point $q$ leaves the disk and, hence, each connected subgraph of $G$ that is contained in the disk consists of one single vertex. This shows that any approximation algorithm for Problem 1 must return the center of $D$. Because of this, the approximation version of Problem 1 has the same time complexity as Problem 1 itself.

## 3.4 A Heuristic for Finding Large $\delta$-planar Regions

In this section, we give a simple heuristic approach to compute large connected regions in a TIN that are $\delta$-planar. (Different regions need not be approximately contained in the same two-dimensional plane.) We cannot prove any non-trivial bounds on the quality of its output, but experiments have shown that the output is good in practice, and that the heuristic is fast.

Let $\mathcal{T}$ be a TIN consisting of $n$ triangles, let $\delta > 0$, let $S$ be the set of normal vectors of these triangles, and let $G = (S, E)$ be the graph as defined in Section 3.1. Recall that the weight $wt(p)$ of any element $p$ of $S$ is equal to the area of the triangle whose normal vector is $p$. Also, recall that $S$ is actually a multiset. Our goal is to find all $\delta$-planar regions in $\mathcal{T}$ having area at least $A$, where $A$ is some given positive real number.

We define a grid on the upper hemisphere $\mathbb{S}^2_+$ using lines of longitude and latitude such that every grid cell is contained in some spherical disk of radius $\delta$. To be more precise, we choose an appropriate real number $\epsilon$ with $\epsilon = \Theta(\delta)$ and use $X := \lceil \pi/\epsilon \rceil$ equally spaced lines of longitude $lo_i$, $0 \leq i < X$, and $Y := \lceil \pi/(2\epsilon) \rceil$ equally spaced lines of latitude $la_j$, $0 \leq j < Y$.

For any two indices $i$ and $j$ with $0 \leq i < X$ and $0 \leq j < Y$, we call the pair $(i, j)$ the *index* of the grid cell bounded by $lo_i$, $lo_{i+1}$, $la_j$, and $la_{j+1}$. For any point $p \in \mathbb{S}^2_+$, we can in $O(1)$ time compute the index of the grid cell containing $p$. Note that each grid cell is adjacent to at most four other cells, except for those that are incident to the north pole.

Our heuristic takes as input the TIN $\mathcal{T}$, the positive real numbers $\delta$ and $A$, and the graph $G = (S, E)$. It starts by computing a subset of all $\delta$-planar regions in $\mathcal{T}$ having area at least $A/4$. (Below, it will become clear why we choose $A/4$ instead of $A$. In fact, the factor $1/4$ can be replaced by any constant between zero and one.) Then it makes a boundary correction step, in which each region is enlarged, while remaining $\delta$-planar. To be more precise, the heuristic makes the following seven steps.

**Step 1:** Initialize an array $C[0..X-1, 0..Y-1]$, and store with each entry an empty list of points and an empty list of edges.

**Step 2:** For each point $p \in S$, compute the index $(i_p, j_p)$ of the grid cell that contains $p$, and add $p$ to the point list of $C[i_p, j_p]$.

**Step 3:** For each edge $(p, q) \in E$ with $(i_p, j_p) = (i_q, j_q)$, add $(p, q)$ to the edge list of $C[i_p, j_p]$. (Hence, in this step, all edges that are completely contained within one grid cell are extracted.)

**Step 4:** Initialize an empty list $L$.

**Step 5:** For each $i$ and $j$ with $0 \leq i < X$ and $0 \leq j < Y$, do the following.

- Let $G_{ij}$ be the graph having the points of $C[i, j]$ as its vertices, and the edges of $C[i, j]$ as its edges. Compute the connected components of $G_{ij}$, together with their weights.
- For each connected component of $G_{ij}$, add it to the list $L$ if it has weight at least $A/4$.

Consider the list $L$ when Step 5 has been completed. Each element of $L$ corresponds to a connected subgraph of $G$ having weight at least $A/4$ and that is contained in one grid cell, i.e., it corresponds to an $\delta$-planar region in the TIN having area at least $A/4$. It may happen that such a region $R$ can be enlarged by adding triangles that are adjacent to $R$ and whose normals are in a grid cell that is adjacent to the grid cell that gave rise to $R$. Of course, the enlarged region should still be $\delta$-planar. Below, we describe a "boundary correction" step that does exactly this.

**Step 6 (Boundary correction step):** This step is performed for each connected subgraph of $G$ that is stored in the list $L$. Let $G'$ be any such subgraph.

We first compute the smallest enclosing spherical disk $D'$ containing all vertices of $G'$, using the linear-time algorithm as presented, e.g., in de Berg *et al.* [dBvKOS97]. Let $c$ be the center of $D'$, and let $D$ be the spherical disk of radius $\delta$ centered at $c$. Note that $G'$ is contained in $D$, because $D'$ is contained in $D$.

We now go back to the TIN $\mathcal{T}$ and mark the triangles corresponding to the vertices of $G'$. Let $T$ be the set of all marked triangles. Note that all normals of triangles in $T$ are contained in $D$. Now we consider each unmarked triangle $t$ of $\mathcal{T}$ that shares an edge with at least one marked triangle, and mark $t$ if its normal is contained in $D$. Let $T'$ be the set of all marked triangles after we have considered all such triangles $t$. If $T = T'$, then we stop the boundary correction step for this subgraph $G'$. Otherwise, we repeat this process by considering unmarked triangles that share an edge with at least one marked triangle.

**Step 7:** After having completed the boundary correction step for each element in $L$, we have a collection of $\delta$-planar regions, each one having area at least $A/4$. Since these regions may overlap, we continue as follows. We sort all these enlarged regions according to their areas. If the largest region has area at least $A$, then we report it, and mark all its triangles in $\mathcal{T}$. Then we consider the second largest region. If its area is at least $A$, and if none of its triangles has been marked yet, we also report it and mark all its triangles in $\mathcal{T}$. We continue this until we have reported all regions whose area is at least $A$ and that do not overlap any of the previously reported regions.

What can we say about the quality of the output? Let $R$ be any $\delta$-planar region in the TIN $\mathcal{T}$, and let $G'$ be the corresponding connected subgraph of $G$. Our algorithm reports the region $R$ if and only if $G'$ contains a connected subgraph $G''$ of weight at least $A/4$ that is completely

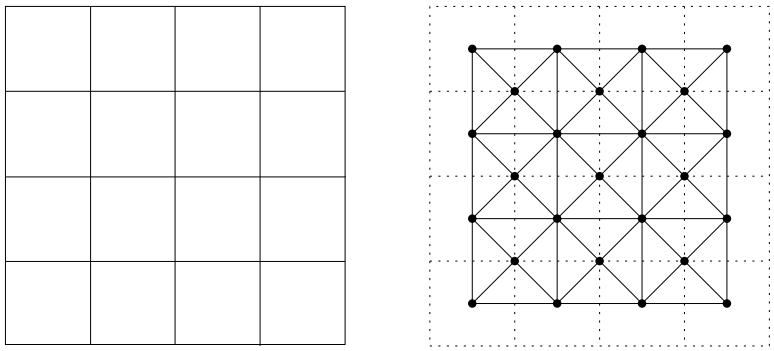

*Fig. 3.2: Converting an array of pixels to a TIN.*

contained in one of our grid cells. If each edge of $G'$ crosses a cell boundary, then $R$ will not be reported. Therefore, in order to improve the chances of finding $R$, we run the algorithm several times using shifted copies of the grid.

## 3.5 Implementation and Experimental Results

We have implemented the heuristic that was discussed in the previous section. The program, which we call $PlaneFinder$, takes as input a $k \times \ell$ array of pixels in `tif` (tagged image file) format. The value stored at each entry is the height of the corresponding surface point, i.e., the $z$-value of the voxel position.

The application is open to all kind of imaging methods which deliver topographical images. Besides confocal laser microscopy that we used to obtain our images, examples include scanning force microscopy [Sch97], white light interference microscopy [HSW00], extended focus conventional light microscopy [YWM99], and photogrammetry based on stereo pairs received by scanning electron microscopy [Boy73].

$PlaneFinder$ starts by computing a TIN as illustrated in Figure 3.2. The vertices of the TIN are (i) the centers of the pixels, where the $z$-coordinate is given by the height of the pixel, and (ii) the centers of all $2 \times 2$ blocks of pixels, where the $z$-coordinate is given by the average height of the four pixels comprising the block. These vertices are joined by edges as indicated in the right part of Figure 3.2. Note that the total number of triangles, which we denote by $n$, is equal to $n = 4(k-1)(\ell-1)$. Given this TIN, $PlaneFinder$ proceeds as described in Section 3.4.

The program is written in C++ and uses the LEDA library [MN99]. In the current version, the following operations are supported.

- Show the $k$ largest $\delta$-planar regions (as found by the heuristic), for some values $k$ and $\delta$ provided by the user.
- Show all $\delta$-planar regions having area at least $A$ (again as found by the heuristic), for some values $\delta$ and $A$ provided by the user.

The output is the original `tif`-image in which all regions that have been detected are colored. (A pixel in the image is colored if and only if at least one of the triangles overlapping the pixel

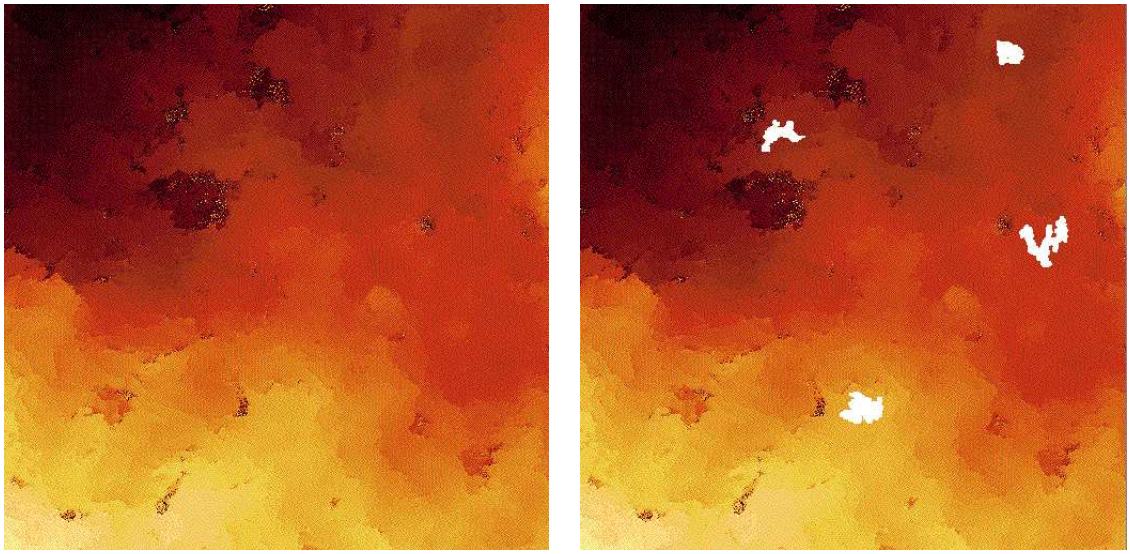

*Fig. 3.3: On the left, a confocal laser scanning micrography (topographical image) of a steel fracture surface is displayed. The colors represent the heights of the pixels. The four $\delta$-planar regions computed by $PlaneFinder$ are marked white in the right image. The image size is 100$\mu$m$\times$100$\mu$m.*

belongs to a region.) The program takes care that adjacent regions are drawn in visibly distinct colors. The program also lists the regions found in increasing order of their area.

