Finding Planar Regions in a Terrain – In Practice and with a Guarantee*

Stefan Funke† Theocharis Malamatos Rahul Ray

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

{funke,tmalamat,rahul}@mpi-sb.mpg.de

ABSTRACT

We consider the problem of computing large connected regions in a triangulated terrain of size n for which the normals of the triangles deviate by at most some small fixed angle. In previous work an exact near-quadratic algorithm was presented, but only a heuristic implementation with no guarantee was practicable. We present a new approximation algorithm for the problem which runs in $O(n/\epsilon^2)$ time and—apart from giving a guarantee on the quality of the produced solution—has been implemented and shows good performance on real data sets representing fracture surfaces consisting of around half a million triangles. Further we present a simple approximation algorithm for a related problem: given a set of n points in the plane, determine the placement of the unit disc which contains most points. This algorithm runs in linear time as well.

Categories and Subject Descriptors: I.3 [Computing Methodologies]: Computer Graphics; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

General Terms: Algorithms Measurement

Keywords: Planarity Terrain Approximation

1. INTRODUCTION

A terrain 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)*. 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 δ, where $\overrightarrow{n_t}$ denotes the normal of triangle t and δ 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).

Note that this definition of 'almost planar' is not the only possible one. But as this notion has been used in previous work [6], we decided to borrow their definition. One advantage of this definition is that it does not depend on the sizes of the terrain triangles. The problem of finding nearly planar regions in a terrain has real-world applications in Materials Science where researchers are interested in analyzing fracture surface topographies [9]. Figure 1 shows part of a

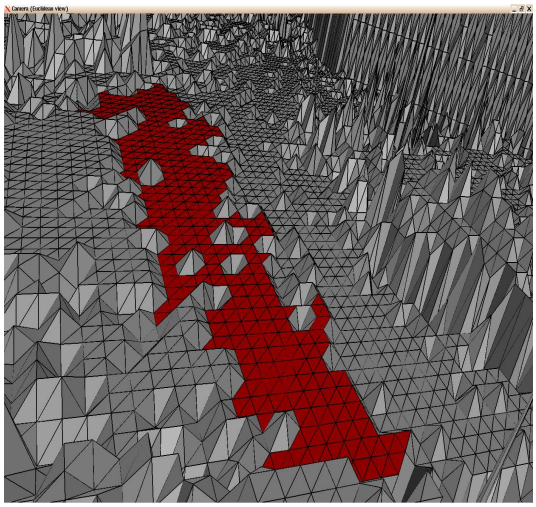

Figure 1: Close-up on a fracture surface and a connected almost planar region in dark.

*Partly supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

†Part of this research was conducted while the author was visiting the University of Illinois at Urbana-Champaign, USA.

SCG'04, June 9–11, 2004, Brooklyn, New York, USA.

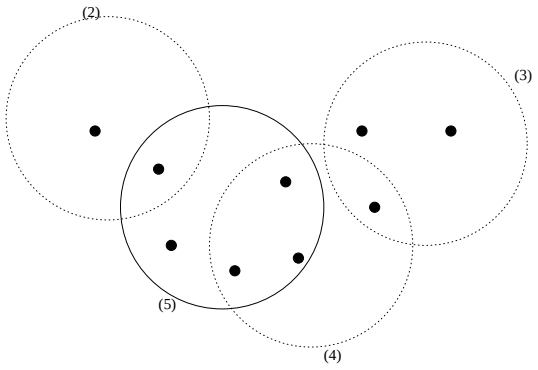

Figure 2: Four placements of a disc and their respective containment of a point set.

triangulated fracture surface and the approximately planar region found by our implemented algorithm. Other applications are also possible in terrain simplification and analysis.

