Carlos Seara.


Institute of Informatics, Warsaw University


Interdisciplinary Centre for Mathematical and Computational Modelling

# 18th European Workshop on Computational Geometry

April 10-12, 2002
Warsaw University
Poland

Faculty of Mathematics, Informatics and Mechanics of Warsaw University
Interdisciplinary Centre for Mathematical and Computational Modelling
Stefan Banach Centre of Excellence
Fundation for Information Technology Development

# Proceedings of the Eighteenth European Workshop on Computational Geometry

April 10-12, 2002
Institute of Informatics
Warsaw University
Poland

# Foreword

The papers in this collection were presented at the Eighteenth European Workshop on Computational Geometry, held in Warsaw, Poland, April 10-12, 2002. The workshop was organized at the Faculty of the Mathematics, Informatics and Mechanics of the Warsaw University in cooperation with the Interdisciplinary Centre for Mathematical and Computational Modelling, the Stefan Banach Centre of Excellence and the Foundation for Information Technology Development. The assistance of the Department of Computer Science of the University of Kentucky is also acknowledged.

The goal of this annual workshop is to bring together the researchers and students from academia and industry interested in Computational Geometry and related fields and to promote - in a relaxed and informal atmosphere - discussion and diffusion of the most recent work, leading to the establishment of new collaborations and research projects.

Following the tradition of the workshop, many of the included papers represent reports of continuing research, and it is expected that most of them will appear in their complete and final versions in scientific journals. Selected papers from the workshop will appear in a special issue of the "Computational Geometry: Theory and Applications". We thank everyone who responded to the Call for Papers and wish to thank Drs. Herbert Edelsbrunner, Zbigniew Marciniak, and Gert Vegter for giving invited plenary lectures.

The organizers are very grateful to the sponsoring institutions and the numerous individuals who helped in organizing and running the workshop. The experience and advice of the chairs of the previous meetings of the European Workshop on Computational Geometry, Dr. Helmut Alt (17th EWCG, Berlin, Germany), Dr. Matt Katz and Dr. Klara Kedem (16th EWCG, Eilat, Israel), were extremely useful to our work.

Jerzy W. Jaromczyk
Mirosław Kowaluk

April 2002, Warszawa

# Table of Contents

## Preliminary Program

## Invited Lectures



## Contributions

# Preliminary Program

## Wednesday, April 10

8:00-8:50 Coffee - Registration

8:50-9:00 Opening

9:00-9:20 V. Koltun, M. Sharir
*Three Dimensional Euclidean Voronoi Diagrams of Lines with a Fixed Number of Orientations*

9:20-9:40 F. Anton, J.-D. Boissonnat, D. Mioc, M. Yvinec
*An exact predicate for the optimal construction of the Additively Weighted Voronoi diagram*

9:40-10:00 F. Hurtado, V. Sacristan, R. Klein, E. Langetepe
*The weighted farthest color Voronoi diagram on trees and graphs*

10:00-10:15 Coffee break

10:15-10:35 B. Speckmann, C.D. Tóth
*Vertex $\pi$-guards in Simple Polygons*

10:35-10:55 A. Spillner, H.-D. Hecker
*The Orthogonal Fortrees Problem with Floodlights*

10:55-11:15 B.K. Nielsen, P. Winter, M. Zachariasen
*Rectilinear Trees under Rotation and Related Problems*

11:15-11:30 Coffee break

11:30-11:50 M. Hoffmann, C.D. Tóth
*Alternating Paths through Disjoint Line Segments*

11:50-12:10 S. Hornus, C. Puech
*A simple kinetic visibility polygon*

12:10-12:30 J.W. Jaromczyk, M. Kowaluk
*A kinetic view of the shooter problem*

12:30-14:30 Lunch break

14:30-15:30 Invited talk:
Herbert Edelsbrunner, Duke University
*Bio-Geometric Modeling*

15:30-15:45 Coffee break

15:45-16:05 M. Andersson, J. Gudmundsson, Ch. Levcopoulos, G. Narasimhan
*Balanced Partition of Minimum Spanning Trees*

16:05-16:25 J. Gudmundsson, H. Haverkort, S.M. Park, Ch.-S. Shin, A. Wolff
*Approximating the Geometric Minimum-Diameter Spanning Tree*

16:25-16:45 A. Bagheri, M. Razzazi
*Drawing Free Trees Inside Rectilinear polygons Using Polygon Skeleton*

17:00-18:30 Business meeting and open problem session

## Thursday, April 11

9:00- 9:20 J.M. Diaz-Báñez, F. Hurtado, M.A. López, J.A. Sellarès
*Optimal Projections onto Grids*

9:20- 9:40 E. Ezra, D. Halperin, M. Sharir
*Efficient Construction of the Union of Geometric Objects*

9:40-10:00 L. Heinrich-Litan
*Exact $L_\infty$ Nearest Neighbor Search in High Dimensions*

10:00-10:15 Coffee break

10:15-10:35 C. Cortés, A. Márquez, J. Valenzuela
*Euclidean position in 2-orbifolds*

10:35-10:55 E. Colin de Verdiere, F. Lazarus
*Optimal Polygonal Schema on an Orientable Surface*

10:55-11:15 N. de Castro
*Grid representations of graphs on surfaces*

11:15-11:30 Coffee break

11:30-11:50 C. O'Dúnlaing
*Flat embeddings of certain 2-complexes*

11:50-12:10 L. Alboul, R. van Damme
*On Flips in Polyhedral Surfaces: a new development*

12:10-12:30 P. Brass, Ch. Knauer
*Computing the symmetries of non-convex polyhedral objects in 3-space*

12:30-14:30 Lunch break

14:30-15:30 Invited talk:
Zbigniew Marciniak, Warsaw University
*Units in Group Rings of Crystallographic Groups*

15:30-15:45 Coffee break

15:45-16:05 O. Devillers, F. Hurtado, C. Seara
*Chromatic Variants of the Erdös-Szekeres Theorem on Points in Convex Position*

16:05-16:25 O. Aichholzer, F. Aurenhammer, H. Krasser
*On the Crossing Number of Complete Graphs*

16:25-16:45 J. Caceres, C.I. Grima, A. Márquez, A. Moreno-González
*Planar graphs and metrically complete graphs*

18:30 Banquet

## Friday, April 12

9:00-10:00 Invited talk:
Gert Vegter, University of Gronningen
*Evolution of apparent contour*

10:00-10:15 Coffee break

10:15-10:35 M. de Berg, P. Bose, O. Cheong, P. Morin
*On Simplifing Dot Maps*

10:35-10:55 H. Shaul, D. Halperin
*Improved Output-Sensitive Construction of the Vertical Decomposition of Three-Dimensional Arrangements*

10:55-11:15 G. Zwoźniak
*A better approximation algorithm for covering polygons with squares*

11:15-11:30 Coffee break

11:30-11:50 N. Kruithof, G. Vegter
*Approximation by skin curves*

11:50-12:10 A.M. Lyakh
*An Application of the Free Form Deformation to Phytoplankton Cells Modeling*

12:10-12:30 L. Alboul, A. Netchaev
*Isomorphic-free generation of some classes of triangulations without repetitions*

14:00 Sightseeing tour

# Invited Lectures

# Bio-Geometric Modeling

Herbert Edelsbrunner
Computer Science and Mathematics,
Duke University and Raindrop Geomagic,
Research Triangle Park, North Carolina, USA
edels@cs.duke.edu

Molecules have forever been modeled geometrically, either as stick-diagrams, emphasizing the covalent bonds between atoms, or as space-filling diagrams, representing the space they occupy. This talk aims at further developing the geometric view of the molecular world. It introduces the filtration of alpha complexes, which are combinatorial objects dual to the space filling diagrams obtained by continuously growing the atom balls. These combinatorial objects lead to fast and robust algorithms for visualization and analysis.

We demonstrate that each space filling diagram is connected the same way as its dual complex, meaning the two are homotopy equivalent. We can therefore express the connectivity of the space filling diagram by the homology groups of the dual complex. We also introduce persistent homology groups to capture the scale-dependence of a topological feature.

We show that the dual complex of a space filling diagram can be used to compute the volume and surface area without constructing the diagram. Similarly, it can be used to compute the weighted area derivative of the surface, which is believed to have a significant contribution to the force that drives the folding process simulated by molecular dynamics.

# Units in Group Rings of Crystallographic Groups

Zbigniew Marciniak
Institute of Mathematics,
Warsaw University, Poland
zbimar@mimuw.edu.pl

A group ring $RG$ is made up of two ingredients: a group $G$ and a commutative ring $R$. It consists of finite formal sums $\sum r_g g$ with $r_g \in R$ and $g \in G$. We add such sums in an obvious way; to multiply, we just open the brackets and multiply the monomials $r_g g$ and $s_h h$ by the rule $(r_g s_h)(gh)$.

Group rings play a very important role in representation theory of groups, algebraic number theory and topology. One of the important aspects of studying group rings is the description of its invertible elements (units), i.e., elements $a, b \in RG$ such that $ab = ba = 1$.

In particular, algebraists try to describe the units of $\mathbb{Z}G$ for crystallographic groups $G$. These groups are classical objects, since long very important in mathematics, physics and, of course, in crystallography. They are defined as discrete, cocompact subgroups of the group of isometries of the Euclidean space.

In the talk we shall outline a method of study of units in $\mathbb{Z}G$, using a computational approach. In particular, we shall present some achievements from the past as well as a couple of open problems in this area which could be attacked by constructing smart algorithms.