We have run $PlaneFinder$ on various images of fracture surfaces obtained by confocal laser scanning microscopy. As mentioned already, these images are in `tif` format, and consist of $512 \times 512$ pixels. Hence, the corresponding TINs consist of $n = 1,044,484$ triangles. Most of the surfaces are very rough and contain only small $\delta$-planar regions. For a typical image $PlaneFinder$ takes about 35 seconds.

An example is given in Figure 3.3, which shows the original image (on the left) as well as the output of $PlaneFinder$ (on the right). The program was run with $\delta$ equal to five degrees and $A$ equal to 1500 square units. The right part shows, in white, the four $\delta$-planar regions found having area at least $A = 1500$ square units. (Note that area refers to the true area in space and not the projected area.)

Figure 3.4 shows an example of an image in which we have added four large slanted triangles. One of these is $\delta$-planar for $\delta = 5$ degrees, whereas the others are $\delta'$-planar for a value of $\delta'$ that is slightly larger than five degrees. $PlaneFinder$ was run on this image with $A = 3000$ square units. It correctly detected only one of the triangles.

In order to flatten local roughness in the images, we have included the option to apply a mean-filter in a preprocessing step. When we apply this filter to the image of Figure 3.4, then $PlaneFinder$ finds all four triangles.

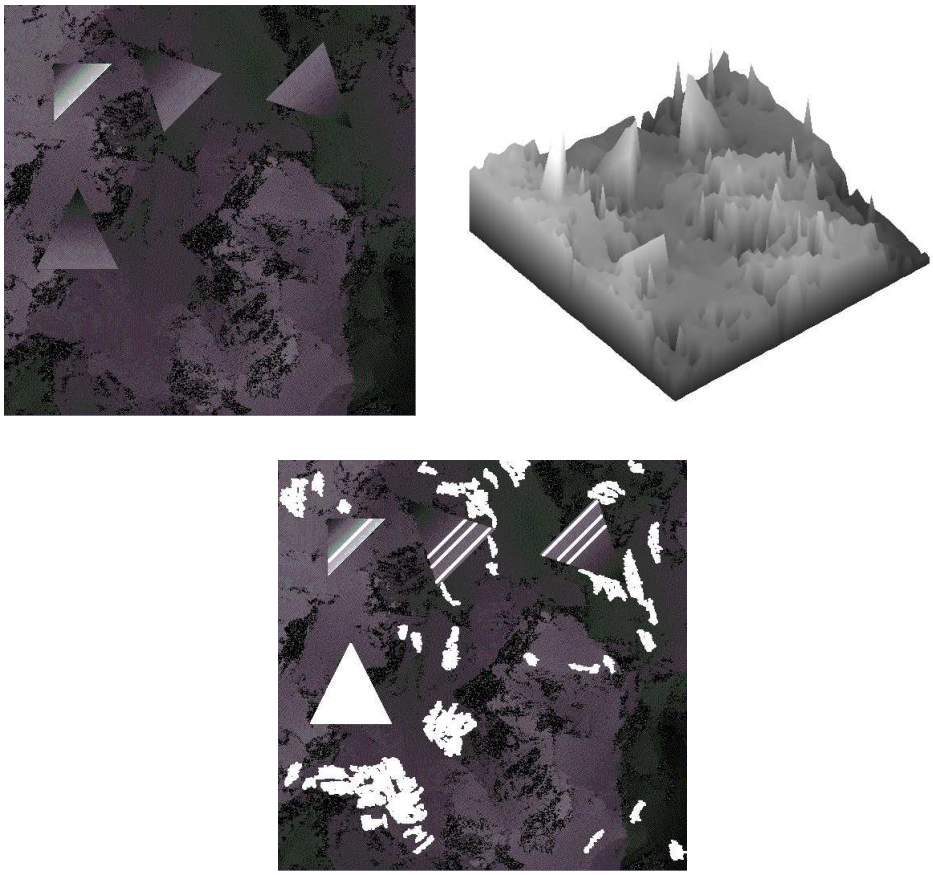

*Fig. 3.4: On the top, a topographical image and its parallel projection onto the $xy$-plane are displayed. One "approximately planar" triangle and three "not so approximately planar" triangles have been inserted for testing purposes. The $\delta$-planar regions computed by $PlaneFinder$ are marked white in the bottom image.*

## 3.6 Concluding Remarks

We have considered the problem of computing large regions in a terrain that are connected and approximately planar. We showed that the problem of computing the largest such region can be solved in a time that is roughly quadratic in the number of triangles in the terrain, and argued that it is unlikely to solve the problem faster. We leave open the problem of proving this rigorously. We also argued that it may even be hard to approximate this largest region. Proving this claim formally is also left as an open problem.

We defined a connected set of triangles to be "approximately planar" if their normals are contained in a spherical disk of a fixed small radius. It would be interesting to solve the problem for other notions of being approximately planar. For example, we could require the angular diameter of the normals to be at most $\delta$, or the set of triangles to be contained between two parallel planes having distance $\delta$.

# Chapter 4

# Locating Planar Regions in a Terrain with a Guarantee

## 4.1 Introduction

We have observed that the heuristic approach in the previous chapter though performs well on the real-world data sets, we were unable to prove any non-trivial bounds on the quality of its output. And also the exact algorithm to compute the largest planar region in terrain was too expensive to be praticable at all. In order to overcome the shortcoming in the heuristic approach, here we present an algorithm for an approximated version of the same problem, which, given some parameters $\delta$ and $\epsilon$ produces a connected subterrain and a reference normal such that all triangle normals in the subterrain deviate at most $(1+\epsilon)\cdot\delta$ from the reference normal, and the weight of the subterrain is at least the weight of the optimal subterrain with maximum deviation $\delta$. The running time of this algorithm is $O(n/\epsilon^2)$. We sketch also a variant of this algorithm with a better dependence on $\epsilon$ but an extra polylogarithmic factor on $n$. For $n$ sufficiently large, both algorithms use optimal $O(n)$ space.

We also elaborate experimental results obtained by the implementation of the approximate algorithm which runs in reasonable time on real-world test data consisting of terrains with several hundred thousands triangles.

## 4.2 Finding a Large Planar Region Approximately

### 4.2.1 Preliminaries

Most of the notations and symbols used in this chapter are the same as in the previous one. We still repeat some of the basic definitions and facts about *terrain* which we shall be using later to develop our approxinate algorithms in order to enhance readability of this chapter.

Let $\mathcal{T}$ be a *triangulated irregular network* (TIN). We associate with $\mathcal{T}$ an undirected weighted graph $G_{\mathcal{T}}(V,E)$ as follows. Each triangle $t$ in $\mathcal{T}$ has an associated weight $w(t)$ and corresponds to a vertex $v_t$ in $V$. An edge connects two vertices of $V$ if and only if the corresponding triangles in $\mathcal{T}$ are adjacent. Note that $G_{\mathcal{T}}$ is the dual graph of $\mathcal{T}$, is planar and has degree three.

Each vertex $v_t \in V$ is assigned the weight of its associated triangle $w(t)$ which can be, for instance, equal to the area of the triangle $t$ (when the objective is to maximize the *area* of the detected region) or simply one (if we want to maximize the *number* of triangles in the region). The weight $w(V')$ of any subset $V'$ of $V$ is defined as the sum of the weights of the vertices in $V'$.

Let $\delta > 0$ and $\epsilon > 0$ be two real parameters taking reasonably small values for our problem, for example, they satisfy $\delta\epsilon \leq 1$. Throughout, we denote the normal of a triangle $t$ by $\overrightarrow{n_t}$. We use the notation $\angle(v, u)$ to denote the angle between two vectors $\overrightarrow{v}$ and $\overrightarrow{u}$. For a point $u \in \mathbb{R}^3$ we denote by $\overrightarrow{u}$ the vector $\overrightarrow{Ou}$, where $O$ is the origin.

We present now some basic definitions. We say that a subset of triangles $T$ of $\mathcal{T}$ is *$\delta$-planar* if (i) the triangles in $T$ are connected and (ii) there is a vector $\overrightarrow{r}$ such that for each $t \in T$, $\angle(r, n_t) \leq \delta$. A subset of triangles $T$ of $\mathcal{T}$ is *optimal* $\delta$-planar if it has the largest possible weight over all $\delta$-planar subsets of $\mathcal{T}$.

***Our Notion of Approximation***

There are at least two ways to define the notion of an $\epsilon$-approximate $\delta$-planar set $T$. One way would be to require $T$ to be $\delta$-planar and of weight at least $(1 - \epsilon)$ times the weight of an optimal $\delta$-planar set. Unfortunately solving this type of approximation seems to be as difficult as solving the problem exactly, see [SRWL04] for more details. We adopt the following notion of approximation: **A subset of triangles $T$ of $\mathcal{T}$ is *$\epsilon$-approximate* $\delta$-planar if it is $\delta(1 + \epsilon)$-planar and has weight at least as large as an optimal $\delta$-planar set.**

### 4.2.2 $\delta\epsilon$-Discretization

Let $\mathbb{S}^2$ denote the unit sphere, i.e., the boundary of the three-dimensional ball of radius one centered at the origin. As it will be clear next, we only need consider the *upper hemisphere* of $\mathbb{S}^2$ but for simplicity we use the whole sphere $\mathbb{S}^2$.

For each triangle $t \in \mathcal{T}$, we can associate a point $v_t \in \mathbb{S}^2$ that represents the normalized normal of triangle $t$. Specifically, $\overrightarrow{v_t} = \overrightarrow{n_t}/|\overrightarrow{n_t}|$. Our goal is to approximate the space $\mathbb{S}^2$ of all normals by a finite set of points $\mathbb{V} \subseteq \mathbb{S}^2$ such that for any $s \in \mathbb{S}^2$, there is a point $p \in \mathbb{V}$ nearby.