In [6] the above problem was reduced to the following: Given an embedding of a degree-3 graph G on the unit sphere $\mathbb{S}^2$ with weighted vertices, compute a connected subgraph of maximum weight that is contained in some spherical disk of fixed radius. So one might be also interested in the following related problem: given a set of points S in $\mathbb{R}^2$, determine the placement of a unit disk that contains the maximum number κ^* of points from S. (See Figure 2.) This problem has many applications in clustering and pattern recognition, see for example [5]. Solving this problem covers also the variant where the disk to be placed has radius r as we can use scaling.

1.1 Related Work

Planarity Detection

Lange, Ray, Smid, and Wendt [6] solve the problem by considering the dual graph of the terrain triangulation (vertices correspond to triangles, edges to adjacencies between triangles). They embed 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 are 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 δ.

The exact solution to this problem follows from their 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 δ around each vertex v. But since just computing this arrangement takes $\Omega(n^2)$, they cannot obtain a sub-quadratic running time for the overall algorithm. In fact their algorithm first computes the arrangement and then uses 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 $O(n^3)$.

As this algorithm is far from being practical, in the same paper, the authors present an easy-to-implement heuristic which computes almost planar regions quite quickly but unfortunately without any guarantee for the computed solution. In fact they show examples where their algorithm fails to detect supposedly almost planar regions.

Point Containment in a Disk

In [1], Agarwal et al. present a probabilistic Monte-Carlo-type algorithm which given a set of points in $\mathbb{R}^2$ computes a placement of the unit disk which contains $(1-\epsilon)\cdot\kappa^*$ points with high probability. Here κ^* denotes the number of points in the optimal placement. The running time of their algorithm for placing a unit disk is $O(n \log n)$ where ϵ is treated as a constant. Their work also includes results for the placement of other non-disk objects. Further they present a deterministic approximation algorithm which is based on cuttings and hence seems less attractive in practice.

A related problem is the following: Given a set of points in $\mathbb{R}^2$, find the disk of minimum radius that contains at least k of these points. In [4], Har-Peled and Mazumdar present an approximation algorithm which computes a disk of radius $r^*\cdot(1+\epsilon)$, where r^* denotes the radius of the optimal disk, which contains at least k points. The running time of their algorithm is $O(n + n\cdot\min\{\frac{1}{k\epsilon^3}\log^2\frac{1}{\epsilon}, k\})$.

1.2 Our Results

The two theoretical contributions of this paper are simple approximation algorithms for the planarity detection and the unit disk point containment problem.

In Section 2, we present an algorithm which, given some parameters δ and ϵ 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 δ. The running time of this algorithm is $O(n/\epsilon^2)$. We sketch also a variant of this algorithm with a better dependence on ϵ but an extra polylogarithmic factor on n. For n sufficiently large, both algorithms use optimal $O(n)$ space.

Section 3 briefly describes 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 κ^* points, where κ^* 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 in [1] (they approximate the size of the resulting set) and rather resembles the notion used in [4] where one approximates the constraining radius/angle. Not only are the running times of our algorithms linear in n and the dependencies on ϵ reasonable, but also the constants involved are small enough to make them relevant in practice.

The experimental and perhaps main contribution of this paper 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. We were provided with the data by the materials science department at Universität Magdeburg.

2. FINDING A LARGE PLANAR REGION APPROXIMATELY

2.1 Preliminaries

Let $\mathcal{T}$ be a 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 *δ-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* δ-planar if it has the largest possible weight over all δ-planar subsets of $\mathcal{T}$.

Our Notion of Approximation

There are at least two ways to define the notion of an ϵ-approximate δ-planar set T. One way would be to require T to be δ-planar and of weight at least $(1 - \epsilon)$ times the weight of an optimal δ-planar set. Unfortunately solving this type of approximation seems to be as difficult as solving the problem exactly, see [6] for more details. We adopt the following notion of approximation: A subset of triangles T of $\mathcal{T}$ is *ϵ-approximate* δ-planar if it is $\delta(1+\epsilon)$-planar and has weight at least as large as an optimal δ-planar set.