# Evolution of apparent contours

Gert Vegter
Department of Mathematics and Computing Science,
University of Groningen, The Netherlands
gert@cs.rug.nl

The *contour generator* of a smooth surface, under parallel or perspective projection, locally separates backward and forward facing regions on the surface. It consists of those points on the surface where the normal is perpendicular to the direction of projection. The *apparent contour* is the projection of the contour generator onto the image plane. See Figure 1. The contour generator and the apparent con-

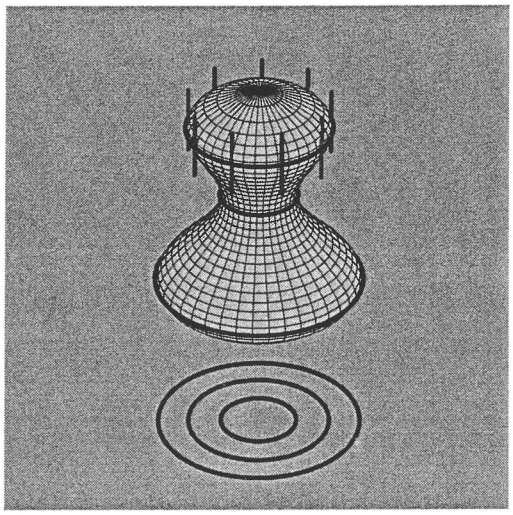


Figure 1: The contour generator and the apparent contour. The red segments correspond to tangent lines of the surface, parallel to the direction of projection. The locus of the points of tangency is the contour generator, its projection onto the view plane is the apparent contour.

tour play are important visibility features of a smooth surface, e.g., in connection with visualization. Koenderink [Koe90] describes in a very intuitive style the geometric features of curves and surfaces relevant for geometric computing. The silhouette of a surface is the curve in the view plane that separates the projected surface from the background. It is a subset of the apparent contour, that plays a prominent role in non-photorealistic rendering, cf. [BH98] and silhouette clipping [SGG+00]. In computer vision [CG00] techniques have been developed for partial reconstruction of the geometry of a surface from a sequence of apparent contours corresponding to a discrete set of projection directions.

Generically, the apparent contour is a smooth curve consisting of several components, and the contour generator consists of regular points and isolated cusp points, like the sharp beak-shaped points in the right-most column of Figure 2. Moreover, the apparent contour may have self-intersections, which correspond to distinct points on the contour generator being projected to the same point on the apparent contour.

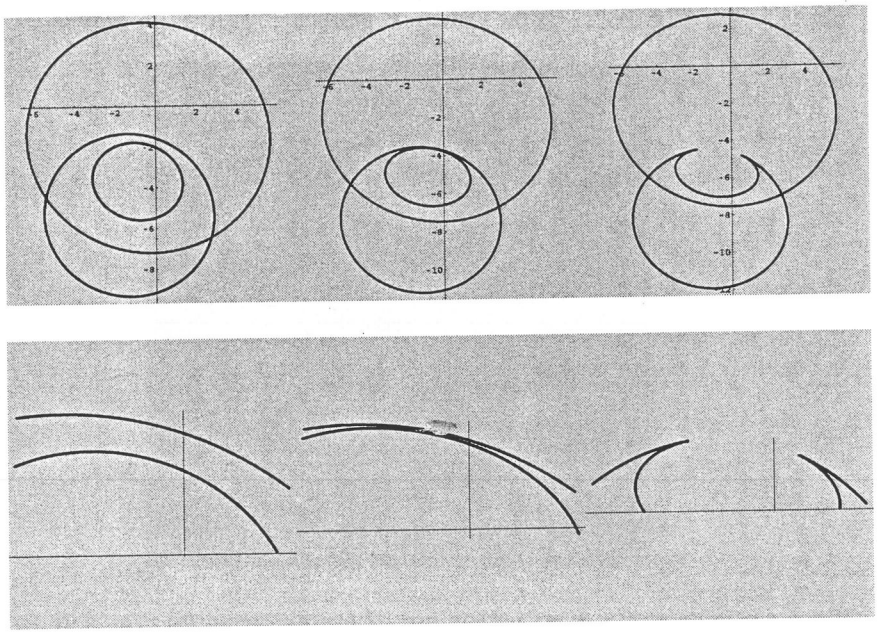


Figure 2: A beak-to-beak bifurcation. Top row: a sequence of views of a smooth surface. Middle row: the corresponding apparent contours. Bottom row: blow up of the apparent contour near the bifurcation event.

The connected components of the contour generator may merge or split when the projection direction changes, or when we rotate or deform the surface. Such *visual events*, or singularities, correspond to non-generic views of the surface, but are unavoidable if the position or shape of the surface changes continuously. Other types of visual events are related to the birth or death of components of the apparent contour.

Algorithms computing the apparent contour are reported to crash if the the evolution of the surface gives rise to the occurrence of these visual events. Therefore, a better understanding of these events is a first step towards a more robust computation of apparent contours. The classification of the various types of visual events, as well as the derivation of simple local models of the surface exhibiting these events, has been achieved using sophisticated methods from singularity theory and differential geometry. In particular, these mathematical methods do not transfer directly to robust numerical computation. An accessible description of the singularity theory behind this classification is contained in [Bru84].

In this talk we review an algorithm for the computation of the apparent contour of implicit surfaces, and describe its generic evolution as the direction of projection changes over time. In particular, we obtain approximate local models describing the generic visual events of evolving surfaces. Furthermore, we discuss how to detect the occurrence of these events for evolving implicit surfaces. Details are described in a forthcoming technical report [SV02].

## References


[BH98] D.J. Bremer and J.F. Hughes. Rapid approximate silhouette rendering of implicit surfaces. In *Proceedings of Implicit Surfaces '98*, pages 155–164, 1998.

[Bru84] J.W. Bruce. Seeing - the mathematical viewpoint. *The Mathematical Intelligencer*, 6:18–25, 1984.

[CG00] R. Cipolla and P.J. Giblin. *Visual Motion of Curves and Surfaces.* Cambridge University Press, 2000.

[Koe90] J.J. Koenderink. *Solid Shape.* Artificial Intelligence. MIT-Press, Cambridge, Massachusetts, 1990.

[SGG+00] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, and J. Snyder. Silhouette clipping. In *Computer Graphics, Proceedings of SIGGRAPH'2000*, pages 327–334, 2000.

[SV02] M. Szafraniec and G. Vegter. Computing evolving contours. Technical Report (In preparation), Groningen University, 2002.

# Contributions

# Three Dimensional Euclidean Voronoi Diagrams of Lines with a Fixed Number of Orientations*

Vladlen Koltun
School of Computer Science,
Tel Aviv University,
Tel Aviv 69978, Israel
vladlen@tau.ac.il

Micha Sharir
School of Computer Science,
Tel Aviv University,
Tel Aviv 69978, Israel;
Courant Institute of Mathematical Sciences,
New York University,
New York, NY 10012, USA.
sharir@cs.tau.ac.il

## ABSTRACT

We show that the combinatorial complexity of the Euclidean Voronoi diagram of $n$ lines in $\mathbb{R}^3$ that have at most $c$ distinct orientations, is $O(c^4 n^{2+\varepsilon})$, for any $\varepsilon > 0$. This result is a step towards proving the long-standing conjecture that the Euclidean Voronoi diagram of lines in three dimensions has near-quadratic complexity. It provides the first natural instance in which this conjecture is shown to hold. In a broader context, our result adds a natural instance to the (rather small) pool of instances of general 3-dimensional Voronoi diagrams for which near-quadratic complexity bounds are known.

## 1. INTRODUCTION

The study of Voronoi diagrams in the plane has been very extensive over the past 20 years, and the structure of such diagrams is by now thoroughly understood. The study has covered diagrams for many kinds of sites, and for many kinds of metrics or distance functions, and has also considered other variants of the problem, such as $k$-th order diagrams, constrained Delaunay triangulations, and more. Surveys of the state of the art are given in [3, 6].

In contrast, Voronoi diagrams in three and higher dimensions have been much less studied, and many basic problems are still wide open. Most variants of planar Voronoi diagrams have linear complexity, which is usually a consequence of the planarity of the diagram. In three dimensions, a prevailing conjecture is that the complexity of Voronoi diagrams should be in general at most quadratic or near-quadratic in the number of sites. This is known to hold only for very few special cases, including the cases of point sites under the Euclidean metric [3], of point sites under any 'polyhedral' metric or distance function (i.e., distance functions induced by a convex polytope with $O(1)$ facets; see [4, 7, 12] for details), and of line sites under similar distance functions [4]. Only very recently, Koltun and Sharir [9] have shown this to hold also for arbitrary polyhedral sites and polyhedral distance functions.

In all the other, 'open' cases, cubic or near-cubic upper bounds for the complexity of 3-dimensional Voronoi diagrams are known. They are a consequence of the representation of such diagrams as lower envelopes of trivariate functions, each measuring the distance from a point in $\mathbb{R}^3$ to one of the sites; see [5] for this representation, and [11] for the bounds just stated. In contrast, only quadratic or near-quadratic lower bounds for the complexity of 3-dimensional diagrams are known [2, 4].

The case of the Euclidean metric appears to be harder than the case of polyhedral metrics (or distance functions), because the surfaces that arise in this case are curved (except for the special case of point sites), and the constraints that define the diagram are harder to analyze. The simplest open case of 3-dimensional Euclidean diagrams is that where the sites are lines. This specific problem is listed as Problem 3 in the list of open problems in computational geometry, recently published by Mitchell and O'Rourke [10]. A recent result that substantiates the conjecture that the complexity of such diagrams is near-quadratic, is due to Agarwal and Sharir [1], who showed that the complexity of the union of $n$ infinite congruent cylinders in 3-space is near-quadratic. The boundary of this union can be interpreted as a cross-section of the Euclidean Voronoi diagram of the axes of the cylinders, being the locus of all those points whose distance to the nearest axis has a fixed value (equal to the common radius). The complicated proof in [1] suggests that the problem involving the whole diagram is indeed likely to be much harder to tackle.

In this paper we obtain the first result towards this goal. We study the special case where the sites are lines that have only a constant number $c$ of distinct orientations (and the metric is Euclidean). Even this special case is quite nontriv-

*Work on this paper has been supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing). Work by Micha Sharir was also supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israel Binational Science Foundation, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

ial to analyze. We show that the complexity of the diagram is $O(c^4 n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on $\varepsilon$.

We first study, in Section 2, the simpler case where the lines have at most three distinct orientations. In this special case, we obtain the slightly improved bound $O(n\lambda_5(n)) = O(n^2 \cdot \alpha(n)^{O(\alpha(n))})$, where $\lambda_5(n)$ is the maximum length of Davenport-Schinzel sequences of order 5 on $n$ symbols, and where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function. The case of four orientations is treated in Section 3, and Section 4 is devoted to the general case of an arbitrary (but constant) number of orientations.

Due to lack of space, we omit the more involved parts of our analysis from this extended abstract. A more complete version of this paper can be downloaded from the first author's home page [8].

## 2. TWO OR THREE ORIENTATIONS

Let $L$ be a set of $n$ lines in 3-space, which have up to three distinct orientations. Thus $L$ can be written as $R \cup B \cup G$, where all the lines in $R$ (called 'red' lines) have the same orientation, and the same holds for the lines of $B$ ('blue' lines) and those of $G$ ('green' lines).

