Electronic Notes in Theoretical Computer Science 46 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume46.html` 13 pages

Computing Large Planar Regions in Terrains

Katharina Lange [1,2], Rahul Ray [1,3], Michiel Smid [1,4], and Ulrich Wendt [1,5]

Department of Materials Science
Otto-von-Guericke University of Magdeburg
D-39106 Magdeburg, Germany

Abstract

We consider the problem of computing the largest region in a terrain that is approximately contained in some two-dimensional plane. We reduce this problem to the following one. Given an embedding of a degree-3 graph G on the unit sphere $\mathbb{S}^2$, whose vertices are weighted, compute a connected subgraph of maximum weight that is contained in some spherical disk of a fixed radius. We given an algorithm that solves this problem in $O(n^2 \log n (\log\log n)^3)$ time, where n denotes the number of vertices of G or, alternatively, the number of faces of the terrain. We also give a heuristic that can be used to compute sufficiently large regions in a terrain that are approximately planar. We discuss a web-based implementation of this heuristic, and show some results for terrains representing three-dimensional (topographical) images of fracture surfaces of metals obtained by confocal laser scanning microscopy.

1 Introduction

A *terrain* is the surface in $\mathbb{R}^3$ described by a real-valued function in two real variables. If this function is piecewise linear and the surface consists of a collection of triangles, then the terrain is called a *triangulated irregular network (TIN)*. We consider the problem of computing the largest connected region in a TIN that is approximately planar. To be more precise, given a TIN $\mathcal{T}$, we want to compute that subset T of the triangles of $\mathcal{T}$ such that (i) for each triangle $t \in T$ there is another triangle $t' \in T$ such that t and t' share an edge,

[1] The authors were supported by the Deutsche Forschungsgemeinschaft, grants WE 2301/3-2 and SM 57/4-1

[2] Email: `Katharina.Lange@mb.Uni-Magdeburg.DE`
[3] Email: `rahul@isg.cs.uni-magdeburg.de`
[4] Email: `michiel@isg.cs.uni-magdeburg.de`
[5] Email: `Ulrich.Wendt@mb.Uni-Magdeburg.DE`

(ii) there is a two-dimensional plane that approximately contains all triangles of T, and (iii) the total area of the triangles of T is as large as possible.

In order to define this problem rigorously, we have to specify the notion of being approximately contained in a plane. In this paper, we define this notion as follows. Let ϵ be a small positive real number, and let T be a set of triangles satisfying (i) above. We say that T is *ϵ-planar* if there is a vector $\mathbf{c}$ such that the normal vectors of all triangles in T make an angle of at most ϵ with $\mathbf{c}$. 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 $\mathbb{S}^2_+ := \mathbb{S}^2 \cap \{(x, y, z) \in \mathbb{R}^3 : z > 0\}$. Note that we can regard the normal vector of any non-vertical triangle in $\mathbb{R}^3$ as a point on $\mathbb{S}^2_+$.

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.) Note 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 ϵ 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 ϵ. 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 1.1 *Given a graph G as above having n vertices, and a real constant $\epsilon > 0$, 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 2ϵ, then it is clear that (p, q) can be ignored when solving Problem 1.1. 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 2ϵ.

1.1 Motivation

The problem considered in this paper 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.

The images we used are taken from fracture surfaces of metals. Knowledge of fracture surface topographies of materials can be used to interpret the mechanical properties and the fracture mechanism of those materials. This knowledge can help to detect the weak points in a material's micro-structure. These relationships can be 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.

The images are given as 512×512 arrays of pixels, where the value stored at each entry is equal to the height of the corresponding pixel. 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. In Section 4, we will see how we converted the array of pixels to a TIN.

1.2 Our results

In Section 2, we will show how computational geometry and dynamic graph algorithms can be used to solve Problem 1.1 in $O(n^2 \log n (\log\log n)^3)$ time.

It is unlikely that Problem 1.1 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, we describe a simple grid-based heuristic. We have implemented this heuristic so that it can be used through the World Wide Web. We give some details about this implementation in Section 4. In Section 5, we 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 ϵ-planar regions whose area is sufficiently large.