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 2.1. *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 2.1. *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.*

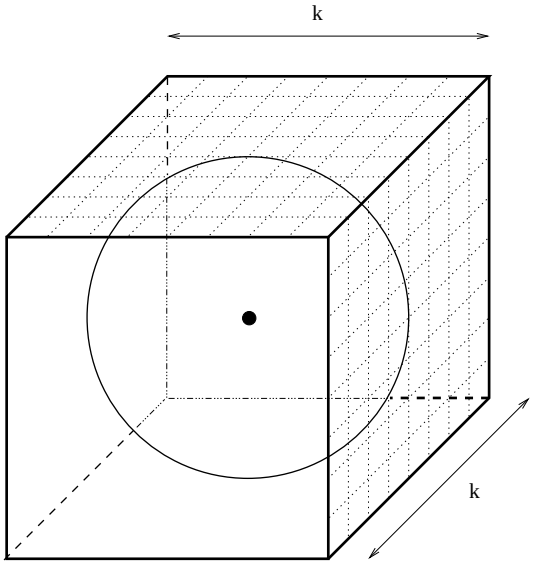

Figure 3: Cube with sidelength two containing $\mathbb{S}^2$ and with a $k \times k$ grid on each of its faces.

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 3. 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.

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 $\angle qOs = \theta$. θ 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. □

2.3 The Basic Algorithm

We describe a first, simple method for our problem that computes an ϵ-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[1] 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 2.2. *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 ϵ-approximate δ-planar subset of triangles T of $\mathcal{T}$.*

PROOF. By Lemma 2.1, 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 δ-planar set T^*. For set T^* there exists a vector $\overrightarrow{r^*}$ such that for all triangles $t \in T^*$, $\angle(r^*, n_t) \le \delta$. 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 ϵ or δ can be improved. In particular, it would be nice to remove the dependence on δ. In the following we will refine our algorithm to obtain a running time of $O(n/\epsilon^2)$.

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 $\angle(p, p_t) \le (1+2\epsilon)\cdot\delta\}$

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 $\angle(r', r) \le (1+2\epsilon)\cdot\delta$.

(b) Prune N_r keeping only triangles t with $\angle(n_t, r) \le (1+\epsilon)\cdot\delta$. 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 2.3. *Let p be a point in the $\delta\epsilon$-discretization $\mathbb{V}$ constructed as in Lemma 2.1. 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 $p_1, p_2 \in \mathbb{V}$, $\angle(p_1, p_2) \ge (2/9)(\delta\epsilon)$. It is easy to see that the minimal angle is attained between a corner p_1 of the cube L and its nearest grid-point 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 $\sin\theta = (|p_1p_2|/|Op_2|)\sin\phi \ge \sqrt{2}/(3k)$. For $\delta\epsilon \le 1/\sqrt{2}$ we get that $\theta \ge \sin\theta \ge (2/9)(\delta\epsilon)$, 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 2.1. *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 ϵ-approximate δ-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 $\angle(r, n_t) \le \delta\epsilon)$. Since $\angle(r_b, r) \le \angle(r_b, n_t) + \angle(r_b, r) \le (1+\epsilon)\cdot\delta + \epsilon\delta = (1+2\epsilon)\cdot\delta$, 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

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

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 2.3. Finally, for the overall running time of Step 3, observe that again according to Lemma 2.3, 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.

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 [6], we use a data structure by Thorup [8] 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 [6] 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 2.3, 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 ϵ it may be of interest.

THEOREM 2.2. *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 ϵ-approximate δ-planar subset of triangles T of $\mathcal{T}$.*

3. FINDING THE UNIT DISK CONTAINING MOST POINTS

Our algorithm for planarity detection was based on the idea of determining a large connected component inside a spherical disk of a fixed radius. A naturally related problem is the following:

> Given a set S of points in the plane, determine a placement (x^*, y^*) for the center of a disc U of unit radius such that the number $\kappa = |U_{(x,y)} \cap S|$ of covered points is maximized.