We begin by bounding the number of vertices of the Voronoi diagram $Vor(L)$ of $L$. Let $v$ be a vertex of the diagram, incident to the cells of four lines $\ell_1, \ell_2, \ell_3, \ell_4$. At least two of them must be of the same color. Suppose first that three of them are of the same color, say, $\ell_1, \ell_2, \ell_3 \in R$. Project the lines of $R$, and $v$, onto a plane $\pi$ orthogonal to the lines of $R$. Then each line of $R$ projects to a point, and $v$ projects onto a vertex $v^*$ of the planar Voronoi diagram of the projected points within $\pi$. The number of such vertex projections $v^*$ is thus at most $2n - 4$. The number of vertices $v$ that can project onto the same point $v^*$ is at most $2n$. This is because the radius of the ball centered at $v$ and touching $\ell_1, \ell_2, \ell_3$ is equal to the radius of the disk (within $\pi$) centered at $v^*$ and touching the point projections of these three lines. As we slide this ball parallel to the lines of $R$, we reach at most $2n$ placements where it touches another line, giving rise to the Voronoi vertices of the kind under consideration. The overall number of such vertices is thus $O(n^2)$.

Suppose then that exactly two of the four lines are of the same color, say $\ell_1, \ell_2 \in R$, and $\ell_3, \ell_4 \in B \cup G$. If we project the lines of $R$, and $v$, onto the same plane $\pi$ as above, we obtain that the projection of $v$ lies on a Voronoi edge of the planar diagram of the point projections of the red lines. The number of such edges is $O(n)$.

Fix such an edge $e$, and consider the 2-dimensional slab $\Sigma_e$, defined as the union of all lines incident to $e$ and perpendicular to $\pi$; by construction, $v \in \Sigma_e$. Moreover, $\Sigma_e$ is the locus of all centers of balls that touch $\ell_1$ and $\ell_2$, and no other red line. Let $H_e$ denote the plane containing $\Sigma_e$, and let $\ell_0$ be the line of intersection between $H_e$ and the plane $\pi_0$ spanned by $\ell_1$ and $\ell_2$—it is the midline of the 2-dimensional slab spanned by $\ell_1$ and $\ell_2$. Denote the two halfspaces bounded by $\pi_0$ as $\pi_0^+$, $\pi_0^-$. See Figure 1.

Fix a point $p \in \ell_0$, and consider the line $\lambda_p$ that passes through $p$, lies in $H_e$, and is orthogonal to $\ell_0$. Parametrize $\lambda_p$ by a real parameter $y$, where $y = 0$ at $p$, $y > 0$ within $\pi_0^+$, and $y < 0$ within $\pi_0^-$. Move a point $q$ along the entire $\lambda_p$, in the direction of increasing $y$. The ball centered at $q$ and touching $\ell_1, \ell_2$ has the property that its intersection with $\pi_0^+$ keeps expanding during the motion (i.e., any point

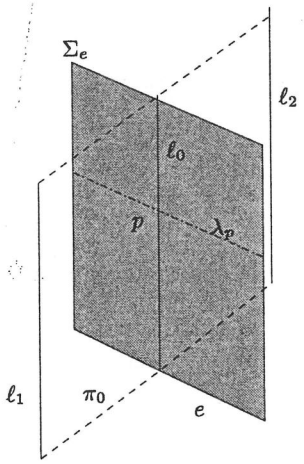



**Figure 1: The bisector of $\ell_1$ and $\ell_2$.**

of $\pi_0^+$ that the moving ball meets will remain inside the ball as its center keeps moving in the above direction). Similarly, the portion of the moving ball within $\pi_0^-$ keeps shrinking.

Replace each line $\ell \in B \cup G$ by the two rays $\ell^+ = \ell \cap \pi_0^+$, $\ell^- = \ell \cap \pi_0^-$. With each ray $\ell^+$ (resp., $\ell^-$), associate a function $\psi_{\ell+}$ (resp., $\psi_{\ell-}$) on $\ell_0$, where $\psi_{\ell+}(p)$ (resp., $\psi_{\ell-}(p)$), for $p \in \ell_0$, is the $y$-value of the center of the ball that touches $\ell_1, \ell_2$, and $\ell^+$ (resp., $\ell^-$), such that the center lies on $\lambda_p$. The functions $\psi_{\ell+}$, $\psi_{\ell-}$ are defined (and continuous) when $\ell$ does not intersect the disk centered at $p$, lying in $\pi_0$, and touching $\ell_1$ and $\ell_2$. Hence, the (common) domain of definition of $\psi_{\ell+}$ and $\psi_{\ell-}$ is either the full line $\ell_0$, if $\ell$ does not intersect the 2-dimensional slab spanned by $\ell_1$ and $\ell_2$, or the union of two rays along $\ell_0$, otherwise.

The preceding observations imply that any Voronoi vertex $v \in \Sigma_e$ (whose two other defining lines are in $B \cup G$) corresponds either to a vertex of the lower envelope of the functions $\psi_{\ell+}$, for $\ell \in B \cup G$, or to a vertex of the upper envelope of the functions $\psi_{\ell-}$, for $\ell \in B \cup G$, or to an intersection point between the two envelopes.

It is easily seen that any pair of functions of the above kind intersect in at most four points. Indeed, any such intersection point $w$ is equidistant from $\ell_1, \ell_2$, and from two other lines $\ell_3, \ell_4 \in B \cup G$. That is, we have

$$d^2(w, \ell_1) = \Big(d^2(w, \ell_2) =\Big) d^2(w, \ell_3) = d^2(w, \ell_4).$$

The squared distance of a point $w$ from a line that passes through a point $a$ and has unit direction $u$, is

$$\|w - a\|^2 - ((w - a) \cdot u)^2,$$

which is a quadratic polynomial in the coordinates of $w$. Since $w$ lies on the plane $H_e$, we obtain a system of two quadratic equations in two variables, which has at most four solutions.[1]

We split each partially defined function into two functions, each defined over a ray. As shown, e.g., in [11, Lemma 1.8],

[1]Here and in what follows, we assume that, for each fixed orientation, the lines of $L$ with that orientation are in *general position*, in the sense that their intersections with any fixed generic plane are points in general position.

the complexity of the upper or lower envelope of continuous functions, so that each function is defined on a ray or on the whole real line, and so that each pair of them intersect in at most four points, is $O(\lambda_5(n)) = O(n \cdot \alpha(n)^{O(\alpha(n))})$ [11], where $\lambda_5(n)$ is the maximum length of Davenport-Schinzel sequences of order 5 on $n$ symbols, and where $\alpha(n)$ is the extremely slowly growing inverse Ackermann function. It is easily checked that the number of intersection points between the two envelopes is proportional to the sum of the complexities of the two envelopes. We thus conclude that the number of Voronoi vertices in (the relative interior of) $\Sigma_e$ is $O(\lambda_5(n))$. Multiplying this bound by the number $O(n)$ of edges $e$, and adding the preceding bound $O(n^2)$ on the number of vertices defined by three lines of the same color, we thus conclude that the number of vertices of $Vor(L)$ is $O(n\lambda_5(n))$.

A different argument, omitted here due to lack of space, shows that the number of edges of the diagram (including unbounded Voronoi edges that are not incident to a Voronoi vertex) is also $O(n\lambda_5(n))$. Hence, we obtain the main result of this section:

THEOREM 2.1. *The complexity of the Voronoi diagram of a set of $n$ lines with at most three distinct orientations is $O(n\lambda_5(n)) = O(n^2 \cdot \alpha(n)^{O(\alpha(n))})$.*

## 3. FOUR ORIENTATIONS

THEOREM 3.1. *The complexity of the Voronoi diagram of a set of $n$ lines with at most four distinct orientations is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on $\varepsilon$.*

The somewhat complicated proof of the above theorem, which is the heart of our analysis, is omitted due to space limitations. It relies critically on Theorem 2.1, and utilizes the technique of *counting schemes* [11]. Informally, we attempt to charge each Voronoi vertex to about $k^2$ Voronoi vertices at level at most $k$ (using a suitably chosen constant $k$, under an appropriate definition of 'level', see [11]). The number of Voronoi vertices that can be charged in this fashion can be expressed by a recurrence relation that solves to $O(n^{2+\varepsilon})$. Not all Voronoi vertices fall into this category; however, we were able to show that those vertices that cannot be charged as above, can be alternatively charged to vertices that are defined by quadruples of lines that have only three distinct orientations. Theorem 2.1 implies that the number of such vertices is $O(n\lambda_5(n))$.

## 4. MORE THAN FOUR ORIENTATIONS

The case of an arbitrary (but constant) number of orientations is easy to handle using Theorem 3.1, by noting that any vertex $v$ of the full Voronoi diagram $Vor(L)$ is also a vertex of the diagram of the set of all lines whose orientations are equal to the (at most) four orientations of the lines whose cells are incident to $v$. Hence, by applying Theorem 3.1 to each of the $O(c^4)$ quadruples of orientations, we obtain the main result of the paper:

THEOREM 4.1. *The combinatorial complexity of the Voronoi diagram of $n$ lines in three dimensions, where the lines have only $c$ distinct orientations, is $O(c^4 n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on $\varepsilon$.*


## Acknowledgements

The work on this paper was initiated while the authors were visiting ETH Zürich, hosted by the European Graduate Program on Combinatorics, Geometry and Computation. The authors are grateful to the program, and in particular to Emo Welzl, for the support.


## 5. REFERENCES


[1] P. Agarwal and M. Sharir, Pipes, cigars, and kreplach: The union of Minkowski sums in three dimensions. *Discrete Comput. Geom.* 24 (2000), 645–685.

[2] B. Aronov, A lower bound on Voronoi diagram complexity, manuscript, 2001.

[3] F. Aurenhammer and R. Klein, Voronoi diagrams, in: *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, Eds.), North-Holland, Amsterdam 2000, pp. 201–290.

[4] L.P. Chew, K. Kedem, M. Sharir, B. Tagansky and E. Welzl, Voronoi diagrams of lines in three dimensions under polyhedral convex distance functions, *J. Algorithms* 29 (1998), 238–255.

[5] H. Edelsbrunner and R. Seidel, Voronoi diagrams and arrangements, *Discrete Comput. Geom.* 1 (1986), 25–44.

[6] S. Fortune, Voronoi diagrams and Delaunay triangulations, in: *Handbook of Discrete and Computational Geometry* (J.E. Goodman and J. O'Rourke, Eds.), CRC Press, Boca Raton, 1997, pp. 377–388.