1.3 Related work

The problem considered in this paper 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, the main reference we are aware of that considers this problem from a complexity point of view is Agarwal and Suri [2]. 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.

If we consider Problem 1.1 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 [4] 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 Gajentaan and Overmars [6]. This indicates that it is probably very difficult to solve the disk placement problem in subquadratic time. Recently, Agarwal *et al.* [1] gave an alternative $O(n^2)$-time algorithm for the optimal disk placement problem, as well as approximation algorithms whose running times are close to linear. Our algorithm in Section 2 for solving Problem 1.1 has been inspired by the $O(n^2)$-time algorithm in [1].

2 Solving Problem 1.1

In this section, we give an algorithm that solves Problem 1.1. 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 2.1 *To solve Problem 1.1, 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 1.1.

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, and for each one compute its weight. The maximum weight of any of these connected components gives the value of W_x.
- Set $W := \max(W, W_x)$.

Step 3: Return W.

It is clear that this algorithm correctly solves Problem 1.1. Let us analyze its running time. Recall that n denotes the number of elements of the point set S. Since each vertex of the graph G has a degree of at most three, G has at most $3n/2$ edges.

The arrangement $\mathcal{A}$ can be computed in $O(n^2)$ time using, e.g., the algorithm of Amato *et al.* [3]. Consider any vertex x of $\mathcal{A}$. The graph G_x, its connected components, and the value W_x can be computed in $O(n)$ time. Since $\mathcal{A}$ has $O(n^2)$ vertices, Step 2 takes $O(n^3)$ time. Hence, the entire algorithm takes $O(n^3)$ 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 one vertex x to a neighboring vertex y. At this moment, we know the connected components of the graph G_x, and want to compute the connected components of G_y as fast as possible. 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.

We assume that we have a data structure CC that stores the connected components, together with their weights, of a 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, and for each one compute its weight. 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 to a neighboring vertex z. At this moment, 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 vertex of $\mathcal{A}$. Since this arrangement has $O(n^2)$ vertices, 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 [9], 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.* [7].) 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 2.2 *Problem 1.1 can be solved in $O(n^2 \log n(\log\log n)^3)$ time.*

In Section 1.3, we argued that it is unlikely that Problem 1.1 can be solved in subquadratic time. Instead, one can ask about the time complexity for approximating the optimal solution. That is, let δ be a real number such that $0 < \delta < 1$, and let $x \in \mathbb{S}^2_+$ be a point for which W_x is maximum. Our goal is to compute a point $y \in \mathbb{S}^2_+$ such that $W_y \geq (1-\delta)W_x$.

Consider the following example. Let D be a spherical disk of radius ϵ, 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. This graph contains more vertices and edges, but all these vertices are "far" away from p and q, and they 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 1.1. 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 1.1 must return the center of D. Because of this, we believe that it is difficult to solve the approximation version of Problem 1.1 in subquadratic time.

3 A heuristic for finding large ϵ-planar regions

In this section, we give a simple heuristic approach to compute large connected regions in a TIN that are ϵ-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 $\epsilon > 0$, let S be the set of normal vectors of these triangles, and let $G = (S, E)$ be the graph as defined in Section 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 ϵ-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 ϵ. To be more precise, we choose an appropriate real number δ with $\delta = \Theta(\epsilon)$ and use $X := \lceil \pi/\delta \rceil$ equally spaced lines of longitude lo_i, $0 \leq i < X$, and $Y := \lceil \pi/(2\delta) \rceil$ 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 ϵ and A, and the graph $G = (S, E)$. It starts by computing a subset of all ϵ-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.)

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 $C[i_p, j_p]$.

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.

- Compute the weights of the connected components of the graph G_{ij} having the point list of $C[i, j]$ as vertex set and the edge list of $C[i, j]$ as edge set.
- For each connected component of G_{ij}, add it to the list L if its weight is at least $A/4$.

Step 6: Return the list L.

Consider the list L that is returned in Step 6. 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 ϵ-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 ϵ-planar. Below, we describe a "boundary correction" step that does exactly this.