We will present a simple algorithm which in $O(n/\epsilon^2)$ time determines a placement (x, y) of a disc $U^{1+\epsilon}$ of radius $1+\epsilon$ with $|U^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$. Here κ^* denotes the maximal number of points of S contained in a unit disk. Note that again as in the previous section we are approximating the radius and not the number of points captured.

3.1 Simple Approximation Algorithm

The idea of the algorithm is first to get a rough estimate of κ^* 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 $C = \{(i, j) : \sum_{g=i-1}^{i+1} \sum_{h=j-1}^{j+1} k_{(g,h)} \geq k/4\}$ be the grid cells which have a 'well-occupied' neighborhood.
3. For each cell $(i, j) \in C$,
   (a) Place a grid of width ϵ over (i, j).
   (b) Check each of the $O(1/\epsilon^2)$ grid points as center of a potential disc $U^{(1+\epsilon)}$ 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 κ^* points.

LEMMA 3.1. *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 $U_{(x^*,y^*)} \subset U^{1+\epsilon}_{(x',y')}$.*

PROOF. Let us first show that the center of an optimal placement of the unit disc falls inside a cell $c \in C$. 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 $c \in C$. 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 $U_{(x^*,y^*)}$. □

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

THEOREM 3.1. *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 $U^{1+\epsilon}$ of radius $(1+\epsilon)$ with $|U^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$, where κ^* 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 $|C| = O(n/k)$, 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. □

3.2 A Variant for Large κ^*

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 ϵ. 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 [3].

For each cell $c = (i, j) \in C$, we put a grid of width ϵ 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 ϵ-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 3.2. *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 $U^{1+\epsilon}$ of radius $(1+\epsilon)$ with $|U^{1+\epsilon}_{(x,y)} \cap S| \geq \kappa^*$, where κ^* denotes the maximal number of points in S contained in a unit disc.*

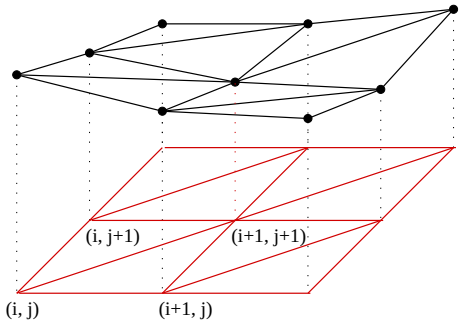

Figure 4: Triangulation scheme for the array of height values

4. IMPLEMENTATION

We have implemented the refined algorithm of Section 2.4 in `C++` using the LEDA library of efficient data types and algorithms [7]. We used several data sets representing fracture surfaces of metals. Input data were given 512×512 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 for our triangulation scheme.

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 2 we have come up with three heuristics which considerably reduced the running time of our algorithm in practice. We refer to Section 5 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 δ 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.

5. 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` [2] 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 δ (in radians, note that 0.2 radians is about 11.5 degrees), the desired approximation quality ϵ, 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.**).

5.1 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 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.

5.2 Dependence on n, ϵ and δ

In the following we will examine more closely the dependence of the running time on the parameters n, ϵ and δ. 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.

Set	Dim	n	δ	ϵ	Obj	B	PP	Time						Obj
								Norm.	Cand.	Coll.	Prune	Grow	Total	Val.
S2-28	512x512	522k	0.2	0.2	#	sP	nPP	2.7	0.27	15.55	64.07	28.85	112.94	5587
S2-28	512x512	522k	0.2	0.2	#	fP	nPP	2.74	0.27	15.48	36.79	29.17	85.74	5587
S2-28	512x512	522k	0.2	0.2	#	sP	PP	3.08	0.27	10.51	49.69	24.47	89.2	5587
S2-28	512x512	522k	0.2	0.2	#	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	#	sP	nPP	2.91	0.3	25.52	76.92	33.54	140.9	10057
T	512x512	522k	0.2	0.2	#	fP	nPP	2.87	0.29	25.56	44.74	34.02	109.24	10057
T	512x512	522k	0.2	0.2	#	sP	PP	3.29	0.29	13.26	45.95	21.87	86.02	10057
T	512x512	522k	0.2	0.2	#	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

Table 1: Evaluation of acceleration heuristics and detailed profiling.