[7] C. Icking and L. Ma, A tight bound for the complexity of Voronoi diagrams under polyhedral convex distance functions in 3D, *Proc. 33rd Annu. ACM Sympos. Theory of Computing*, 2001, pp. 316–321.

[8] *http://www.cs.tau.ac.il/~vladlen*

[9] V. Koltun and M. Sharir, Polyhedral Voronoi diagrams of polyhedra in three dimensions, manuscript, 2001.

[10] J.S.B. Mitchell and J. O'Rourke, Computational geometry Column 42, *SIGACT News* 32(3), September 2001, 63–72.

[11] M. Sharir and P.K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[12] B. Tagansky, *The Complexity of Substructures in Arrangements of Surfaces*, Ph.D. Dissertation, Computer Science Department, Tel Aviv University, 1996.

# An exact predicate for the optimal construction of the Additively Weighted Voronoi diagram

Extended Abstract

François Anton
University of British Columbia
Department of Computer Science
201-2366 Main Mall, Vancouver, B.C., Canada,
V6T 1N4
fanton@cs.ubc.ca

Jean-Daniel Boissonnat, Darka Mioc,
Mariette Yvinec
INRIA Sophia Antipolis
2004, route des Lucioles, BP 93
06902 Sophia Antipolis Cedex, France
Jean-Daniel.Boissonnat@sophia.inria.fr,
dmioc@cs.ubc.ca,
Mariette.Yvinec@sophia.inria.fr

**Keywords**
Additively Weighted Voronoi diagram, Power Voronoi diagram, algebraic predicates

## 1. INTRODUCTION

The ordinary Voronoi diagram has been extended or generalised in several directions, and these generalised Voronoi diagrams have found many practical applications (see [2] for a general survey on Voronoi diagrams). Abstract Voronoi diagrams [4] include a large number of concrete Voronoi diagrams, e.g., Voronoi diagrams for point sites under any $L_p$ -metric, $1 \leq p \leq \infty$, or under any convex distance function, whose unit circle is algebraic. Furthermore, they comprise weighted Voronoi diagrams and Voronoi diagrams for line segments or circles under the Euclidean metric.

The weighted Voronoi diagrams (additively, multiplicatively, compound, etc.) differ from the ordinary Voronoi diagram in that the generators do not all have the same weight and the distance depends on these weights. The Additively Weighted Voronoi diagram is defined using the additively weighted distance: $d_{AW}(p, p_i) = \|x - x_i\| - w_i$ , where $x$ is the position vector of $p$, $x_i$ is the position vector of the point $p_i$, and $w_i$ is the weight of the point $p_i$.

The algorithm proposed by Aurenhammer [1] is based on the relation between the Additively Weighted Voronoi diagram in dimension $d$ and the Power Voronoi diagram in dimension $d+1$.

In this work, we have developed an exact algebraic predicate and an optimal algorithm for constructing the Additively Weighted Voronoi diagram from the Power Voronoi diagram.

In the next section we review the relation between the Additively Weighted Voronoi diagram and the Power Voronoi diagram. In Section 3, we introduce the algorithm based on this relation. In Section 4, we explore the algebraic predicate that is the core of the algorithm presented in Section 3.

## 2. PRELIMINARIES

We place ourselves in the Euclidean plane $E^2$. The algorithm uses the mapping $\phi$ of the weight circle (centred on $(x_i, y_i)$ and radius $w_i$) corresponding to a weighted point $P_i$ with coordinates $(x_i, y_i)$ and weight $w_i$ to the sphere $\sum_i$ of centre $M_i$ with coordinates $(x_i, y_i, w_i)$ and radius $r_i = \sqrt{2}w_i$.

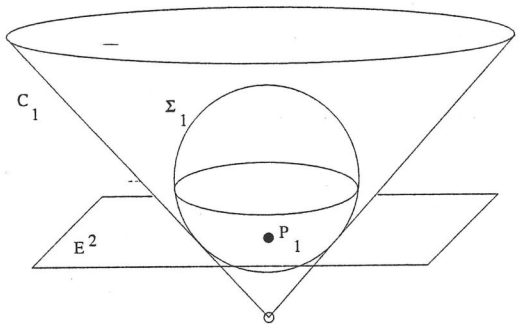



**Figure 1: The cone inscribing the sphere image of the weighted point**

The $\frac{\Pi}{4}$ vertex angle cone $C_i$ that inscribes the weighted sphere $\sum_i$ has its apex at $(x_i, y_i, -w_i)$ (see Fig. 1). The vertical projection $\pi(M)$ of a point $M$ of $E^2$ on the cone $C_i$ is the image by the map $\phi$ of the circle centered at $M$ and tangent to the weight circle associated with $P_i$. The signed distance from $M$ to $\pi(M)$ equals the additive distance from $M$ to the weighted point $P_i$ . Thus, the additively Voronoi diagram of the set $\{(P_i, w_i)\}$ is the projection onto $E^2$ of

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

the lower envelope of the set of cones $\{C_i\}$. Then, it is easy to show that the contribution of each cone $C_i$ to this lower envelope is just the intersection of the cone $C_i$ with the region of the sphere $\Sigma_i$ in the Power Voronoi diagram of the set $\{\Sigma_i\}$.

## 3. THE ALGORITHM

The algorithm consists of three steps: first compute all the spheres $\sum_i$, then compute the Power Voronoi diagram of these spheres $\sum_i$, and finally for each $i$, compute the intersection with the cone $C_i$ of the cell of the Power Voronoi diagram that corresponds to $\sum_i$ .

To compute, the intersection of the power cell of $\{\Sigma_i\}$ with the cones $C_i$, we propose to consider in turn each edge of this cell. Each edge of the Power Voronoi diagram is determined by five spheres. The algorithm is based on a predicate that determines if such an edge intersect the cone $C_i$ and the number (at most two) of intersection points if any.

The worst case running time of the algorithm is the same as the one of the Power Voronoi diagram, i.e., $O(n^2)$ (see [3] for a description of the algorithm for constructing the Power Voronoi diagram and its optimality). However, as we will see it in the next section, this algorithm requires algebraic predicates of degree 16 only.

## 4. THE ALGEBRAIC PREDICATE FOR THE CONSTRUCTION OF THE ADDITIVELY WEIGHTED VORONOI DIAGRAM

This predicate is based on the intersection of the cone inscribing a weighted sphere resulting from the mapping of a weighted point to the 3D space with an edge of its Power Voronoi cell. Each edge of the Power Voronoi cell connects two Power Voronoi vertices and involves five spheres. Each vertex is the centre of the circle orthogonal to four spheres. The centres of these spheres form a tetrahedron.

### 4.1 The Power Voronoi vertices

An edge between the vertices corresponding to two adjacent tetrahedra is determined by two adjacent vertices of the Power Voronoi cell. For the computations of the first vertex the first four weighted points are needed. In 3D space they form a tetrahedron.

Let $S_1\left((x_1,y_1,w_1),\sqrt{2}w_1\right)$, $S_2\left((x_2,y_2,w_2),\sqrt{2}w_2\right)$, $S_3\left((x_3,y_3,w_3),\sqrt{2}w_3\right)$, and $S_4\left((x_4,y_4,w_4),\sqrt{2}w_4\right)$ be the four spheres defining the tetrahedron.

The intersection of these linearly independent radical planes gives the first Power Voronoi vertex:

$$\begin{pmatrix} -2(x_1-x_2) & -2(y_1-y_2) & -2(w_1-w_2) \\ -2(x_1-x_3) & -2(y_1-y_3) & -2(w_1-w_3) \\ -2(x_1-x_4) & -2(y_1-y_4) & -2(w_1-w_4) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 \\ x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 \\ x_4^2-x_1^2+y_4^2-y_1^2+w_1^2-w_4^2 \end{pmatrix}.$$

The coordinates of the first Power Voronoi vertex are:

$$x' = \frac{\begin{vmatrix} x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 & -2(y_1-y_2) & -2(w_1-w_2) \\ x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 & -2(y_1-y_3) & -2(w_1-w_3) \\ x_4^2-x_1^2+y_4^2-y_1^2+w_1^2-w_4^2 & -2(y_1-y_4) & -2(w_1-w_4) \end{vmatrix}}{d}$$

$$y' = \frac{\begin{vmatrix} -2(x_1-x_2) & x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 & -2(w_1-w_2) \\ -2(x_1-x_3) & x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 & -2(w_1-w_3) \\ -2(x_1-x_4) & x_4^2-x_1^2+y_4^2-y_1^2+w_1^2-w_4^2 & -2(w_1-w_4) \end{vmatrix}}{d}$$

$$z' = \frac{\begin{vmatrix} -2(x_1-x_2) & -2(y_1-y_2) & x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 \\ -2(x_1-x_3) & -2(y_1-y_3) & x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 \\ -2(x_1-x_4) & -2(y_1-y_4) & x_4^2-x_1^2+y_4^2-y_1^2+w_1^2-w_4^2 \end{vmatrix}}{d}$$

where $d = \begin{vmatrix} -2(x_1-x_2) & -2(y_1-y_2) & -2(w_1-w_2) \\ -2(x_1-x_3) & -2(y_1-y_3) & -2(w_1-w_3) \\ -2(x_1-x_4) & -2(y_1-y_4) & -2(w_1-w_4) \end{vmatrix}$.

The second vertex can be computed in a similar way. For the computation of the second vertex, the three first weighted points and the fifth weighted point are needed. The coordinates of the second Power Voronoi vertex are:

$$x'' = \frac{\begin{vmatrix} x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 & -2(y_1-y_2) & -2(w_1-w_2) \\ x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 & -2(y_1-y_3) & -2(w_1-w_3) \\ x_5^2-x_1^2+y_5^2-y_1^2+w_1^2-w_5^2 & -2(y_1-y_5) & -2(w_1-w_5) \end{vmatrix}}{D}$$

$$y'' = \frac{\begin{vmatrix} -2(x_1-x_2) & x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 & -2(w_1-w_2) \\ -2(x_1-x_3) & x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 & -2(w_1-w_3) \\ -2(x_1-x_5) & x_5^2-x_1^2+y_5^2-y_1^2+w_1^2-w_5^2 & -2(w_1-w_5) \end{vmatrix}}{D}$$

$$z'' = \frac{\begin{vmatrix} -2(x_1-x_2) & -2(y_1-y_2) & x_2^2-x_1^2+y_2^2-y_1^2+w_1^2-w_2^2 \\ -2(x_1-x_3) & -2(y_1-y_3) & x_3^2-x_1^2+y_3^2-y_1^2+w_1^2-w_3^2 \\ -2(x_1-x_5) & -2(y_1-y_5) & x_5^2-x_1^2+y_5^2-y_1^2+w_1^2-w_5^2 \end{vmatrix}}{D}$$

where $D = \begin{vmatrix} -2(x_1-x_2) & -2(y_1-y_2) & -2(w_1-w_2) \\ -2(x_1-x_3) & -2(y_1-y_3) & -2(w_1-w_3) \\ -2(x_1-x_5) & -2(y_1-y_5) & -2(w_1-w_5) \end{vmatrix}$.