Step 7 (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.* [5]. Let c be the center of D', and let D be the spherical disk of radius ϵ 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 8: After having completed the boundary correction step for each element in L, we have a collection of ϵ-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 ϵ-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

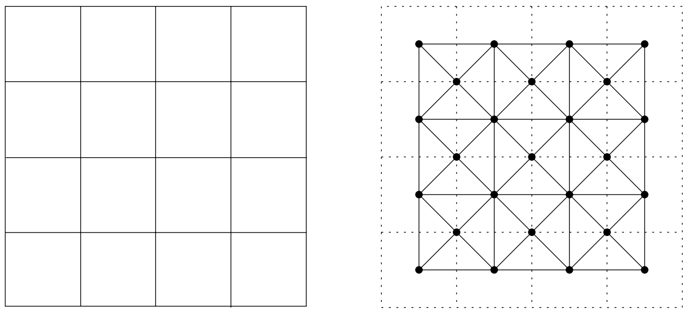


Fig. 1. *Converting an array of pixels to a TIN.*

subgraph G'' of weight at least $A/4$ that is completely contained in one of our grid cells. We can improve the chances of finding R by running the algorithm several times using shifted copies of the grid.

4 A web-based implementation

In this section, we discuss our implementation of the heuristic of the previous section.

The program, which we call PLANE_FINDER, 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.

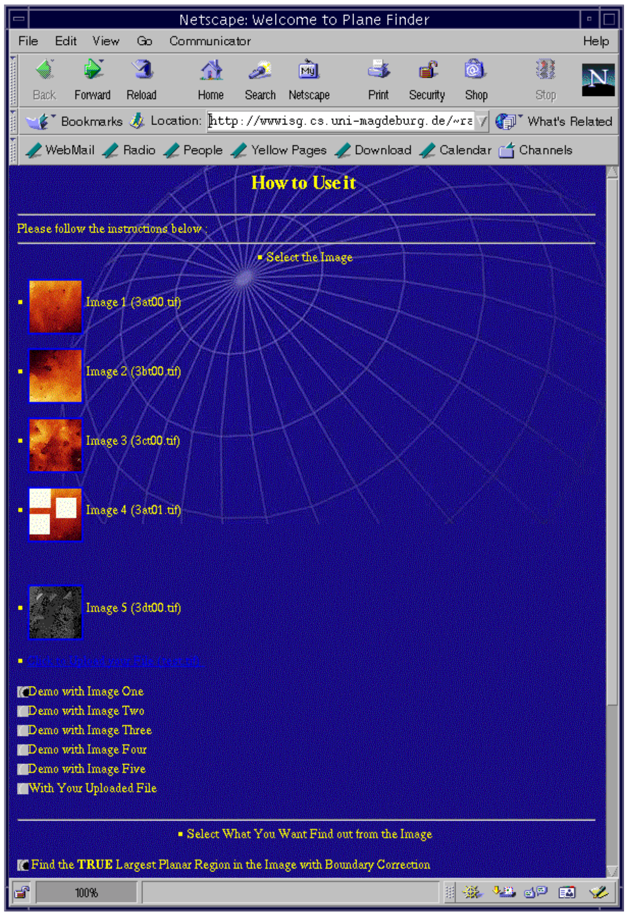
PLANE_FINDER starts by computing a TIN as illustrated in Figure 1. The vertices of the TIN are (i) the centers of the pixels, where the z-coordinate is given by the height of the pixel, and (ii) the centers of all 2×2 blocks of pixels, where the z-coordinate is given by the average height of the four pixels comprising the block. These vertices are joined by edges as indicated in the right part of Figure 1. Note that the total number of triangles, which we denote by n, is equal to $n = 4(k-1)(\ell-1)$. Given this TIN, PLANE_FINDER proceeds as described in Section 3.

The program is written in C++ and uses the LEDA library [8]. We have added a web interface to PLANE_FINDER so that a user can upload tif-images and run the program from any platform with Netscape 4.x or higher. This interface is written in Common Gateway Interface script, which runs on our web server, a Sun Ultra Sparc machine. Figure 2 shows a screen shot of the web page, see also

`http://wwwisg.cs.uni-magdeburg.de/~rahul/proj.html`

Currently, the following operations are supported.