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

Table 2: Running time versus n for $\delta = 0.2$.

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 2 and Figure 5 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 5 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 ϵ

For the test set **S2-31** we have varied ϵ. The results can be found in Table 3 and Figure 6. As to be expected the increase in running time with changing ϵ 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 ϵ.

In Figure 6, 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.

Set	n	δ	ϵ	Obj	Time	Val.
S2-31	522k	0.3	1.4	#	20.89	58115
S2-31	522k	0.3	1.2	#	17.95	58097
S2-31	522k	0.3	1	#	28.85	57624
S2-31	522k	0.3	0.8	#	19.47	50673
S2-31	522k	0.3	0.6	#	20.73	50673
S2-31	522k	0.3	0.4	#	38.66	7281
S2-31	522k	0.3	0.2	#	88.55	7281
S2-31	522k	0.3	0.1	#	262.8	7281
S2-31	522k	0.3	0.05	#	970.37	7281

Table 3: Running time versus ϵ.

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

Table 4: Running time versus δ.

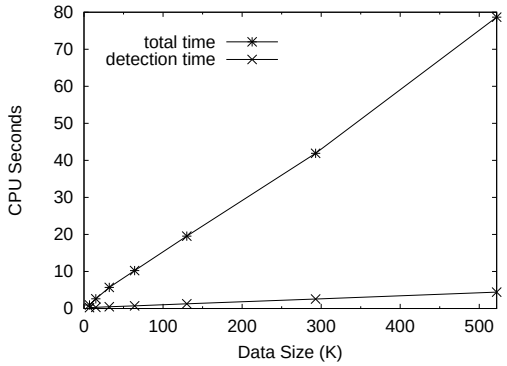

Figure 5: Running time versus n for $\delta = 0.2$, $\epsilon = 0.2$.

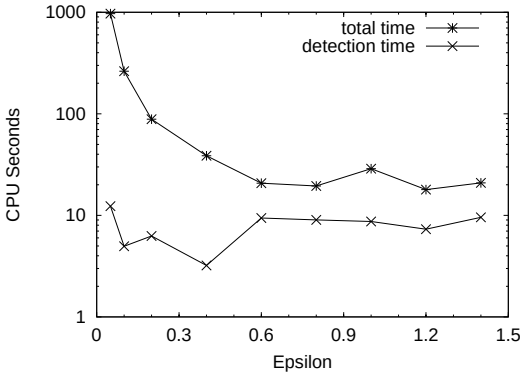

Figure 6: Running time versus ϵ for $n = 522$k, $\delta = 0.3$.

Dependence on δ

Table 4 shows the running times on the data set **S2-34** for several values of δ. 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 δ. Figure 7 depicts a top-view of the largest almost planar regions corresponding to the numbers in Table 4.

5.3 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 500×500 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 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 8 and 9.