### 4.2 The algebraic predicate

It is possible to simplify the expressions of the coordinates of the Power Voronoi vertices by using the following submatrices:

$$M_1 = \begin{pmatrix} -2(y_1-y_2) - 2(w_1-w_2) \\ -2(y_1-y_3) - 2(w_1-w_3) \end{pmatrix}$$

$$M_2 = \begin{pmatrix} -2(x_1-x_2) - 2(w_1-w_2) \\ -2(x_1-x_3) - 2(w_1-w_3) \end{pmatrix}$$

$$M_3 = \begin{pmatrix} -2(x_1-x_2) - 2(y_1-y_2) \\ -2(x_1-x_3) - 2(y_1-y_3) \end{pmatrix}$$

$$M_4 = \begin{pmatrix} x_2^2-x_1^2+y_2^2-y_1^2-w_2^2+w_1^2-2(w_1-w_2) \\ x_3^2-x_1^2+y_3^2-y_1^2-w_3^2+w_1^2-2(w_1-w_3) \end{pmatrix}$$

$$M_5 = \begin{pmatrix} x_2^2 - x_1^2 + y_2^2 - y_1^2 - w_2^2 + w_1^2 - 2(y_1 - y_2) \\ x_3^2 - x_1^2 + y_3^2 - y_1^2 - w_3^2 + w_1^2 - 2(y_1 - y_3) \end{pmatrix}$$

$$M_6 = \begin{pmatrix} -2(x_1 - x_2)\, y_2^2 - y_1^2 + x_2^2 - x_1^2 - w_2^2 + w_1^2 \\ -2(x_1 - x_3)\, y_3^2 - y_1^2 + x_3^2 - x_1^2 - w_3^2 + w_1^2 \end{pmatrix}$$

and their determinants: $de_1 = det(M_1)$, $de_2 = det(M_2)$, $de_3 = det(M_3)$, $de_4 = det(M_4)$, $de_5 = det(M_5)$, and $de_6 = det(M_6)$.

We get: $x' = \frac{2de_4(y_1-y_4)-2de_5(w_1-w_4)+de_1k_4}{d}$,

$x'' = \frac{2de_4(y_1-y_5)-2de_5(w_1-w_5)+de_1k_5}{D}$,

$y' = \frac{-2de_4(x_1-x_4)-2de_6(w_1-w_4)-de_2k_4}{d}$,

$y'' = \frac{-2de_4(x_1-x_5)-2de_6(w_1-w_5)-de_2k_5}{D}$,

$z' = \frac{2de_5(x_1-x_4)+2de_6(y_1-y_4)+de_3k_4}{d}$,

$z'' = \frac{2de_5(x_1-x_5)+2de_6(y_1-y_5)+de_3k_5}{D}$,

where $k_4 = x_4^2 - x_1^2 + y_4^2 - y_1^2 + w_1^2 - w_4^2$,

and $k_5 = x_5^2 - x_1^2 + y_5^2 - y_1^2 + w_1^2 - w_5^2$.

The coordinates of a point on the straight line defined by the two vertices $((x', y', z'), (x'', y'', z''))$ and supporting the corresponding edge of the Power Voronoi diagram are: $(x' + \lambda(x'' - x'), y' + \lambda(y'' - y'), z' + \lambda(z'' - z'))$.

The equation in $(x, y, z)$ of the algebraic cone corresponding to the weighted point $(a, b, c)$ (where $(a, b, c)$ is one of the first three sphere centres) is: $(x - a)^2 + (y - b)^2 - (z + c)^2 = 0$.

Thus, the intersection of the straight line defined by the two Power Voronoi vertices and the cone is given by the following quadratic equation in $\lambda$:
$(x' + \lambda(x'' - x') - a)^2 + (y' + \lambda(y'' - y') - b)^2 -$
$(z' + \lambda(z'' - z') + c)^2 = 0$.

This equation can be written in the following form:

$$A\lambda^2 + 2B\lambda + C = 0 \qquad (1)$$

where $A = (x'' - x')^2 + (y'' - y')^2 - (z'' - z')^2$,
$B = (x' - a)(x'' - x') + (y' - b)(y'' - y') - (z' + c)(z'' - z')$,
$C = (x' - a)^2 + (y' - b)^2 - (z' + c)^2$,
and $\Delta = 4B^2 - 4AC = 4(B^2 - AC)$.

Let $\Delta_x = x' - x"$, $\delta_x = x' - a$, $\Delta_y = y' - y"$, $\delta_y = y' - b$, $\Delta_z = z' - z"$, and $\delta_z = z' + c$. Then, $A = \Delta_x^2 + \Delta_y^2 - \Delta_z^2$, $B = -\delta_x\Delta_x - \delta_y\Delta_y + \delta_z\Delta_z$, and $C = \delta_x^2 + \delta_y^2 - \delta_z^2$. Then, by developing and simplifying, we can rewrite the discriminant of the quadratic equation as follows: $B^2 - AC = -(\Delta_x\delta_y - \Delta_y\delta_x)^2 + (\Delta_y\delta_z - \Delta_z\delta_y)^2 + (\Delta_z\delta_x - \Delta_x\delta_z)^2$.

In the first term $(\Delta_x\delta_y - \Delta_y\delta_x)^2 = ((x' - x")(y' - b) - (y' - y")(x' - a))^2$, the terms in $x'y'$ vanish, and we get $(\Delta_x\delta_y - \Delta_y\delta_x)^2 = (-x"y' - y"x' - bx' - bx" - ay' - ay")^2$. The first term admits $d^2D^2$ as common denominator, with a numerator of degree 16. Indeed, the degree of the numerator of the first term equals the degree of the product of the square of the numerator of $x'$ by the square of the numerator of $y''$, which is the same as the degree of the product of the square of the numerator of $x''$ by the square of the numerator of $y'$. Similarly, we can prove that the second and third terms admit $d^2D^2$ as common denominator, with numerators of degree 16.

The degree of the numerator of $\Delta$ is 16. Since the common denominator is a perfect square ($(dD)^2$), the sign of $\Delta$ is the sign of its numerator. Thus, its evaluation requires the evaluation of a polynomial of degree 16 in the input (the coordinates of the sphere centres and the sphere radii).

The predicate involves determining if the Power Voronoi diagram edge intersects with the cone. This can be done by checking if the straight line supporting the edge intersects with the cone (i.e. evaluating if $\Delta$ is positive or zero), and if it does, checking if there is an intersection that belongs to the edge between the Power Voronoi vertices.

In order to determine the intersections that lie in between the two Power Voronoi vertices, we need to locate the roots of the equation (1) with respect to the $[0, 1]$ interval. If $A = 0$ or the sign of $\Delta$ is 0, then there is only one root. In the first case, determining its position with respect to 0 amounts to computing its sign (i.e. the sign of $\frac{-C}{2B}$), i.e. checking if $C$ and $B$ have the same signs. Determining its position with respect to 1 amounts to check if $\frac{-C}{2B} \leq 1$. Depending on the sign of $B$, this unequation can be rewritten as $-C - 2B \geq 0$ or $-C - 2B \leq 0$. In the second case, determining the position of the root with respect to 0 amounts to compute the sign of the common root, which is also the sign of the sum of the roots ($\frac{-B}{A}$), i.e. checking if $B$ and $A$ have the same signs. Determining the position with respect to 1 can also be done by evaluating the signs of $B$ and $A$.

In the general case of two distinct roots, the determination of presence of roots in between the Power Voronoi vertices can be done by considering first the signs of $A$ and $A + 2B + C$. The left hand side of equation 1 has the same sign as $A$ outside the roots of that equation, $C$ equals the left hand side of 1 when $\lambda = 0$, and $A + 2B + C$ equals the left hand side of 1 when $\lambda = 1$. When $C$ and $A + 2B + C$ have different signs, then there is only one root in the $[0, 1]$ interval. If their signs are identical, either both roots are inside the interval, or both roots are outside. The determination of the position of the roots with respect to the $[0, 1]$ interval can proceed by using methods similar to the ones shown above for the case of one root only. Using the same reasoning as for the discriminant, we can conclude that determining the sign of $A$ involves determining the sign of a polynomial of degree 16; determining the sign of $B$ involves determining the sign of a polynomial of degree 16, and determining the sign of $C$ involves determining the sign of a polynomial of degree 8. The denominators of $A$, $B$, and $C$ are respectively $d^2D^2$, $d^2D^2$, and $d^2$. Thus determining the sign of $A + 2B + C$ or $-C - 2B$ or $-B - A$ involves determining the sign of a polynomial of degree 16. Thus, the predicate simply amounts to the evaluation of the sign of polynomials of degree bounded by 16 (namely $A, B, C, A + 2B + C, -C - 2B, -B - A$). This evaluation can be done exactly provided the input coordi-

nates are coded using integer types.

Finally, the cone that is to be tested for the intersection with the edge of the Power Voronoi cell is the upper cone of axis parallel to $O_z$, with the apex at $(a, b, c)$ and a $\frac{\Pi}{4}$ vertex angle. The cone that we used in the equations is the algebraic cone.

To detect the intersection of the edge of the Power Voronoi cell with the upper cone, we can do a first filtering: if both Power Voronoi vertices $(x', y', z')$ and $(x'', y'', z'')$ lie above or on the plane $z = -c$, then the intersections obtained before are in the upper nape, and they are valid intersections. If both Power Voronoi vertices $(x', y', z')$ and $(x'', y'', z'')$ lie below the plane $z = -c$, then the intersections obtained before are in the lower nape, and they are not valid intersections.