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


Netscape: Welcome to Plane Finder
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://wwwisg.cs.uni-magdeburg.de/~ra
What's Related
WebMail Radio People Yellow Pages Download Calendar Channels
100%

How to Use it

Please follow the instructions below:

- Select the Image

- Image 1 (3at00.tif)
- Image 2 (3bt00.tif)
- Image 3 (3ct00.tif)
- Image 4 (3at01.tif)
- Image 5 (3dt00.tif)

Demo with Image One
Demo with Image Two
Demo with Image Three
Demo with Image Four
Demo with Image Five
With Your Uploaded File

- Select What You Want Find out from the Image

Find the **TRUE** Largest Planar Region in the Image with Boundary Correction

Fig. 2. *Screen shot of the PLANE_FINDER web page.*

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

5 Experimental results

We have run PLANE_FINDER on various images of fractured surfaces obtained by confocal laser scanning microscopy. These images are in tif format and consist of 512×512 pixels. Hence, the corresponding TINs consist of $n = 1,044,484$ triangles. Most of these surfaces are very rough and contain only small ϵ-planar regions. For a typical image PLANE_FINDER takes about 35 seconds on our web server. When the program is used through the web, the time for uploading and sending back the output has to be added. For the authors, it takes about 65 seconds to process a query from their computers at home in Magdeburg.

An example is given in Figure 3, which shows the original image (on the left) as well as the output of PLANE_FINDER (on the right). The program was run with ϵ equal to five degrees and A equal to 1500 square units. The

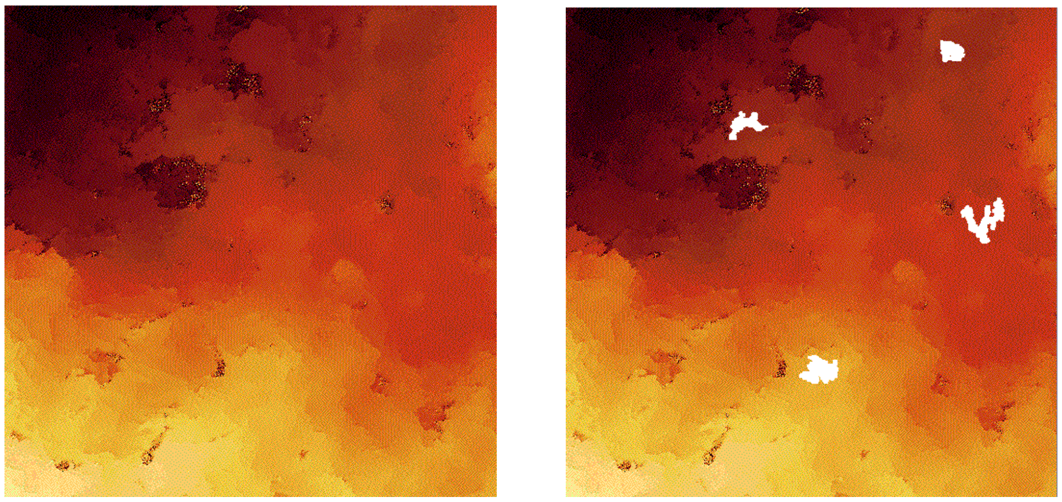


Fig. 3. *On the left, a confocal laser scanning micrography (topographical image) of a steel fracture surface is displayed. The colors represent the heights of the pixels. The four ϵ-planar regions computed by PLANE_FINDER are marked white in the right image. The image size is 100μm$\times$100μm.*

right part shows, in white, the four ϵ-planar regions found having area at least $A = 1500$ square units. Here, area refers to area in space and not the projected area.

Figure 4 shows an example of an image in which we have added four large slanted triangles. One of these is ϵ-planar for $\epsilon = 5$ degrees, whereas the others are ϵ'-planar for a value of ϵ' that is slightly larger than five degrees. PLANE_FINDER was run on this image with $A = 3000$ square units.

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.

In this paper, 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 ϵ, or the set of triangles to be contained between two parallel planes having distance ϵ.