**Definition 2.** *A set of points $\mathbb{V} \subseteq \mathbb{S}^2$ is called a* $\delta\epsilon$-discretization *of $\mathbb{S}^2$ if $\forall s \in \mathbb{S}^2 : \exists p \in \mathbb{V}$ with $\angle(s, p) \leq \delta \cdot \epsilon$.*

**Lemma 8.** *There exists a $\delta\epsilon$-discretization of $\mathbb{S}^2$ of size $O(1/(\delta\epsilon)^2)$ which can be computed in the same time.*

*Proof.* The following construction yields a $\delta\epsilon$-discretization for $\mathbb{S}^2$. Consider a cube $L$ with side-length 2 centered at the origin. Note that $\mathbb{S}^2 \subset L$. Place a 2-dimensional grid of size $k \times k$ with $k = \lceil\sqrt{2}/(\delta\epsilon)\rceil$ over each of the six facets of $L$. This generates $k^2$ equally sized square grid cells on each face of $L$, where each cell has side-length at most $(\delta\epsilon\sqrt{2})$, and $6k^2 + 2$ grid points overall. See Figure 4.1. Let $Q$ be the set consisting of these grid points. Our $\delta\epsilon$-discretization $\mathbb{V}$ of $\mathbb{S}^2$ is defined as

$$\mathbb{V} = \Big\{ \frac{\overrightarrow{q}}{|\overrightarrow{q}|} : q \in Q \Big\},$$

that is, for each grid-point $q$ we shoot a ray from the origin through $q$ and include the point where the ray leaves $\mathbb{S}^2$ into our set $\mathbb{V}$. It remains to prove that $\mathbb{V}$ is indeed a $\delta\epsilon$-discretization.

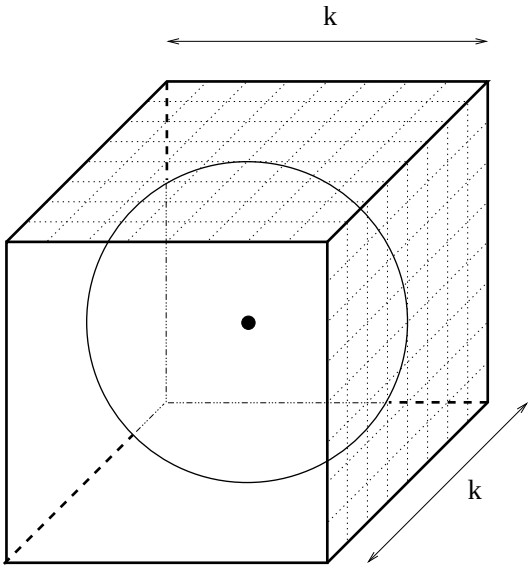


*Fig. 4.1:* Cube with sidelength two containing $\mathbb{S}^2$ and with a $k \times k$ grid on each of its faces.

Consider any point $s$ on $\mathbb{S}^2$ and the point $\widetilde{s}$ where the ray starting at the origin and passing through $s$ hits the boundary of the cube $L$. Since $k = \lceil \sqrt{2}/(\delta\epsilon) \rceil$, there is a grid point $q \in Q$ that has distance to $\widetilde{s}$ at most $d = (\sqrt{2}/2) \cdot (\delta\epsilon\sqrt{2}) = \delta\epsilon$. We want to bound the angle $\angle qOs = \theta$. $\theta$ is maximized when $\angle O\widetilde{s}q = \angle Oq\widetilde{s}$. But since $\tan(\theta/2) \leq d/2$, we get $\theta = 2(\theta/2) \leq 2(\arctan d/2) \leq d = \delta\epsilon$. Also, as $k = \lceil \sqrt{2}/(\delta\epsilon) \rceil$, it follows that $\mathbb{V}$ has size $|\mathbb{V}| \leq 12/(\delta\epsilon)^2 + 18/(\delta\epsilon) + 18$, which completes the proof. □

### 4.2.3 The Basic Algorithm

We describe a first, simple method for our problem that computes an $\epsilon$-approximate solution and has running time $O(n/(\delta\epsilon)^2)$.

1. Compute a $\delta\epsilon$-discretization $\mathbb{V}$ of $\mathbb{S}^2$.
2. For each $p \in \mathbb{V}$,
   (a) Compute the set $V_p$ of vertices $v_t$ with $\angle(p, n_t) \leq (1 + \epsilon) \cdot \delta$.
   (b) Consider the subgraph of $G_{\mathcal{T}}$ induced* by the set $V_p$ and determine its connected component $C_p$ with maximum weight.
3. Return the set of triangles $T$ corresponding to the heaviest component $C_p$ found in Step 2 and the respective reference normal $\overrightarrow{p}$.

In the following we prove the correctness and running time of this algorithm.

**Lemma 9.** *Given a triangulated irregular network $\mathcal{T}$, and two real parameters $\delta > 0$ and $\epsilon > 0$ we can compute in $O(n/(\delta\epsilon)^2)$ time an $\epsilon$-approximate $\delta$-planar subset of triangles $T$ of $\mathcal{T}$.*

* In other words, this graph arises from $G_{\mathcal{T}}$ by keeping only those vertices and edges that lie entirely in $V_p$.

*Proof.* By Lemma 8, computing the $\delta\epsilon$-discretization takes $O(1/(\delta\epsilon)^2)$ time. For each element $p \in \mathbb{V}$ we have to determine the subgraph induced by $V_p$ and compute its connected components, which can be done in $O(n)$ time. So the total running time of Step 2 is $O(n/(\delta\epsilon)^2)$ which also dominates the overall running time.

For the correctness, observe that $\delta(1+\epsilon)$-planarity follows immediately from the formulation of the algorithm; we only consider triangles whose normals deviate at most $(1+\epsilon)\cdot\delta$ from some vector $\overrightarrow{r}$ and we only return triangles whose dual vertices in $G_{\mathcal{T}}$ form a connected component.

It remains to show that the weight of our computed set is at least that of an optimal $\delta$-planar set $T^*$. For set $T^*$ there exists a vector $\overrightarrow{r^*}$ such that for all triangles $t \in T^*$, $\angle(r^*, n_t) \leq \delta$. Let $p$ be a point in $\mathbb{V}$ for which $\angle(p, r^*) = \min_{u\in\mathbb{V}} \angle(u, r^*)$. By the definition of $\mathbb{V}$, the angle $\angle(p, r^*)$ must be at most $\delta\epsilon$. Then, for any triangle $t \in T^*$ the angle between $\overrightarrow{n_t}$ and $\overrightarrow{p}$ is at most $\delta + (\delta\epsilon) = \delta(1+\epsilon)$. Therefore for all $t \in T^*$, $v_t \in V_p$ and hence our algorithm will find a connected component with at least the same weight. □

The running time of the basic algorithm is optimal in terms of $n$. But one may ask whether the dependence on $\epsilon$ or $\delta$ can be improved. In particular, it would be nice to remove the dependence on $\delta$. In the following we will refine our algorithm to obtain a running time of $O(n/\epsilon^2)$.

### 4.2.4 The Refined Algorithm

There are two ideas which help the refined algorithm improve the running time. First we determine a set of reference normals $\mathbb{V}'$ of size $O(n/\epsilon^2)$ which contains all relevant reference normals, avoiding the inspection of $\Omega(1/(\delta\epsilon)^2)$ potential reference normals. Secondly by a bucketing scheme, we reduce significantly the number of times a triangle has to be considered. The refined algorithm proceeds as follows:

1. For each triangle $t \in \mathcal{T}$ with normal $n_t$, let $p_t$ be a point in $\mathbb{V}$ for which $\angle(p_t, n_t) = \min_{u\in\mathbb{V}} \angle(u, n_t)$; store $t$ in the bucket associated with $p_t$.

2. Determine a set $\mathbb{V}' \subset \mathbb{V}$ of potential reference normals as $\mathbb{V}' = \{p \in \mathbb{V} : \exists p_t$ with non-empty bucket and $\angle(p, p_t) \leq (1+2\epsilon)\cdot\delta\}$

3. For each $r \in \mathbb{V}'$,

   (a) Collect the set of triangles $N_r$ contained in buckets of reference normals $r' \in \mathbb{V}'$ with $\angle(r', r) \leq (1+2\epsilon)\cdot\delta$.

   (b) Prune $N_r$ keeping only triangles $t$ with $\angle(n_t, r) \leq (1+\epsilon)\cdot\delta$. Let $N'_r$ be the pruned set.

   (c) Consider the subgraph of $G_{\mathcal{T}}$ induced by the vertices corresponding to triangles in $N'_r$ and determine its heaviest component $C_r$.

4. Output the heaviest component $C_r$ from Step 3.

Before we prove the running time and the correctness of the algorithm, we state a small lemma which informally says that in the $\delta\epsilon$-discretization, the points are distributed somewhat sparsely.

**Lemma 10.** *Let $p$ be a point in the $\delta\epsilon$-discretization $\mathbb{V}$ constructed as in Lemma 8. Then the number of points $p' \in \mathbb{V}$ with $\angle(p, p') < (1 + 2\epsilon) \cdot \delta$ is $O(1/\epsilon^2)$.*

*Proof.* We first claim that for any two gridpoints $p_1, p_2 \in \mathbb{V}$, $\angle(p_1, p_2) \geq (2/9)(\delta\epsilon)$. It is easy to see that the minimal angle is attained between a corner $p_1$ of the cube $L$ and its nearest gridpoint $p_2$. Assume without loss of generality that $p_1 = (1, 1, 1)$ and $p_2 = (1, 1, 1 - (1/k))$. (Recall that $k$ is the size of the grid.) Let $\angle p_1 O p_2 = \theta$ and $\angle p_2 p_1 O = \phi$. In the triangle $\triangle p_1 O p_2$, we have $\sin\phi = \sqrt{2/3}$, $|p_1 p_2| = 1/k$ and $|Op_2| = \sqrt{2 + (1 - (1/k))^2}$. It follows from the law of sines that $\sin\theta = (|p_1 p_2|/|Op_2|)\sin\phi \geq \sqrt{2}/(3k)$. For $\delta\epsilon \leq 1/\sqrt{2}$ we get that $\theta \geq \sin\theta \geq (2/9)(\delta\epsilon)$, which proves our claim.

This fact implies that every grid point $p$ projected to $\overrightarrow{p}/|\overrightarrow{p}|$ on the sphere $\mathbb{S}^2$ has an empty spherical disk of radius at least $(\delta\epsilon)(2/9)$ that is free of other projected grid points. A spherical disk of radius $(1 + 2\epsilon) \cdot \delta$ therefore can contain only $O(1/\epsilon^2)$ grid points by a simple packing argument. □

Observe also that given a triangle normal $n_t$ we can determine the grid point $p_t$ with $\angle(p_t, n_t) = \min_{u \in \mathbb{V}} \angle(u, n_t)$ in constant time by first determining which face of the cube $L$ is hit by the ray $\overrightarrow{n_t}$ and then locating the position of the intersection point within the grid on that face. We now state the main theorem of this section.