If nothing is known after the first filtering, we locate the parameter $\lambda_l$ of the point where the straight line through the Voronoi vertices intersects the plane $z = -c$ with respect to the roots of the quadratic equation 1. This amounts to check if $f(\lambda_l)$ is the same as the sign of $A$. We will show that this is a degree 16 predicate. Since $z = z' + \lambda_l (z'' - z')$, and $z = -c$ define $\lambda_l$, $\lambda_l = (z' + c) / (z' - z'')$. By multiplying all the terms of $A\lambda_l^2 + 2B\lambda_l + C$ by $\Delta_z^2$, the sign of $f(\lambda_l)$ does not change. We get that the sign of $f(\lambda_l)$ is the same as the sign of $\left(\Delta_x^2 + \Delta_y^2 - \Delta_z^2\right)(z' + c)^2 + 2\left(\delta_z \Delta_z - \delta_x \Delta_x - \delta_y \Delta_y\right)(z' + c)\Delta_z + \left(\delta_x^2 + \delta_y^2 - \delta_z^2\right)\Delta_z^2$. Collecting the terms in $z'^2$ and $z'$, we get that the terms in $x'^2 z'^2$ as well as those in $y'^2 z'^2$ and those in $z'^4$ vanish. We get an expression of the form $z'^2\left[(x'' - a)^2 + ...\right] + z'[2c(x'' - x')(x'' - a) - 2z''(x' - a)(x'' - a) + ...]$, from where it follows that every term accepts a numerator of degree 16 with $d^2 D^2$ as denominator.

If $sgn(f(l)) = -sgn(A)$ then the two intersections are on different nape of the algebraic cone. Then, the only intersection that lies in the upper nape of the algebraic cone is the solution of Equation 1 which is closer to the higher Power Voronoi vertex.

If $sgn(f(l)) = sgn(A)$ then either both intersections lie in the upper nape, or both intersections lie in the lower nape. In that case, what we need to do is to compare the sum of the roots $\frac{-B}{A}$ with $\lambda_l$ which again can be shown to be a test of degree 16.

## 5. CONCLUSIONS

We have provided an exact predicate for the computation of Additively Weighted Voronoi diagrams in the plane. The degree of this predicate is 16. Even though the worst case running time of the algorithm may not be optimal if we consider algebraic computations as $O(1)$ running time computations, the algebraic complexity of this algorithm is optimal (degree 16 predicate). The algebraic predicates needed in order to construct incrementally the Additively Weighted Voronoi diagram seem to have a higher algebraic complexity. This is still an open problem to know if degree 16 algebraic predicates are sufficient to maintain incrementally the Additively Weighted Voronoi diagram.

## 6. ACKNOWLEDGMENTS

The first author research has received the financial support of the French Government (BGF) and of the Institut National de Recherche en Informatique et en Automatique (INRIA).

## 7. REFERENCES

[1] F. Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM J. Comput.*, 16(1):78–96, 1987.

[2] F. Aurenhammer and R. Klein. Voronoi diagrams. In *Handbook of computational geometry*, pages 201–290. North-Holland, Amsterdam, 2000.

[3] J.-D. Boissonnat and M. Yvinec. *Algorithmic geometry.* Cambridge University Press, Cambridge, 1998. Translated from the 1995 French original by Hervé Brönnimann.

[4] R. Klein. *Concrete and abstract Voronoï diagrams.* Springer-Verlag, Berlin, 1989.

# The weighted farthest color Voronoi diagram on trees and graphs

## [Extended Abstract]

Ferran Hurtado, Vera Sacristán
Dept. de Matemàtica Aplicada II
Univ. Politècnica de Catalunya, Barcelona, Spain

Rolf Klein, Elmar Langetepe
Institut für Informatik I
Universität Bonn, Germany

## ABSTRACT

Let $n$ point sites be situated on the vertices or edges of a geometric graph $G$ over $e$ edges. Each site can be assigned a multiplicative weight and a color. We discuss the complexity, and provide efficient algorithms for the construction, of the Voronoi diagram in which each point of $G$ belongs to the region of that site which is the closest of the furthest color. Special algorithms are presented for the cases when all colors are identical, when all weights are identical, or when $G$ is a tree.

## Keywords

Voronoi diagram, graph, network, locational planning.

## 1. INTRODUCTION

The Voronoi diagram is one of the most interesting structures in computational geometry; many variants and applications have been described in the surveys [2, 4, 9]. It decomposes a given space, $G$, into Voronoi regions, one to each element $p$ of a given site set, $S$, such that the region of site $p$ contains all points of $G$ that are closer to $p$ than to any other site in $S$. In most cases, $G$ is a space of dimension at least 2.

Some variants of the Voronoi diagram are of particular interest to applications in locational planning. First, each site $p$ can be assigned a positive *weight factor*, $c$, to model its importance. The distance from $p$ to some point $z$ is then the standard distance $|pz|$ in space $G$, times $c$. (Note that a bigger weight factor yields a smaller Voronoi region.) For the Euclidean plane, Aurenhammer and Edelsbrunner [3] have shown that the weighted Voronoi diagram of $n$ point sites can be of complexity $\Theta(n^2)$, and provided an optimal algorithm.

The second variant applies to a situation where each of the $n$ sites represents one of $k$ different types of facilities, like schools, shopping malls, hospitals, etc. A person choosing a new residence might want to have one representative of each type as close by as possible. This problem can be solved by means of the *furthest color* Voronoi diagram. Let us assume that each site is colored according to its type. For each point $z$ of the plane, let $c(i, z)$ denote the closest site of color $i$. Then, by definition, $z$ belongs to the Voronoi region of the site among $c(1, z), \ldots, c(k, z)$ that is farthest away from $z$. This distance is minimized by a circle centered at a point on the Voronoi diagram.

For the Euclidean plane, Huttenlocher et al. [7] have shown how to construct this diagram in time $O(kn \log n)$. Abellanas et al. [1] gave efficient algorithms for finding the smallest axis-parallel strip or rectangle containing a point of each color, that were not based on Voronoi diagrams.

In this paper we study both, the weighted and the colored variant, and their combination, in the one-dimensional setting, where $G$ is a geometric tree or graph. So far, only the standard (i. e. unweighted and uncolored) Voronoi diagram on graphs has received attention. In their monograph [9], Okabe et al. introduce the standard Voronoi diagram and discuss its structure. Among other papers, they cite Hakimi et al. [6] who gave an $O(en + m \log m)$ algorithm based on shortest path computations; here $e$ and $m$ denote the number of edges and vertices, respectively. If the sites are vertices of the graph, Mehlhorn [8] avoids the factor $n$ by a more efficient way of using Dijkstra's shortest path algorithm, and Erwig [5] improves on this by giving an $O(e + m + (m - n)\log(m - n))$ algorithm.

Throughout this paper, we consider a geometric tree or graph $G$ of $m$ vertices over $e$ edges, and $n$ point sites that are placed on edges or vertices of $G$. Just for simplicity we assume that $G$ is planar, and that its edges are realized by straight line segments.

## 2. WEIGHTED FARTHEST COLOR VORONOI DIAGRAM ON A TREE

Consider a tree $T = (V, E, d)$ of size $m$, where $V$ is the set of vertices, $E$ the set of edges, and $m = |V| = |E| + 1$. For any two points $x$, $y$ on the tree, $d(x, y)$ indicates the Euclidean distance from $x$ to $y$ along the unique path that connects the two points.

Given a set $S = \{p_1, \ldots, p_n\}$ of $n$ points on the tree, we define the Voronoi diagram of $S$ on the tree, $VD_T(S)$, as a

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

**Table 1: Summary of the results**

| diagram | measure | tree | graph |
|---|---|---|---|
| VD | Complexity | $n-1+m$ | $\Theta(n+e)$ |
| | Comp. Time | $\Theta(m+n\log n)$ | $O((n+e)\log n)$ |
| WVD | Complexity | $\Theta(mn)$ | $O(en)$ |
| | Comp. Time | $O(mn\log n)$ | $O(en\log n)$ |
| FCVD | Complexity | $2(n-k)+1+m$ | $O(ke+n)$ |
| | Comp. Time | $O(km+n\log n)$ | $O((ke+n)\log n)$ |
| WFCVD | Complexity | $O(mn\alpha(k))$, $\Omega(mn)$ | $O(en\alpha(k))$, $\Omega(en)$ |
| | Comp. Time | $O(mn(\log n+\alpha^2(k)\log k))$ | $O(en(\log n+\alpha^2(k)\log k))$ |

partition $\{VR_T(p_1),\ldots,VR_T(p_n)\}$ of $T$, where the Voronoi region of each site $p_i$ is the set $VR_T(p_i)$ of all the points $x$ of $T$ that are closer to $p_i$ than to any other site $p_j$, in terms of the tree distance $d$. We consider such Voronoi diagram completely described if the following information is known:

- The set of bisector points, i.e. all the boundary points between adjacent Voronoi regions.
- For each bisector point, its location in the tree (the edge to which it belongs, or the vertex it coincides with), the sites of its adjacent Voronoi regions, and the distance to them.
- For each vertex of the tree, it(s) closest site(s) and its distance to it (them).

THEOREM 1. *The Voronoi diagram of a set of $n$ points on a tree of size $m$ has complexity $n+m-1$ and can be computed in optimal $\Theta(m+n\log n)$ time.*

*Sketch of the proof:* The complexity of the diagram follows from the fact that the Voronoi regions are connected. The construction of the diagram can be obtained by the following algorithm:

1. Sort the sites on each edge, producing the Voronoi partition of each edge between its extreme sites. If there are more than two sites on some edge the two extreme sites are relevant for the next steps.
2. Compute the spanning tree $S_T$ of the remaining sites in $T$. Let $T'$ be the result of the following operation: First we remove all maximal subtrees $u$ of $T-S_T$; by the *hanging* node $h(u)$ of $u$ we denote the unique node of $u$ that belongs to $S_T$. Next we replace every maximal chain of edges of $S_T$ that contains only degree 2 vertices by a single edge of the same length.
3. Partition the remaining sites into two sets of about the same cardinality and compute their Voronoi diagram on the contracted tree $T'$ by divide and conquer.
4. Traverse the original tree, placing the bisectors in their real position, and assign each subtree to the Voronoi region of its hanging node.

The correctness of the algorithm is based on the connectivity of the Voronoi regions, and its complexity analysis strongly relays on the fact that only $O(m)$ sites survive step 1, and that the contracted tree has complexity $O(n)$.

Even on a single line any algorithm producing the Voronoi diagram of a set of $n$ sites solves the $\epsilon$-closeness problem. □

**Observation:** Notice that for $m=O(1)$ the previous theorem matches the well known results for the Voronoi diagram of $n$ points on a line.