Our software is in a preliminary stage and has a lot of scope for improvements. We plan to improve the plane detection process by applying a mean image filter in a preprocessing step. This filter will flatten local roughness.

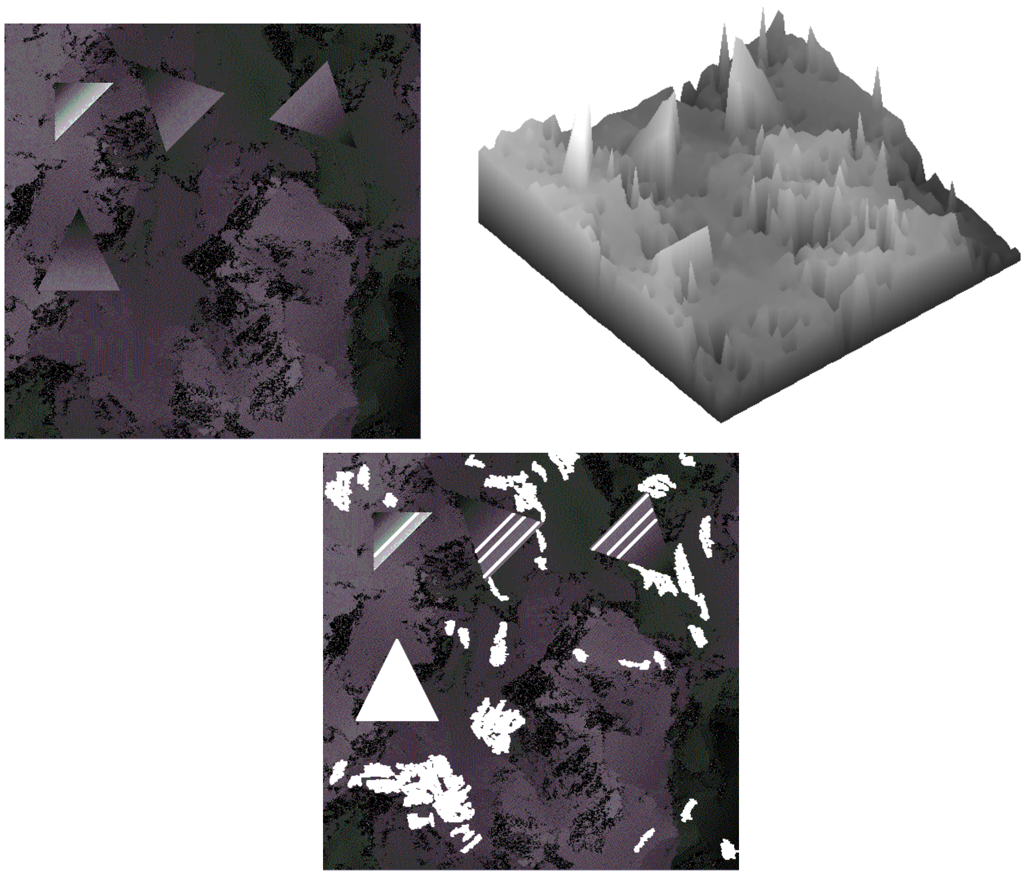


Fig. 4. *On the top, a topographical image and its parallel projection onto the xy-plane are displayed. One "approximately planar" triangle and three "not so approximately planar" triangles have been inserted for testing purposes. The ϵ-planar regions computed by PLANE_FINDER are marked white in the bottom image.*

Currently, we are working on memory optimization. Furthermore, we will add various features to improve the user interaction. In a future version of PLANE_FINDER, adjacent ϵ-planar regions will be drawn in visibly distinct colors and provide the user with more information about the planar areas. For example, by clicking on a region, information such as the normal vector and area of the region will be displayed. Also, we will add a function that provides a three-dimensional view of the image with the ability to rotate it.

References

[1] P. K. Agarwal, T. Hagerup, R. Ray, M. Sharir, M. Smid, and E. Welzl. Translating a planar object to maximize point containment: exact and approximation algorithms. Manuscript, 2001.

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

[3] 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.

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

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

[6] 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.

[7] 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.

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

[9] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd Annu. ACM Sympos. Theory Comput.*, 2000.