**Theorem 12.** *Given a triangulated irregular network $\mathcal{T}$, and two real parameters $\delta > 0$ and $\epsilon > 0$ we can compute in $O(n/\epsilon^2)$ time an $\epsilon$-approximate $\delta$-planar subset of triangles $T$ of $\mathcal{T}$.*

*Proof.* We first prove correctness. Let $r_b$ and $T_b$ denote the reference normal and triangle set, respectively, as computed by the basic algorithm. We claim $r_b \in \mathbb{V}'$. This can be easily seen as follows: Assume $t \in T_b$. $t$ is stored in the bucket associated with some reference normal $r$ with $\angle(r, n_t) \leq \delta\epsilon$). Since $\angle(r_b, r) \leq \angle(r_b, n_t) + \angle(r_b, r) \leq (1 + \epsilon) \cdot \delta + \epsilon\delta = (1 + 2\epsilon) \cdot \delta$, it follows that $r_b \in \mathbb{V}'$. In addition, using the same argument, when $r_b$ is examined in Step 3, it must be that $t \in N_r$ and $t \in N'_r$. Thus our refined algorithm computes the same solution as the basic algorithm whose correctness was established before.

Let us now look at the running time. Step 1 of the refined algorithm takes $O(n)$ since for each $t$ we can determine $p_t$ in constant time as well as access the associated bucket using hashing. Step 2, where we form the set $\mathbb{V}'$, takes $O(n/\epsilon^2)$ since there are at most $n$ non-empty buckets. For each of the non-empty buckets, we explore $O(1/\epsilon^2)$ grid points in the neighborhood according to Lemma 10. Finally, for the overall running time of Step 3, observe that again according to Lemma 10, each triangle can be collected by at most $O(1/\epsilon^2)$ reference normals and that the running time of one iteration of Step 3 is $O(|N_r|)$. Therefore Step 3 takes $O(n/\epsilon^2)$ time overall. □

**Remark.** It is clear that we can avoid the pruning Step 3(b) in the algorithm by selecting a finer discretization initially. This would simplify the algorithm, but it would also increase by a constant factor the number of potential reference normals to be examined.

### 4.2.5 Scanning Algorithm

We propose another variant of our algorithm which improves the $1/\epsilon^2$ term but incurs an additional polylogarithmic factor in $n$. Similar to the exact algorithm in [SRWL04], we use a data structure by Thorup [Tho00] which allows the maintenance of connected components of a graph under insertions and deletions. The update time is $O(\log n(\log\log n)^3)$ amortized and one can also maintain which one is the heaviest component within the same time bound, see [SRWL04] for more details.

Recall that the basic algorithm of Section 2.3 naively tested all $O(1/(\epsilon\delta)^2)$ potential reference normals, each at a cost of $O(n)$. We will improve upon that by scanning the grid-points in a certain order such that the result of inspecting the previous grid point can be used in the inspection of the current. The order is defined as follows. Consider all grid-points $F_l$ which have a fixed $x$-coordinate: $F_l = \{p \in \mathbb{V} : p_x = l\}$. All grid-points in $F_l$ lie on the boundary of a square parallel to the $yz$-plane, so we call $F_l$ a *frame*. We pick one grid-point of the frame and compute, using the data structure by Thorup, the decomposition into connected components of the graph induced by all triangles located in buckets within distance $O((1+O(\epsilon))\cdot\delta)$. We then move on to the next grid-point of the frame in clockwise order always inserting/deleting triangles appearing/disappearing until we have reached the first grid-point again. Observe that the contents of a bucket are inserted at most twice and deleted at most once during the scan over the frame. Let $T_l$ be the set of triangles encountered. The running time of processing frame $F_l$ is clearly $O(1/(\epsilon^2\delta) + |T_l|\cdot\log n(\log\log n)^3)$, since the frame has $O(1/(\epsilon\delta))$ grid-points, we only inspect $O(1/(\epsilon^2\delta))$ buckets of grid-points nearby and have $O(|T_l|)$ insertion and deletion operations on Thorup's data structure.

It is easy to see that all grid points in $\mathbb{V}$ can be covered by $(k+1)+(k-1) = 2k = O(1/(\epsilon\delta))$ frames (recall $k$ is the grid-size), e.g. $k+1$ frames with fixed $x$-coordinates and another $k-1$ frames with fixed $y$-coordinates. The running time of the whole procedure is therefore $O(1/(\epsilon^3\delta^2)+(\sum_l |T_l|)\cdot\log n(\log\log n)^3)$. For $\sum|T_l|$ we observe that a bucket (and therefore each triangle in it) is inspected only by $O(1/\epsilon)$ frames using a very similar argument as our Lemma 10, so $\sum_l |T_l| = O(n/\epsilon)$, yielding the following result which for large values of $n$ is worse than the running time of the refined method, but for moderate values of $n$ and sufficiently small $\epsilon$ it may be of interest.

**Theorem 13.** *Given a triangulated irregular network $\mathcal{T}$, and two real parameters $\delta > 0$ and $\epsilon > 0$ we can compute in $O((n/\epsilon)\log n(\log\log n)^3) + 1/(\delta^2\epsilon^3))$ time an $\epsilon$-approximate $\delta$-planar subset of triangles $T$ of $\mathcal{T}$.*

## 4.3 Implementation

We have implemented the refined algorithm of Section 2.4 in `C++` using the LEDA library of efficient data types and algorithms [MN99]. We used several data sets representing fracture surfaces of metals. Input data were given $512\times512$ raster images with the intensity of each pixel corresponding to its height value. To obtain the TIN, we triangulated the point set by creating triangles $(i,j),(i+1,j),(i+1,j+1)$ and $(i,j),(i,j+1),(i+1,j+1)$ for all possible $i$ and $j$. See Figure 4.2 for our triangulation scheme.

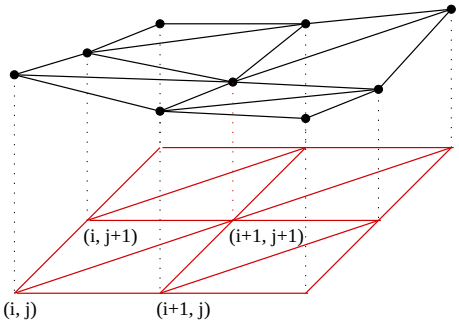


*Fig. 4.2:* Triangulation scheme for the array of height values

There are some aspects of the algorithm which can be tuned for better performance in practice, without of course sacrificing the theoretical guarantee of the output.

## Tuning for Practical Performance

Looking at the behavior of the original algorithm as stated in Section 4.2 we have come up with three heuristics which considerably reduced the running time of our algorithm in practice. We refer to Section 4.4 for actual timings of the improvements.

### *Prioritizing the Reference Normals*

Naturally it seems to make sense first to examine those reference normals for which the weight of the triangles in buckets nearby is large. So what we did in our implementation is to associate with each reference normal the weight of all triangles contained in buckets at distance at most $(1 + 2\epsilon) \cdot \delta$ and then process them in decreasing order of weight. We can stop examining further reference normals as soon as the weight of the best solution found so far exceeds the associated weight of the next reference normal. The weight information can easily be collected as follows: In the first phase while computing normals and bucketing the triangles we also take care of the weight and add it to the respective bucket. Then in the second phase, when potential reference normals are determined we propagate for each non-empty bucket its weight over all reference normals at distance at most $(1 + 2\epsilon) \cdot \delta$. The running time guarantee is only affected by the sorting step for the reference normals according to their weight. This costs $O((n/\epsilon^2) \cdot \log(n/\epsilon^2))$ time, which in practice was negligible.

### *Prepruning of Triangles*

In our data set, there is typically quite a number of triangles $t$ for which all three neighboring triangles have normals more than $2(1 + \epsilon)\delta$ off $\overrightarrow{n_t}$. These triangles are isolated and cannot be part of a larger connected region of the output. Thus we can preprune these triangles in the first phase when bucketing and simply not consider them in the next steps of the algorithm. We only have to ensure at the end that the computed region has weight at least as large as the heaviest single triangle of the terrain. This does not affect the theoretical running time guarantee.

***Fast Bucket Pruning of Relevant Triangles***

In Step 3(b) where we determine the set of relevant triangles $N'_r$ of the refined algorithm we collect all triangles in buckets within distance at most $(1 + 2\epsilon) \cdot \delta$ and test each triangle for normal deviation of at most $(1 + \epsilon) \cdot \delta$. But clearly, all triangles in buckets within distance $\delta$ will fulfill this requirement and therefore can be added without an additional check (which is relatively expensive as it involves floating-point computation). This does not affect the theoretical running time guarantee.

## 4.4 Experimental Evaluation

Our program was compiled using `g++ 3.2.3` with `-O` flag and LEDA 4.4 and timings were taken on a single processor Pentium 4, 1.8 GHz machine with 256 MB RAM running Debian Linux kernel version 2.4.20. `geomview` [ALMP95] was used for visualization of the results.

We benchmarked our implementation on several data sets provided by the Department of Materials Science at Universität Magdeburg, Germany. Researchers in the Materials Science are interested in surface topographies of materials since they provide useful information about the generation process and the internal structure of the material. Surfaces generated by fracture, wear, corrosion and machining are of interest. Among many other criteria, they want to examine feature-related parameters like facets in brittle fracture surfaces. All the data were acquired by confocal laser scanning microscopy. The following test sets were used for our experiments:

*T:* A surface with three artificially introduced almost planar triangles the largest of which our algorithm is supposed to detect.

*S2-28, S2-30, S2-31, S2-34:* Terrains with many terrace-like planar subregions.

*K8:* A very rough surface which exhibits only few planar subregions.

For each experiment we state the name of the test set (**Set** in the tables), the dimensions of the raster image (**Dim**), the number of triangles in the resulting terrain (**n**), the allowed deviation $\delta$ (in radians, note that $0.2$ radians is about $11.5$ degrees), the desired approximation quality $\epsilon$, the maximization objective (**Obj** which is either the total number (**C**) or area (**A**) of the triangles), the running time in seconds (**Time**), and finally the objective function value of the solution obtained (**Val.**).