We now consider the weighted Voronoi diagram $WVD_T(S)$, where each site $p_i \in S$ gets assigned a weight $w_i$, as a multiplicative factor to its associate distance function. In other words, the weighted Voronoi regions are $WVR(p_i)=\{x \in T \mid w_i{\cdot}d(x,p_i) < w_j{\cdot}d(x,p_j)\ \forall j \neq i\}$. Note that the weighted Voronoi regions do not need to be connected and subtrees do not necessarily belong to the same weighted Voronoi region as their hanging node. Therefore we need more information in the description of the diagrams, i.e. for each edge we keep the bisector points sorted by their order on the edge.

THEOREM 2. *The weighted Voronoi diagram of a set of $n$ points on a tree of size $m$ has complexity $\Theta(nm)$ and can be computed in $O(mn\log n)$ time.*

*Sketch of the proof:* The complexity of the diagram follows from the known fact that the complexity of the diagram on a line is $O(n)$. In fact, a linear number of bisector points may

appear outside the interval given by the two extreme sites and so we can multiply this by adding a branching vertex of degree $m$ in between.

A straightforward adaption of the divide and conquer approach of the first algorithm (without contracting process) allows to compute the diagram in $O(mn \log n)$ time. □

**Observation:** The previous bound is tight in the sense that for $m = O(1)$ it fits the known computation results of the weighted Voronoi diagram on a line.

We now generalize the previous Voronoi diagrams in the sense of [1]. Assume that the $n$ sites are classified in $k$ types, modeled by $k$ colors ($k \leq n$). If $p$ denotes a site of color $c$, its farthest color Voronoi region $FCVR_T(p)$ is formed by all the points $x$ of $T$ for which $c$ is the farthest color and $p$ is the closest $c$-colored site.

THEOREM 3. *The farthest color Voronoi diagram of $n$ points of $k$ colors on a tree of complexity $m$ has complexity $2(n-k)+m+1$ and can be computed in $O(km + n \log n)$ time.*

*Sketch of the proof:* The complexity of the $FCVD$ is $2(n-k)+m+1$, since the regions are connected and the $FCVD$ of $n$ points of $k$ colors on a line can be proved to have complexity $2(n-k)+1$.

For the construction, the $FCVD$ of $n$ points on a line can be computed in optimal $\Theta(n \log n)$ time by a careful use of the fact that the diagram is the projection of the upper envelope of some monochrome polygonal lines of slopes 1 and $-1$. The tree structure adds a factor $k$ at each of its vertices. □

**Observation:** Recall that the above bounds fit the results of Theorem 1 when $k = 1$, as well as the mentioned line results when $m = O(1)$. It is also worth to say that the previous theorem improves a straightforward application of the known results on envelopes [10].

We finally consider the weighted farthest color Voronoi diagram on a tree, $WFCVD_T(S)$, a generalization of the $FCVD_T(S)$ where each site has a weight assigned, in the usual multiplicative way.

THEOREM 4. *The complexity of the weighted farthest color Voronoi diagram of $n$ points of $k$ colors on a tree of complexity $m$ is bounded above by $O(mn\alpha(k))$, where $\alpha(k)$ is the inverse of Ackermann's function, and from below by $\Omega(mn)$, and it can be computed in $O(mn(\log n + \alpha^2(k) \log k)$ time.*

*Sketch of the proof:* For any value of $m$, $n$ and $k$, we can construct a tree $T$, with size $m$ and arbitrary degrees, and a set $S$ of $n$ points of $k$ colors, with an arbitrary number of points of each color, such that the $WFCVD_T(S)$ has complexity $(m-1)n$.

The upper bound is obtained by applying Corollary 6.3 of [10] to each edge of the tree.

The algorithm that constructs the diagram combines the $k$ monochrome weighted Voronoi diagrams. □

**Observation:** When all the sites are of the same type ($k = 1$) this result matches that of Theorem 2. On the other hand, for the case of the tree being a line ($m = O(1)$), there is still space for a small improvement, since the running time of the algorithm turns out to have an extra $\alpha(k)$ factor.

## 3. EXTENSION TO GRAPHS

In case of graphs things are a bit different. For example the shortest path between two points is no longer unique. Despite such effects similar arguments can be applied.

Let $e$ denote the number of egdes of a graph $G = (V, D)$. The Voronoi diagram on graphs has complexity $O(n + e)$. Let $n_i$ denote the number of sites of an edge $i$. Each edge is divided into at most $n_i + 2$ pieces; two bisectors may have sites outside $i$. Summing up yields $\sum_i (n_i + 2) = n + 2e$.

For computing the diagram we again apply a divide and conquer approach. Divide the set of sites into two subsets of about the same cardinality, compute their Voronoi diagrams on the graph. Fortunately, the merge step can be done in $O(e)$. The site of one of the diagrams wins at a vertex, i.e. it is the closest site to the vertex. This closest site may build new bisectors with some sites of the other diagram. Fortunately, this can be done vertex by vertex, every edge may be concerned.

Similar techniques and adaptions work also for the more complicated cases giving rise to the results stated in Table 1.

## 4. REFERENCES


[1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, B. Palop, and V. Sacristán. Smallest color-spanning objects. In *Proc. 9th Annu. European Sympos. Algorithms*, volume 2161 of *Lecture Notes Comput. Sci.*, pages 278–289. Springer-Verlag, 2001.

[2] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.

[3] F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted Voronoi diagram in the plane. *Pattern Recogn.*, 17:251–257, 1984.

[4] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[5] M. Erwig. The graph voronoi diagram with applications. *Networks*, 36:156–163, 2000.

[6] S. L. Hakimi, M. Labbé, and E. Schmeichel. The Voronoi partition of a network and its implications in

location theory. *ORSA J. Comput.*, 4(4):412–417, 1992.

[7] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete Comput. Geom.*, 9:267–291, 1993.

[8] K. Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Inform. Process. Lett.*, 27:125–128, 1988.

[9] A. Okabe, B. Boots, K. Sugihara, and S-N. Chiu *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 2nd edition, 2000.

[10] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.

# Vertex $\pi$-guards in Simple Polygons

Bettina Speckmann *
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
speckmann@inf.ethz.ch

Csaba D. Tóth *
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
toth@inf.ethz.ch

## Abstract

We consider the problem of finding the worst case optimal number of vertex $\pi$-guards that collectively monitor a simple polygon with $n$ vertices. Variants of the problem differ in conditions on the allowable orientations of $\pi$-guards at vertices. We show that any simple polygon with $n$ vertices, $k$ of which are convex, can be monitored by at most $\lfloor(2n-k)/3\rfloor$ vertex $\pi$-guards which are *edge-aligned*. This bound is tight for the worst case families of polygons known so far. We also show that $\lfloor n/2\rfloor$ *general* vertex $\pi$-guards are always sufficient, improving upon the earlier bound of $\lfloor(3n-5)/4\rfloor$. To achieve these bounds we allocate vertex $\pi$-guards to vertices using a pseudo-triangulation of the polygon; each pseudo-triangle is monitored by vertex $\pi$-guards on its boundary.

## 1 Introduction

Urrutia [2] asked the following question: What is the minimal number of vertex $\pi$-guards that can collectively monitor any simple polygon $P$ with $n$ vertices. A vertex $\pi$-guard is given by a pair $(v, H^v)$ where $v$ is a vertex of $P$ and $H^v$ is a closed half-plane $H^v$ such that $v$ is on the boundary of $H^v$. There may be at most one $\pi$-guard at each vertex of $P$. A $\pi$-guard $(v, H^v)$ *monitors* $a \in P$ if and only if the closed line segment $va$ is in $P \cap H^v$.
A $\pi$-guard at a reflex vertex cannot monitor the complete angular domain. Restrictions on the orientation of the half-plane $H^v$, where $v$ is a reflex vertex, lead to three variants of the problem. We may require that at every reflex vertex $v$, the possible guard $(v, H^v)$ be

- *inward-facing*, i.e. the two sides of $P$ adjacent to $v$ are disjoint from the interior of $H^v$;
- *edge-aligned*, i.e. inward-facing and the boundary of $H^v$ is collinear with one of the edges adjacent to $v$;
- *general*, i.e. we do not restrict the orientation of $H^v$.

*The authors acknowledge support from the Berlin-Zürich European Graduate Program "Combinatorics, Geometry, and Computation".

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

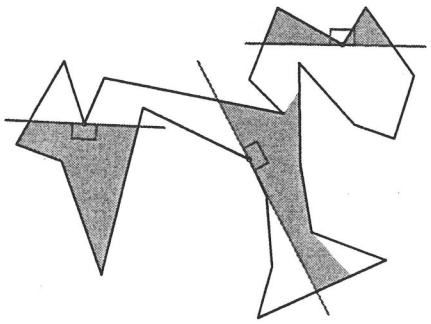


**Figure 1: A polygon with an inward-facing (left), an edge-aligned (middle), and an outward-facing (right) vertex $\pi$-guard.**

Urrutia [8] found a family of polygons that require $\lfloor 5(n-1)/8\rfloor$ inward-facing or edge-aligned vertex $\pi$-guards, but showed only the sufficiency of $n-2$ vertex $\pi$-guards [2]. In the restricted models of *inward-facing* or *edge-aligned* $\pi$-guards, this is still the best previously known upper bound. For *general* $\pi$-guards the upper bound was improved to $\lfloor(3n-5)/4\rfloor$ using a so-called *dense decomposition* of the simple polygon [6]; later Brumberg et al. [1] claimed a simple proof for an upper bound of $\lfloor 5n/6\rfloor$. The best known lower bound of $\lfloor(n-1)/2\rfloor$ follows from a family of polygons called *monotone mountains* [4] (polygons with two chains of reflex vertices). In this paper, we improve the upper bounds for all three models.

**Theorem 1** *$\lfloor n/2\rfloor$ vertex $\pi$-guards can always monitor a simple polygon with $n$ vertices in the general model.*

**Theorem 2** *$\lfloor(2n-k)/3\rfloor$ edge-aligned vertex $\pi$-guards can always monitor a simple polygon with $n$ vertices, $k$ of which are convex.*

The latter bound is tight for the family of polygons of Urrutia, for which the number of convex vertices is $k = (n-1)/8 + 2$.
Our allocation of guards is based on a *pseudo-triangulation* [3] or *geodesic triangulation* of the simple polygon $P$. A pseudo-triangulation decomposes $P$ along non-crossing diagonals into *pseudo-triangles*. It is easy to see that a pseudo-triangle with $\ell$ vertices can always be monitored with at most $\lfloor(2\ell-3)/3\rfloor$ vertex $\pi$-guards. Note though, that this does not immediately imply that an upper bound of $2n/3 + O(1)$ for the total number of guards required by $P$,

since adjacent pseudo-triangles of the decomposition share two vertices and optimal guard allocations in two pseudo-triangles sharing a vertex $v$ can prescribe contradicting orientations for a guard $(v, H^v)$. As we will see in Section 3, this kind of conflict can be resolved by using a characteristic property of pseudo-triangulations, namely the fact that every vertex is either a convex vertex of $P$ or a reflex vertex of one of its adjacent pseudo-triangles.