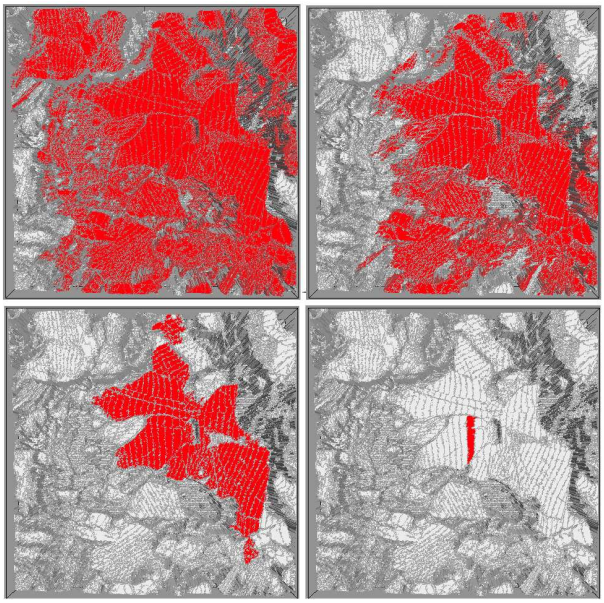

Figure 7: Discovered regions for parameters $\delta = 1.0$, $\delta = 0.8$, $\delta = 0.4$, $\delta = 0.1$ (top to bottom, left to right) in fracture surface S2-34.

Set	n	δ	ϵ	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

Table 5: Running times for various test sets.

5.4 Further Remarks

So far we have not compared our implementation with the heuristic described in [6], since we could only obtain a partial implementation of this algorithm (the boundary correction step and the shifting mentioned there were not included). This implementation was simple and fast but it also can very easily miss a large almost planar region and has no control on δ. A full implementation would be harder to fool, but it would also substantially increase the running time (we believe to more than 40 seconds). Still bad examples can be easily constructed for it too.

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.

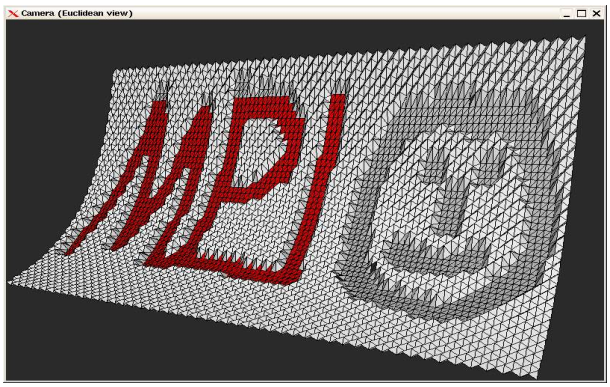

Figure 8: Discovered region shaded in dark for $\delta = 0.3$, $\epsilon = 0.2$ in an artificial test set.

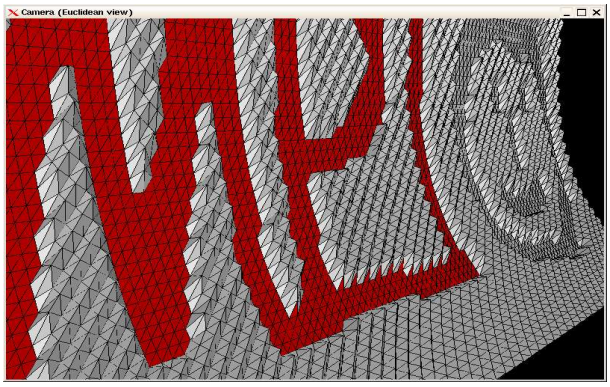

Figure 9: Close-up of an almost planar region from the same test set as in Figure 8.

6. CONCLUSIONS

We have presented simple approximation algorithms for detecting connected almost planar regions in a terrain and placing a unit disk in the plane to maximize point containment. Their running time 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, 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.

Acknowledgements

We are grateful to Ulrich Wendt and Katharina Lange from the Department of Material Sciences of the Otto-Guericke Universität Magdeburg for providing us with real-world test data for our implementation. Furthermore we thank Edgar Ramos for helpful comments on the problem.

References

[1] 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., pages 42–53. Springer-Verlag, 2002.

[2] 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.

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

[4] 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.

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

[6] K. Lange, R. Ray, M. Smid, and U. Wendt. Computing large planar regions in terrains. In *Proc. 8th Int. Workshop on Combinatorial Image Analysis*, volume 46 of *ENTCS*. Elsevier, 2001. (Also to appear in *Discrete Applied Mathematics*).

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

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

[9] 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.