### 4.4.1 Efficiency of Speed-Up Heuristics

In this part we show how much our proposed heuristics improve the running-time in practice. To allow for a more precise evaluation we have profiled the parts of the program which correspond to the single phases of our algorith. Table header **Norm.** denotes the time to compute the normals of all triangles and assigning each triangle to its closest bucket in the $\delta\epsilon$-discretization. Table header **Cand.** is the time to determine all potential reference normals. Table headers **Coll.**, **Prune**, and **Grow** are the accumulated times for collecting, pruning, and growing connected components on the relevant triangles for a reference normal. We experimented with all possible settings of slow or fast bucket pruning routine (**sP/fP**, Table header:

**B**), with or without prepruning (**PP/nPP**, Table header: **PP**), and both area (**A**) and number of triangles (**C**) as objective function (Table header: **Obj**). The results taken from the test sets **T** and **S2-28** can be found in Table 4.1. We remark that our algorithm indeed detected the largest triangular planar region artificially created in test set **T**.

#### *Pruning Heuristics*

As it can be observed, each of the pruning heuristics on its own yields a gain of at least 20% in running time. Combined the three heuristics reduce the running time by nearly a factor of two. The fast bucket pruning only affects this phase, whereas prepruning decreases the running time of all phases since the number of triangles to be looked at as well as the number of reference normals to be considered is reduced. Only the initial normal computation and bucketing phase requires more effort, which is though negligible.

#### *Area vs. Count*

In general, the running times for area maximization as objective function are higher than for just counting the number of triangles. This is due to the fact that prioritizing gets less effective when the maximum area is the objective. Very steep triangles have a very large area and hence the priorities of the respective reference normals become very high (if there are several of these steep triangles). So most of the time they have to be examined even though they will not lead to a large terrain (as they mostly occur isolated). This can only partly be compensated for by using the prepruning heuristic.

#### *Prioritizing*

In the same table the reference normals were always prioritized and we stopped examining as soon as the best solution found so far exceeded the priority level of the next reference normal. We have not listed the comparison with the unprioritized version, though the running time in this case is about that using slow bucket pruning and no prepruning with triangle area weights. Furthermore we note that the final best solution is typically found with one of the first reference normals, so most of the running time is spent on checking that no better solution exists.

| Set | Dim | n | $\delta$ | $\epsilon$ | Obj | B | PP | Time | | | | | | Obj |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Norm. | Cand. | Coll. | Prune | Grow | Total | Val. |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | C | sP | nPP | 2.7 | 0.27 | 15.55 | 64.07 | 28.85 | 112.94 | 5587 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | C | fP | nPP | 2.74 | 0.27 | 15.48 | 36.79 | 29.17 | 85.74 | 5587 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | C | sP | PP | 3.08 | 0.27 | 10.51 | 49.69 | 24.47 | 89.2 | 5587 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | C | fP | PP | 3.01 | 0.26 | 10.77 | 28.43 | 24.29 | 68.09 | 5587 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | A | sP | nPP | 2.8 | 0.28 | 23.65 | 73.65 | 34.54 | 137.31 | 2793 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | A | fP | nPP | 2.92 | 0.28 | 23.72 | 45.1 | 35.2 | 109.18 | 2793 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | A | sP | PP | 3.07 | 0.28 | 15.53 | 56.73 | 28.32 | 105.92 | 2793 |
| S2-28 | 512x512 | 522k | 0.2 | 0.2 | A | fP | PP | 3.1 | 0.27 | 15.46 | 33.25 | 27.84 | 81.78 | 2793 |
| T | 512x512 | 522k | 0.2 | 0.2 | C | sP | nPP | 2.91 | 0.3 | 25.52 | 76.92 | 33.54 | 140.9 | 10057 |
| T | 512x512 | 522k | 0.2 | 0.2 | C | fP | nPP | 2.87 | 0.29 | 25.56 | 44.74 | 34.02 | 109.24 | 10057 |
| T | 512x512 | 522k | 0.2 | 0.2 | C | sP | PP | 3.29 | 0.29 | 13.26 | 45.95 | 21.87 | 86.02 | 10057 |
| T | 512x512 | 522k | 0.2 | 0.2 | C | fP | PP | 3.15 | 0.28 | 13.5 | 27.17 | 21.47 | 66.74 | 10057 |
| T | 512x512 | 522k | 0.2 | 0.2 | A | sP | nPP | 3.06 | 0.29 | 32.75 | 86.01 | 39.02 | 163.3 | 7113 |
| T | 512x512 | 522k | 0.2 | 0.2 | A | fP | nPP | 3.06 | 0.29 | 33.4 | 51.91 | 38.7 | 129.69 | 7113 |
| T | 512x512 | 522k | 0.2 | 0.2 | A | sP | PP | 3.36 | 0.29 | 19.53 | 57.3 | 26.74 | 109.39 | 7113 |
| T | 512x512 | 522k | 0.2 | 0.2 | A | fP | PP | 3.34 | 0.3 | 19.43 | 34.49 | 27.21 | 86.85 | 7113 |

*Tab. 4.1:* Evaluation of acceleration heuristics and detailed profi ling.

| Set | Dim | n | $\epsilon$ | Obj | Time | Val. |
|---|---|---|---|---|---|---|
| S2-30 | 64x64 | 7k | 0.2 | C | 0.96 | 462 |
| S2-30 | 90x90 | 15k | 0.2 | C | 2.62 | 462 |
| S2-30 | 128x128 | 32k | 0.2 | C | 5.7 | 462 |
| S2-30 | 181x181 | 64k | 0.2 | C | 10.24 | 778 |
| S2-30 | 256x256 | 130k | 0.2 | C | 19.55 | 1209 |
| S2-30 | 384x384 | 293k | 0.2 | C | 41.9 | 2773 |
| S2-30 | 512x512 | 522k | 0.2 | C | 78.67 | 2773 |

*Tab. 4.2:* Running time versus $n$ for $\delta = 0.2$.

### 4.4.2 Dependence on $n$, $\epsilon$ and $\delta$

In the following we will examine more closely the dependence of the running time on the parameters $n$, $\epsilon$ and $\delta$. For the remaining part of the section we run experiments with all accelerating options turned on, i.e. with prioritized candidate selection, fast bucket pruning, as well as prepruning. In addition, we aim at maximizing the *number* of triangles in the almost planar region.

#### *Dependence on* $n$

To determine the dependence on $n$ we took an $i \times i$ crop from the lower left corner of the **S2-30** data set and varied $i$. See Table 4.2 and Figure 4.3 for the results. As to be expected, the running time grows linearly in the number of triangles. So we are very confident that our program can be used also for much larger datasets.

In Figure 4.3 we have denoted by an additional curve the time when the final solution was detected. Note that this happened within the first ten seconds due to the prioritization scheme.

#### *Dependence on* $\epsilon$

For the test set **S2-31** we have varied $\epsilon$. The results can be found in Table 4.3 and Figure 4.4. As to be expected the increase in running time with changing $\epsilon$ is the most pronounced. The increase kicks in for values of $\epsilon \leq 0.4$, making our approach not so practicable for $\epsilon < 0.05$. For values $\epsilon > 0.4$ our program for this test set behaves basically independent of $\epsilon$.

In Figure 4.4, we have denoted by an additional curve the time when the final solution was found. Note that this happened always within the first 20 seconds due to the prioritization scheme.

#### *Dependence on* $\delta$

Table 4.4 shows the running times on the data set **S2-34** for several values of $\delta$. The rather large variations in the running time are mostly due to the way in which the triangles are bucketed and the varying efficiency of the prioritizing heuristic. There seems to be no real dependence between the running time and the choice of $\delta$. Figure 4.5 depicts a top-view of the largest almost planar regions corresponding to the numbers in Table 4.4.

| Set | n | $\delta$ | $\epsilon$ | Obj | Time | Val. |
|---|---|---|---|---|---|---|
| S2-31 | 522k | 0.3 | 1.4 | C | 20.89 | 58115 |
| S2-31 | 522k | 0.3 | 1.2 | C | 17.95 | 58097 |
| S2-31 | 522k | 0.3 | 1 | C | 28.85 | 57624 |
| S2-31 | 522k | 0.3 | 0.8 | C | 19.47 | 50673 |
| S2-31 | 522k | 0.3 | 0.6 | C | 20.73 | 50673 |
| S2-31 | 522k | 0.3 | 0.4 | C | 38.66 | 7281 |
| S2-31 | 522k | 0.3 | 0.2 | C | 88.55 | 7281 |
| S2-31 | 522k | 0.3 | 0.1 | C | 262.8 | 7281 |
| S2-31 | 522k | 0.3 | 0.05 | C | 970.37 | 7281 |

*Tab. 4.3:* Running time versus $\epsilon$.

| Set | n | $\delta$ | $\epsilon$ | Obj | Time | Val. |
|---|---|---|---|---|---|---|
| S2-34 | 522k | 1 | 0.2 | C | 88.15 | 332091 |
| S2-34 | 522k | 0.9 | 0.2 | C | 73.05 | 257773 |
| S2-34 | 522k | 0.8 | 0.2 | C | 94.48 | 213976 |
| S2-34 | 522k | 0.7 | 0.2 | C | 90.19 | 188021 |
| S2-34 | 522k | 0.6 | 0.2 | C | 61.45 | 120855 |
| S2-34 | 522k | 0.5 | 0.2 | C | 57.83 | 107414 |
| S2-34 | 522k | 0.4 | 0.2 | C | 40.35 | 79463 |
| S2-34 | 522k | 0.3 | 0.2 | C | 96.63 | 1856 |
| S2-34 | 522k | 0.2 | 0.2 | C | 77.1 | 1856 |
| S2-34 | 522k | 0.1 | 0.2 | C | 68.22 | 1856 |
| S2-34 | 522k | 0.05 | 0.2 | C | 78.81 | 1856 |

*Tab. 4.4:* Running time versus $\delta$.

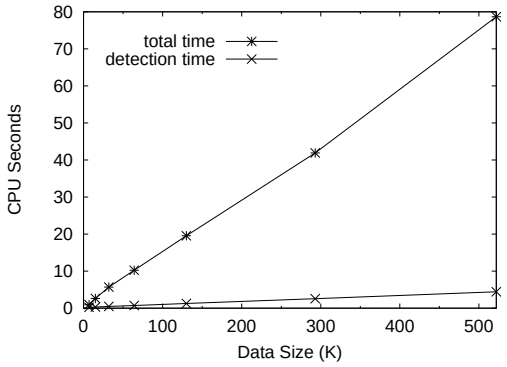


*Fig. 4.3:* Running time versus $n$ for $\delta = 0.2$, $\epsilon = 0.2$.

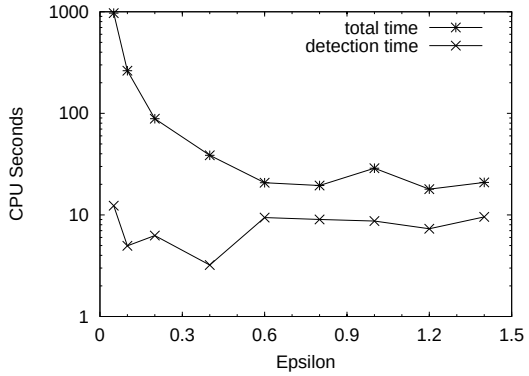


*Fig. 4.4:* Running time versus $\epsilon$ for $n = 522$k, $\delta = 0.3$.

### 4.4.3 Some More Examples

To compare running times between different test data sets, we have run our algorithm with the parameters $\delta = 0.2$, $\epsilon = 0.2$ on six of the data sets. We only considered a $500 \times 500$ crop as the set **K8** had a completely flat strip on the right part of the image, probably due to some problem during data acquisition. As we also used the *area* as maximization objective, the running times are slightly higher than in the experiments before. Table 4.5 shows the results.

Finally we synthetically generated some test sets by taking a surface with slightly (parabolic) increasing slope and perturbing unwanted data points. One of the results can be seen in Figures 4.6 and 4.7.

### 4.4.4 Further Remarks

The output of this approximate algorithm described in this chapter is more acceptable for the simple reason that it produces a guaranteed $\delta(1+\epsilon)$ connected subterrain of weight at least as much as the optimal $\delta$-planar set in comparison with the output of the heuristic (in previous

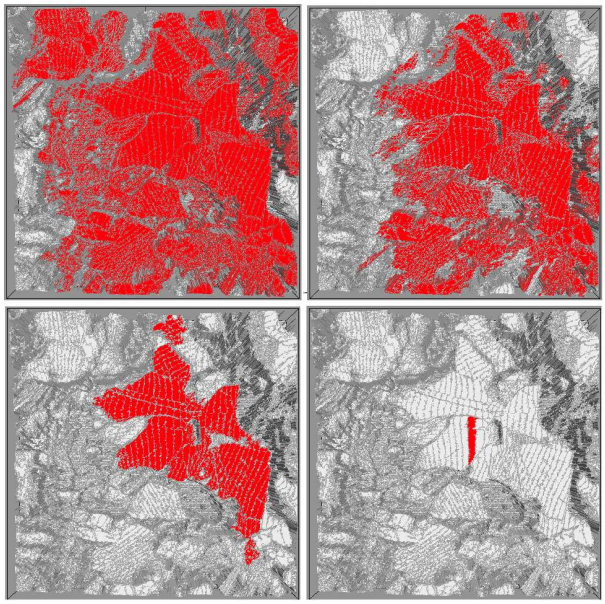

*Fig. 4.5:* Discovered regions for parameters $\delta = 1.0$, $\delta = 0.8$, $\delta = 0.4$, $\delta = 0.1$ (top to bottom, left to right) in fracture surface **S2-34**.

chapter) which comes with no guarantee. However, the heuristic implementation was simple and fast but it also can very easily miss a large almost planar region.

What might be interesting for practitioners is the fact that in all our cases, the final result of our algorithm was found within the first 20 seconds of the running time due to the prioritization heuristic. The remaining running time was spent on checking that there exists no larger planar region. If one is not required to have a strict guarantee for the quality of the result, one might simply stop the algorithm, for example, after thirty seconds and use the best solution computed so far.

| Set | n | $\delta$ | $\epsilon$ | Obj | Time | Val. |
|---|---|---|---|---|---|---|
| T | 498k | 0.2 | 0.2 | A | 79.24 | 7113 |
| S2-28 | 498k | 0.2 | 0.2 | A | 76.48 | 2793 |
| S2-30 | 498k | 0.2 | 0.2 | A | 85.17 | 1386 |
| S2-31 | 498k | 0.2 | 0.2 | A | 77.16 | 3640 |
| S2-34 | 498k | 0.2 | 0.2 | A | 80.49 | 928 |
| K8 | 498k | 0.2 | 0.2 | A | 106.1 | 1704 |

*Tab. 4.5:* Running times for various test sets.

## 4.5 Conclusions

We have presented simple approximation algorithms for detecting connected almost planar regions in a terrain. Running time of our algorithm is linear in $n$ and the dependence on $1/\epsilon$ is only quadratic. The algorithm for planarity detection has been implemented and tested on real-world data from an application domain in Materials Science and performs quite well in practice.

There are still a number of issues to be looked at particularly on the practical side. For instance it might be interesting to pose additional conditions on the structure of the connected almost planar region. At the moment our algorithm sometimes outputs large strip-like regions, so one may only consider fat planar regions. The regions computed by our algorithm also very often exhibit holes as can be seen in Figure 1.3, which might be undesirable. Different measures of 'near planarity' are of interest as well. In future work we plan to extend our implementation to *enumerate* the large almost planar regions in decreasing order of weight.

Finally our algorithm works for any surface mesh, thus it can also be used to detect flat regions on any polyhedral surface.

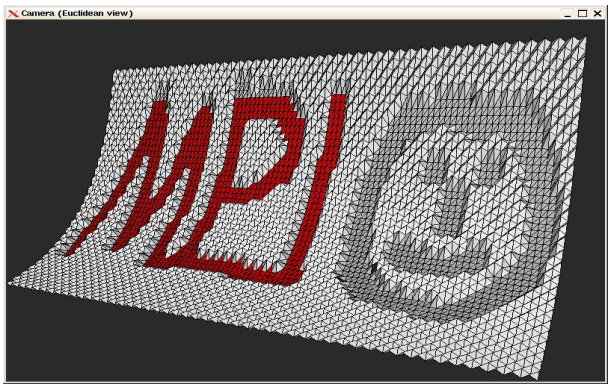


*Fig. 4.6:* Discovered region shaded in dark for $\delta = 0.3$, $\epsilon = 0.2$ in an artifi cial test set.

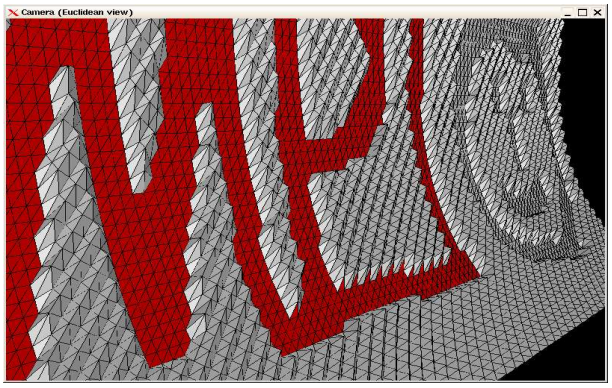


*Fig. 4.7:* Close-up of an almost planar region from the same test set as in Figure 4.6.

# Summary

We considered the following *placement problem*. Let $C$ be a compact set in $\mathbb{R}^2$ and let $S$ be a set of $n$ points in $\mathbb{R}^2$. The objective was to compute a translate of $C$ that contains the maximum number, $\kappa^*$, of points of $S$. This placement problem has been extensively investigated in the computational geometry literature for different types of sets $C$ mainly for its the useful application in clustering and pattern recognition. We have given a unified approach to solve this placement problem exactly and approximately in chapter 2. We approximated the problem to the following $\varepsilon$-optimal-placement as a translate $C + t$ of $C$ such that $|S \cap (C + t)| \geq (1 - \varepsilon)\kappa^*(C, S)$. In Section 2.5, it was shown using random sampling and/or bucketing, we can transform any deterministic algorithm for the optimal-placement problem to a Monte-Carlo algorithm for the $\varepsilon$-optimal-placement problem. Given a parameter $\gamma \geq 0$, the first algorithm in Section 2.5, based on a random-sampling technique, computes an $\varepsilon$-approximate placement in $O(n + T(\gamma n))$ time with error probability at most $sne^{-\varepsilon^2\gamma\kappa^*}$. The second algorithm combines the random-sampling technique with the bucketing technique and computes an $\varepsilon$-approximate placement in $O(T_g(n) + nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ time with error probability at most $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$. For circular disks and constant $\varepsilon$, the running time becomes $O(n \log n)$ and the error probability is at most $e^{-\sqrt{\kappa^* \log n}}$. If $C$ is fat and $m = O(1)$, by combining two levels of random sampling with the bucketing technique, we have computed an $\varepsilon$-approximate placement in $O(n)$ time for constant $\varepsilon$ with error probability at most $ne^{-\sqrt{\kappa^*}}$.

We have also presented a deterministic algorithm, adapted from the cutting algorithm of Chazelle, that computes an $\varepsilon$-approximate placement in $O(n^{1+\delta} + n/\varepsilon)$ time for a arbitrary constant $\delta > 0$ and $m = O(1)$. If $C$ is a convex $m$-gon, the running time is $O((n^{1+\delta} + n/\varepsilon) \log m)$.

Finally, in Section 2.7, we briefly described an algorithm for placing a unit disk in the plane such that the number of points contained is maximized. Our algorithm computed a placement of a disk of radius $(1 + \epsilon)$ which contains at least $\kappa^*$ points, where $\kappa^*$ denotes the maximum number of points in any unit disk. The running time of this algorithm is $O(n/\epsilon^2)$. We also sketch a more complicated variant running in $O(n + (n/\kappa^*) \cdot (1/\epsilon^3))$ time which is better for $\kappa^* = \omega(1/\epsilon)$.

Observe that in our approach of Section 2.7 the notion of approximation differs from the one used so far in that chapter which is the same as in [AHR$^+$02] (they approximate the size of the resulting set) and rather resembles the notion used in [HPM03] where one approximates the constraining radius. Not only are the running times of our algorithms linear in $n$ and the dependencies on $\epsilon$ reasonable, but also the constants involved are small enough to make them relevant in practice. The results on the placement problems were published in *10th Annual European Symposium of Algorithms* in 2001 ( [AHR$^+$02]) and partly in *20th ACM Symposium*

*on Computational Geometry (SoCG)* ( [FMR04]) in 2004.

We leave open the problem of improving the error probabilities of our Monte Carlo approximation algorithms for placement problem for small values of $k^*$. Both the randomized $O(n \log n)$ and $O(n)$ time algorithms we obtained had very low error probabilites when $\kappa^*$ is at least $\log n$.

Dickerson and Scharstein have considered the optimal placement problem if, besides translations, it is also allowed to rotate the set $C$. For the case when $C$ is a convex polygon with $m$ vertices, they give an algorithm for computing an optimal placement in $O(n^2 k^* m^2 \log(mn))$ time. In the worst case, this at least cubic in $n$. It would be interesting to know if our techniques can be extended to obtain a faster approximation algorithm allowing translation and rotation combined.

Prior to this thesis, there has been very little progress in locating approximately planar regions in terrain by the computational geometers. This thesis sheds a new light on this interesting geometric problem, motivated by the material scientists' need of analysing fractured surface topographies. We showed that the question of locating the largest approximately planar could be solved by considering the dual graph of the terrain triangulation (vertices correspond to triangles, edges to adjacencies between triangles). We embedded this degree-3 graph on the unit sphere $\mathbb{S}^2$ by placing each vertex $v$ at the position on $\mathbb{S}^2$ corresponding to the normal of the triangle represented by $v$. The vertices were weighted with the areas of the respective triangles. The largest almost planar region can then be found by determining the maximum weight connected component of this graph that is contained in a spherical disk of radius $\delta$.

The exact solution to this problem followed from the observation that at least one of the spherical disks that contains a maximum weight connected component has its center on a vertex of the arrangement of $n$ spherical disks which is defined by placing a disk of radius $\delta$ around each vertex $v$. But since just computing this arrangement takes $\Omega(n^2)$, a sub-quadratic running time for the overall algorithm cannot be obtained. Our algorithm first computed the arrangement and then made use of a data structure to dynamically maintain the connected components of a graph under insertions and deletions of edges, which finally yields a running time of $O(n^2 \log n (\log \log n)^3)$, instead of the naive $O(n^3)$.

Because of the expensive running time of the above algorithm, we have designed a grid-based heuristic that computes the same much faster. Also we have implemented this heuristic with some experimental results from the real data sets provided to us by the material scientists in Magdeburg University. This work has been published in *Materialwissenschaft und Werkstofftechnik* in 2002 ( [WLS$^+$02]) and *Discrete Applied Mathematics* in April 2004 ( [SRWL04]).

Since the output of the heuristic comes with no quality assurance, in Chapter 4, we presented a new approximation scheme for the problem which apart from giving a guarantee on the quality of the produced solution has been also implemented in practice and showed good performance on real-world data representing fracture surfaces. This approximate deterministic algorithm computes in $O(n/\varepsilon^2)$ time a connected $\delta$-planar set in which all the triangle normals can deviate at most $(1 + \varepsilon) \cdot \delta$ from the reference normal. With the reduced running time, the speed of the algorithm was competitive against previous heuristic algorithm using reasonably small values of $\varepsilon$. We also showed a variation of our algorithm which trades parameters of the running time, heuristics that speed up the program for practical inputs and was demonstrated

by thorough experiment that compares the effect of changing various parameters of the algorithms. We also gave a variant of this algorithm with a better dependence on $\epsilon$, but an extra polylogarithmic factor on $n$. Both the algorithms used $O(n)$ space for sufficiently large $n$, which is optimal. This result has been published in *20th ACM Symposium on Computational Geometry (SoCG)* in 2004 and a journal version of the same has been invited for *International Journal on Computational Geometry and Applications* ( [FMR04]).

We considered the problem of computing large regions in a terrain that are connected and approximately planar. We showed that the problem of computing the largest such region can be solved in a time that is roughly quadratic in the number of triangles in the terrain, and argued that it is unlikely to solve the problem faster. We leave open the problem of proving this rigorously. We also argued that it may even be hard to approximate this largest region. Proving this claim formally is also left as an open problem.

There is not yet an exact algorithm for the problem, which can solve it in $O(n^2)$ time. It might be interesting to consider additional conditions on the structure of the connected almost planar region. At present our algorithm sometimes outputs large strip-like regions, so one may only consider fat planar regions. The regions computed by our algorithm also very often exhibit holes, which might be undesirable. One can investigate different measures of 'near planarity' that suits the requirement of the applications or which is out of purely theoretical interests.

# Zusammenfassung

Wir betrachten das folgende *Platzierungsproblem*: Sei $C$ eine kompakte Menge im $\mathbb{R}^2$ and $S$ eine Menge von $n$ Punkten im $\mathbb{R}^2$. Das Ziel ist es, ein Verschiebung von $C$ zu berechnen, welche eine maximale Anzahl $\kappa^*$ an Punkten aus $S$ enthält. Aufgrund zahlreichen Anwendungen im Clustering und in der Mustererkennung wurde dieses Problem in den letzten Jahrzehnten in einer Vielzahl von Arbeiten im Gebiet der Algorithmischen Geometrie für verschiedenste Arten von Mengen betrachtet. In Kapitel 2 präsentieren wir einen allgemeinen Ansatz, um dieses Problem exakt und approximativ zu lösen. Unser Algorithmus berechnet eine $\epsilon$-optimale Verschiebung $C + t$ von $C$ sodass $|S \cap (C + t)| \geq (1 - \varepsilon)\kappa^*(C, S)$. In Abschnitt 2.5, zeigen wir, wie ein deterministischer Algorithmus für die optimale Platzierung in einen Monte-Carlo-Algorithmus zur $\epsilon$-optimalen Platzierung umgewandelt werden kann. Für einen beliebigen Parameter $\gamma \geq 0$ berechnet der erste Algorithmus in diesem Abschnitt eine $\varepsilon$-approximative Platzierung in Zeit $O(n + T(\gamma n))$ mit Fehlerwahrscheinlichkeit $sne^{-\varepsilon^2\gamma\kappa^*}$. Der zweite Algorithmus kombiniert zufällige Stichprobenentnahme mit einer Bucketingtechnik und berechnet eine $\varepsilon$-approximative Platzierung in $O(T_{\mathrm{g}}(n) + nQ(m) + nT(\alpha\beta\gamma\kappa^*)/\kappa^*)$ mit Fehlerwahrscheinlichkeit $s\alpha\beta\kappa^* e^{-\varepsilon^2\gamma\kappa^*}$. Für Kreissscheiben und konstantes $\varepsilon$, beläuft sich die Laufzeit auf $O(n \log n)$ und die Fehlerwahrscheinlichkeit auf $e^{-\sqrt{\kappa^* \log n}}$. Für den Fall, dass $C$ 'fett' ist und $m = O(1)$, kann eine $\varepsilon$-approximative Platzierung in Zeit $O(n)$ bei konstantem $\varepsilon$ mit Fehlerwahrscheinlichkeit $ne^{-\sqrt{\kappa^*}}$ berechnet werden. Wir stellen auch einen deterministischen Algorithmus basierend auf Chazelles Cuttings vor, welcher eine $\varepsilon$-approximative Platzierung in Zeit $O(n^{1+\delta} + n/\varepsilon)$ berechnet, wobei $\delta > 0$ eine beliebige Konstante ist und und $m = O(1)$. Für den Fall, dass $C$ ein konvexes $m$-gon ist, beträgt die Laufzeit $O((n^{1+\delta} + n/\varepsilon) \log m)$.

Schliesslich beschreiben wir in Abschnit 2.7 einen Algorithmus zur Platzierung einer Einheitskreissscheibe in der Ebene, sodass die Anzahl der überdeckten Punkte maximiert wird. Unser Algorithmus berechnet eine Platzierung einer Kreisscheibe mit Radius $(1 + \epsilon)$ welche mindestens $\kappa^*$ Punkte enthält, wobei $\kappa^*$ die maximale Anzahl von Punkten in einer Einheitskreisscheibe bezeichnet. Die Laufzeit dieses Algorithmus ist $O(n/\epsilon^2)$. Man beachte, dass der in Abschnitt 2.7 verwendete Approximationsbegriff vom zuvor im Kapitel verwendeten Begriff abweicht, welcher bis dahin derselbe wie in [AHR$^+$02] ist (man approximiert die Grösse der berechneten Menge). Vielmehr benutzen wir den Approximationsbegriff aus [HPM03], wo der einschränkende Radius approximiert wird. Die Laufzeiten unserer Algorithmen sind linear in $n$, die Abhngigkeiten von $\epsilon$ gutmütig und die auftrendenden Konstanten klein genug, um den Algorithmus praktisch nützlich zu machen. Die Resultate über Platzierungsprobleme wurden beim *10th Annual European Symposium of Algorithms* 2001 ( [AHR$^+$02]) und teilweise beim *20th ACM Symposium on Computational Geometry (SoCG)* ( [FMR04]) 2004 veröffentlicht.

Eine offenes Problem bleibt es, die auftretenden Fehlerwahrscheinlichkeiten unserer Monte Carlo Approximationsalgorithmen insbesondere für kleine Werte von $k^*$ zu verbessern. Dickerson und Scharstein haben das Platzierungsproblem betrachtet für den Fall dass ausser Verschiebungen, auch Rotationen der Menge $C$ erlaubt sind. Für konvexe $m$-gone beschreiben sie einen Algorithmus der eine optimale Platzierung in Zeit $O(n^2k^*m^2\log(mn))$ berechnet. Im schlimmsten Fall ist dies mindestens kubisch in $n$. Eine interessante Fragestellung ist es, ob unsere Techniken auch für den Fall von Verschiebung und Rotation zu schnelleren Algorithmen führen.

Vor dieser Arbeit gab es nur wenige Arbeiten aus dem Gebiet der algorithmischen Geometrie, die von der Bestimmung von fast-planaren Regionen in Terrains handeln. In unseren Resultaten haben wir erste Lösungsansätze für dieses interessante geometrische Problem vorgestellt, welches durch eine Anwendung in den Materialwissenschaften motiviert wurde, wobei es um die Analyse der Topographien von Bruchoberflächen geht. Wir zeigen, dass die grösste fast-planare Region durch genauere Untersuchung des dualen Graphen der Terraintriangulierung (Knoten entsprechen Dreiecken, Kanten zwischen adjazenten Dreiecken) ermittelt werden kann. Wir betten diesen Grad-3 Graphen auf die Einheitskugel $\mathbb{S}^2$ ein, indem wir jeden Knoten $v$ an der Position auf $\mathbb{S}^2$ platzieren, welche der normalisierten Dreiecksnormalen des entsprechenden Dreiecks entspricht. Die Knoten werden mit den Flächeninhalten der entsprechenden Dreiecke gewichtet. Die grösste fast-planare Region kann dann durch Bestimmung der Zusammenhangskomponente mit maximalem Gewicht, welche in einer Kugelkreissscheibe von Radius $\delta$ liegt, berechnet werden.

Die exakte Lösung für dieses Problem folgt aus der Beobachtung, dass mindestens eine der Kugelkreissscheiben, welche eine Zusammenhangskomponente maximalen Gewichts enthlt, ihren Mittelpunkt auf einem Knoten des Arrangements von $n$ Kreissscheiben auf der Kugeloberfläche hat. Hierzu konstruiert man um jeden Knoten $v$ einen Kreis mit Radius $\delta$. Da jedoch allein die Berechnung dieses Arrangements $\Omega(n^2)$ Zeit benötigt, ist eine subquadratische Laufzeit des gesamten Algorithmus nicht erreichbar. Unser exakter Algorithmus berechnet zuerst das Arrangement und berechnet dann mithilfe einer Datenstruktur zur dynamischen Verwaltung von Zusammenhangskomponenten eines Graphen unter Einfügungen und Löschungen von Kanten in Zeit $O(n^2\log n(\log\log n)^3)$ eine optimale Lösung (im Gegensatz zur naiven $O(n^3)$ Laufzeit).

Aufgrund der unpraktikablen Laufzeit des exakten Algorithmus haben wir eine Gitterbasierte Heuristik entwickelt, welche auch praktisch implementiert wurde. Wir haben diese Heuristik auf reellen Datensätze experimentell evaluiert, welche uns von den Materialwissenschaftlern an der Universität Magdeburg zur Verfügung gestellt wurden. Diese Resultate wurden in *Discrete Applied Mathematics* im April 2004 veröffentlicht ( [SRWL04]).

Da jedoch die Ausgabe der Heuristik keinerlei Qualitätsgarantie erfüllt, präsentieren wir in Kapitel 4 ein neuartiges Approximationsschema welches sowohl eine Ausgabequalität garantiert, als auch praktisch implementiert wurde und auf unseren Testdaten gute Resultate erzielte. Dieser deterministische Approximationsalgorithms berechnet in $O(n/\varepsilon^2)$ Zeit eine zusammenhängende $\delta$-planare Menge, in welcher alle Dreiecke höchstens um $(1+\varepsilon).\delta$ von einer Referenznormalen abweichen. Die Laufzeiten des Algorithmus sind vergleichbar mit vorherigen heuristischen Algorithmen, die keinerlei Qualitätsgarantie besassen. Dieses Resultat wurde beim *20th ACM Symposium on Computational Geometry (SoCG)* 2004 veröffentlicht und

zur Veröffentlichung im *International Journal on Computational Geometry and Applications* ( [FMR04]) eingeladen. In Zukunft wäre es vor allem von praktischem Interesse, die Struktur der erkannten fast-planaren Regionen stärker einzuschränken. Im Moment geben unsere Algorithmen in manchen Fällen lange, streifenartige Regionen zurück. Es könnte gewünscht sein, nur ”fette” fast-planere Regionen zu berücksichtigen. Ebenso werden oft Löcher in den erkannten Regionen – wie z.B. in Figure 1.3 – toleriert, was je nach Anwendung unerwünscht sein kann. Weitere, andere Masse für ’Fast-Planarität’ sind auch von Interesse.

# Bibliography


[AASS93] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

[AGR00] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 705–706, 2000.

[AHR+02] P. K. Agarwal, T. Hagerup, R. Ray, M. Sharir, M. Smid, and E. Welzl. Translating a planar object to maximize point containment. In *Proc. 10th Annu. European Sympos. Algorithms*, Lecture Notes Comput. Sci. Springer-Verlag, 2002.

[ALMP95] N. Amenta, S. Levy, T. Munzner, and M. Philips. Geomview: A system for geometric visualization. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C12–C13, 1995.

[AM00] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, 2000.

[AS98] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM J. Comput.*, 27:1016–1035, 1998.

[BCM99] H. Brönnimann, B. Chazelle, and J. Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28:1552–1575, 1999.

[BDP97] G. Barequet, M. Dickerson, and P. Pau. Translating a convex polygon to contain a maximum number of points. *Comput. Geom. Theory Appl.*, 8:167–179, 1997.

[BH91] H. Bast and T. Hagerup. Fast and reliable parallel hashing. In *Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 50–61, 1991.

[Boy73] A. Boyde. Quantitative photogrammetric analysis and qualitative stereoscopic analysis of SEM images. *J. of Microsc.*, 98:452–471, 1973.

[Cha93] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9:145–158, 1993.

[CL86] B. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36:1–16, 1986.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[DHKP97] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25:19–51, 1997.

[DK83] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27:241–253, 1983.

[DLSS95] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for $k$-point clustering problems. *J. Algorithms*, 19:474–503, 1995.

[DS98] M. Dickerson and D. Scharstein. Optimal placement of convex polygons to maximize point containment. *Comput. Geom. Theory Appl.*, 11:1–16, 1998.

[EE94] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.

[ESZ94] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest $k$-enclosing circle and related problems. *Comput. Geom. Theory Appl.*, 4:119–136, 1994.

[FKS84] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31:534–544, 1984.

[FMR04] S. Funke, T. Malamatos, and R. Ray. Finding planar regions in a terrain - in practice and with a guarantee. In *Proc. 20th ACM Symposium on Computational Geometry (SoCG)*, 2004. Also appeared in Proc. 20th European Workshop on Computational Geometry (EWCG) 2004, Sevilla.

[GO95] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[HdLT98] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 79–89, 1998.

[HPM03] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest $k$-enclosing disc. In *Proc. 11th Annu. European Sympos. Algorithms*, Lecture Notes Comput. Sci., pages 278–288. Springer-Verlag, 2003.

[HR90] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Inform. Process. Lett.*, 33:305–308, 1990.

[HSW00] A. Harasaki, J. Schmit, and J. C. Wyant. Improved vertical scanning interferometry. *Appl. Opt.*, 39:2107–2115, 2000.

[HU87] D. P. Huttenlocher and S. Ullman. Object recognition using alignment. In *Proc. 1st Internat. Conf. Comput. Vision*, pages 102–111, 1987.

[KLPS86] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1:59–71, 1986.

[LS95] H. P. Lenhof and M. Smid. Sequential and parallel algorithms for the $k$ closest pairs problem. *Internat. J. Comput. Geom. Appl.*, 5:273–288, 1995.

[LSW88] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. Object recognition by affine invariant matching. In *Proc. IEEE Internat. Conf. Comput. Vision Pattern. Recogn.*, pages 335–344, 1988.

[MN99] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, U.K., 1999.

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

[SB95] N. S. Sapidis and P. J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Transactions on Graphics.*, 14:171–200, 1995.

[Sch97] U. D. Schwarz. Scanning force microscopy. In S. Amelinckx, D. van Dyck, and J. van Landuyt van G. van Tendeloo, editors, *Handbook of Microscopy, Vol. II*, page 827. VCH Verlagsgesellschaft, Weinheim, 1997.

[SFRW98] O. Schwarzkopf, U. Fuchs, G. Rote, and E. Welzl. Approximation of convex figures by pairs of rectangles. *Comput. Geom. Theory Appl.*, 10:77–87, 1998.

[Sha91] M. Sharir. On $k$-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.

[SRWL04] M. Smid, R. Ray, U. Wendt, and K. Lange. Computing large planar regions in terrains, with an application to fracture surface. *Discrete Applied Mathematics*, 139: Issues 1-3:253–264, 2004.

[Tho00] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32th Annu. ACM Sympos. Theory Comput.*, pages 343–350, 2000.

[WLS+02] U. Wendt, K. Lange, M. Smid, R. Ray, and K. Tönnies. Surface topography quantification by integral and feature-related parameters. *Materialwissenschaft und Werkstofftechnik*, 33(10):621–627, 2002.

[YWM99] Y. Yamaguchi, M. K. Weldon, and M. D. Morris. Fractal characterization of SERS-active electrodes using extended focus reflectance microscopy. *Appl. Spectrosc.*, 53:127–132, 1999.

# Curriculum Vitae

## Personal Details


| | |
|---|---|
| Name: | Rahul Ray |
| Date of birth: | December 30, 1973 |
| Nationality: | Indian |
| Office address: | Max-Planck Institut für Informatik(MPII)<br>Algorithms and Complexity Group (AG1)<br>Im Stadtwald<br>D-66123 Saarbrücken<br>Germany<br>Phone: +49 (0)681 9325122<br>Fax: +49 (0)681 9325199<br>**http://www.mpi-sb.mpg.de/~ rahul**<br>E-mail: `rahul@mpi-sb.mpg.de` |


## Education

B.Sc Physics, Mathematics and Chemistry, Calcutta University, India, 1992–95.

M.E. Computer Science & Automation, Indian Institute of Science, Bangalore, 1995–1999.
Project: *Graph Drawing.*
Advisors: `Dr.Ramesh Hariharan, Dr.Sanjay Jain`

Ph.D. Computer Science, Max-Planck Institut für Informatik, Germany, Aug 2001-Present.
Expecting to finish in a couple of months.
Thesis title: *Geometric Algorithms for Object Placement and Planarity in a Terrain*
Advisors: `Prof.Dr.Kurt Mehlhorn`

## Scholarships & Sponsorship

- *Institute Scholarship* during Aug, 1995 – July, 1998 at IISc, Bangalore.
- *GATE* during Aug, 1998 – Jan, 1999 at IISc, Bangalore
- *DFG* grant April, 2000 – July, 2001 at Magdeburg University, Germany for the post of research scientist
- *IMPRS* funding from Aug, 2001 – till date during Ph.D studies at MPII

## Research & Industrial Experience

- April, 2000 – July, 2001: **Research Scientist** at Magdeburg University in Germany.
- June 5, 1999 – March 7, 2000: Systems Engineer at Siemens Information Systems Limited, Bangalore, India.
- May, 1998 – August, 1998: Summer Trainee at Texas Instrument in Bangalore, India.

## Publications

### Journal Publications

1. M. Smid, R. Ray, U. Wendt , K. Lange *Computing Large Planar Regions in Terrains, with an Application to Fracture Surface* , Discrete Applied Mathematics, Volume 139, Issues 1-3, 30 April 2004, Pages 253-264
2. U. Wendt , K. Lange , M. Smid , R. Ray , K.-D. Toennies *Surface Topography Quantification by Integral and Feature-related Parameters*, Materialwissenschaft und Werkstofftechnik (Materials Science and Engineering Technology) Volume 33, Issue 10, 2002, Page 621-627

### Conference Publications

1. K. Lange, R. Ray, M. Smid, U. Wendt *Computing planar regions in terrains* , Proceedings of the 8th International Workshop on Combinatorial Image Analysis (IWCIA), 2001, pp. 139–151. Appears also in Electronic Notes in Theoretical Computer Science, Volume 46.
2. U. Wendt, K. Lange, M. Smid, R. Ray *Surface Topography Quantification using Computational Geometry*, Dreilaendertagung fuer Elektronenmikroskopie (Conference on Modern Microscopical Methods), Innsbruck/Austria, 9-14 Sept. 2001
3. P. K. Agarwal, T. Hagerup , R. Ray , M. Sharir, M. Smid , E. Welzl *Translating a Planar Object to Maximize Point Containment*, 10th Annual European Symposium on Algorithms (ESA), in Rome, Italy, September 2002. Appears also in Lecture Notes in Computer Science, pp 42-53. (LNCS 2461, Springer)
4. S. Funke, T. Malamatos , R. Ray *Finding Planar Regions in Terrain - In Practice and with a guarantee*, Proc. 20th ACM Symposium on Computational Geometry (SoCG) 2004, New York
5. S. Funke, T. Malamatos , R. Ray *Finding Planar Regions in Terrain* , Proc. 20th European Workshop on Computational Geometry (EWCG) 2004, Sevilla