The remainder of the paper is organized as follows. We first recall some definitions and give some basic facts concerning pseudo-triangles and pseudo-triangulations. We then proceed to prove Theorem 2 in Section 3. Section 4 illustrates how a single pseudo-triangle with $\ell$ vertices can be guarded with $\lfloor \ell/2 \rfloor$ vertex $\pi$-guards on its boundary. Finally in Section 5, we sketch a proof of Theorem 1.

## 2 Preliminaries

A *pseudo-triangle* $T$ is a simple polygon with exactly three convex vertices, called *corners*. If $a, b$, and $c$ are the corners of $T$, we denote by $\overline{ab}$ (and similarly by $\overline{bc}$ and $\overline{ca}$) the chain of reflex vertices between $a$ and $b$ in counterclockwise direction on the boundary of $T$. For a simple polygon $P$, a *(minimum) pseudo-triangulation* is a set $D$ of non-overlapping pseudo-triangles such that $P = \bigcup D$, vertices of pseudo-triangles are vertices of $P$, and every vertex of $P$ is either convex in $P$ or reflex in a pseudo-triangle $T \in D$.

As triangles are also pseudo-triangles, any triangulation of a simple polygon is a decomposition into pseudo-triangles. Note, though, that we consider exclusively decompositions (i.e., pseudo-triangulations) where each vertex is a reflex vertex of a face: either of the outer face, or of a pseudo-triangle. This property is also known as *pointedness* [5]. Every simple polygon has at least one pseudo-triangulation, but this pseudo-triangulation is not necessarily unique. Observe that a pseudo-triangle in a pseudo-triangulation is uniquely determined by its three corners; hence we use the (non-abusive) notation $abc$ for a pseudo-triangle with corners $a, b$, and $c$. The number of pseudo-triangles in a (minimum) pseudo-triangulation of a simple polygon $P$ is always the number of convex vertices of $P$ minus two.

The *dual graph* $G(D)$ of a pseudo-triangulation $D$ is defined by choosing $D$ as the node set of $G(D)$ and connecting two nodes by an edge iff the corresponding pseudo-triangles have a common edge (a diagonal of $P$). Note that $G(D)$ is always a tree if $D$ is a pseudo-triangulation of a simple polygon. (In our terminology, polygons have *vertices* and *sides*; graphs have *nodes* and *edges*.)

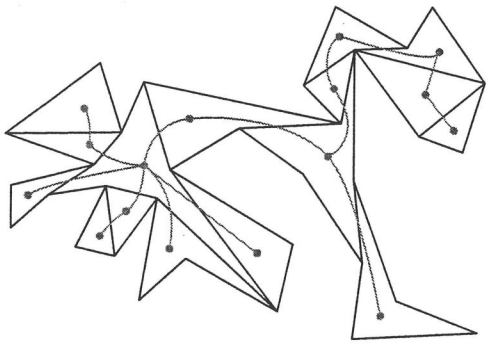


**Figure 2: A pseudo-triangulation and its dual graph for a simple polygon.**

For every pseudo-triangulation $D$ of $P$, there is a convex vertex $x \in V(P)$ such that $x$ is a vertex (a corner) of a unique pseudo-triangle $T \in D$. Let $y \in V(P)$ be a vertex adjacent to $x$. We denote by $G(D)_{xy}$ the rooted tree $G(D)$ where the root corresponds to the pseudo-triangle containing side $xy$. We write $S < T$ for two pseudo-triangles $S, T \in D_{xy}$ if $S$ is the successor of $T$ in $G(D)_{xy}$. As a shorthand, we denote by $D_{xy}$ the pseudo-triangulation $D$ together with the partial order of successor-ancestor relationship in the tree $G(D)_{xy}$.

For a reflex vertex $v$ of a pseudo-triangle we consider only vertex $\pi$-guards which are *inward-facing* (as defined above) or *outward-facing*, that is, the sides of $P$ adjacent to $v$ are in the closed halfplane $H^v$ (see also Fig. 1). For a corner $v$, we consider only two possibilities: either $(v, H^v)$ completely covers the angular domain, or it does not.

**Proposition 3** *A pseudo-triangle $T$ with $\ell$ vertices can be monitored by at most $\lfloor (2\ell - 3)/3 \rfloor$ vertex $\pi$-guards: one at a corner and at most $\lfloor 2(\ell - 3)/3 \rfloor$ outward-facing guards at reflex vertices along two chains (see Fig. 3 left).*

Using the same idea, we can deduce an upper bound of $\lfloor (\ell - 1)/2 \rfloor$ for so-called *monotone mountains* – defined by J. O'Rourke [4].

**Proposition 4** *A pseudo-triangle $T$ with $\ell$ vertices and $\overline{bc} = \emptyset$ can be monitored by at most $\lfloor (\ell - 1)/2 \rfloor$ vertex $\pi$-guards: one at the corner $b$ or $c$ (but not at both) and at most $\lfloor (\ell - 3)/2 \rfloor$ outward-facing guards at reflex vertices along one (non-empty) chain (see Fig. 3 right).*

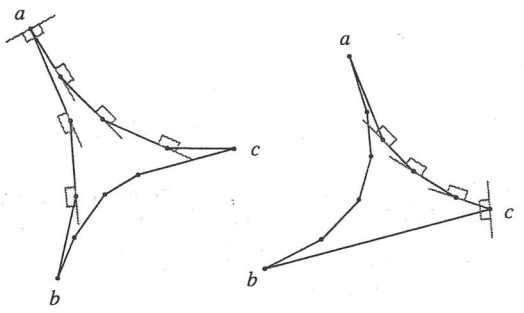



**Figure 3: Guards at a corner and along two adjacent chains monitor the complete pseudo-triangle.**

## 3 Edge-aligned $\pi$-guards

**Proof [Theorem 2]:** Following, and generalizing the idea of Proposition 3, we describe three different guard allocations for $P$ that jointly use a total of $2n - k$ edge-aligned vertex $\pi$-guards. We define these allocations with resepct to an arbitrary but fixed pseudo-triangulation $D_{xy}$. In all three guard allocations and for each pseudo-triangle $T \in D_{xy}$, there will be vertex $\pi$-guards at one corner of $T$ and along the two chains adjacent to that corner such that these guards collectively monitor $T$. The three guard allocation schemes will jointly use all three corners (and thus different pairs of chains) for each $T \in D_{xy}$.

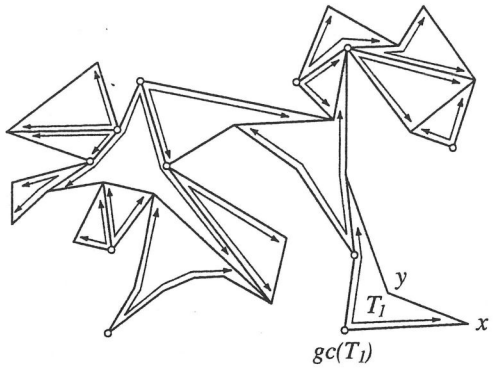



**Figure 4: Propagating the side directions along $G(D)_{xy}$.**

Consider the pseudo-triangle $T_1$ at the root of $G(D)_{xy}$. We can guard $T_1$ in three different ways by choosing any one of its three corners and placing guards at that corner (the *guarded corner* $gc(T_1)$ of $T_1$) and along its two adjacent chains (see Fig. 3). Every choice of guard allocation for $T_1$ induces a direction on its chains: The sides on the chains adjacent to $gc(T_1)$ are directed *away* from $gc(T_1)$, the sides on the remaining chain have *no direction*. Note that we do indeed distinguish between three directions for each side – the two standard directions and *no direction*. Fixing the directions for one chain of a pseudo-triangle uniquely determines the directions for the remaining ones. Furthermore, each side of the pseudo-triangulation can only have one direction, i.e. a pseudo-triangle inherits its directions from its parent via their joined side. Therefore, choosing a guarded vertex $gc(T_1)$ in $T_1$ not only induces a unique direction on the sides of $T_1$, but actually determines the direction of all sides of $D_{xy}$ (see Fig. 4). Furthermore, since each pseudo-triangle $T$ now has exactly one corner with two outgoing edges, we define this corner to be the guarded corner $gc(T)$. Finally, note that for each corner $v$ of every $T \in D_{xy}$ there is a choice of the guarded corner of $T_1$, such that the resulting orientation of $D_{xy}$ implies that $v$ is the guarded corner of $T$. After choosing $gc(T_1)$ and propagating the directions along $G(D)_{xy}$ we place in a second step vertex $\pi$-guards along the directed edges of $D_{xy}$. More specifically, in each $T \in D_{xy}$ we place a vertex $\pi$-guard at the corner $gc(T)$ and *outward-facing* $\pi$-guards along the two directed chains adjacent to $gc(T)$ (see Fig. 5). By Property 3, these guards collectively monitor $T$. Note that it is always sufficient to place at most one $\pi$-guard at each vertex since every vertex is a reflex vertex of at most one pseudo-triangle of $D_{xy}$ and $\pi$-guards at reflex vertices are outward-facing. (This already gives an upper bound of $\lfloor (2n-k)/3 \rfloor$ in the general model.)
In the last step, we rotate every vertex $\pi$-guard $(v, H^v)$ at every reflex vertex $v$ until it is edge-aligned. Recall that $v$ is a reflex vertex of exactly one $T_v \in D_{xy}$ and its guard monitors $T_v$ only if $v$ is on a chain adjacent to $gc(T_v)$. Rotate $(v, H^v)$ so that the area monitored in $T_v$ increases. The resulting set of vertex $\pi$-guards still monitors each $T \in D_{xy}$: The $\pi$-guards at reflex vertices of a $T \in D_{xy}$ monitor at least as much area as before, and the $\pi$-guards at $gc(T)$ still monitor the angular domain of $gc(T)$ in $T$, by the choice of $gc(T)$ (see Fig. 6).

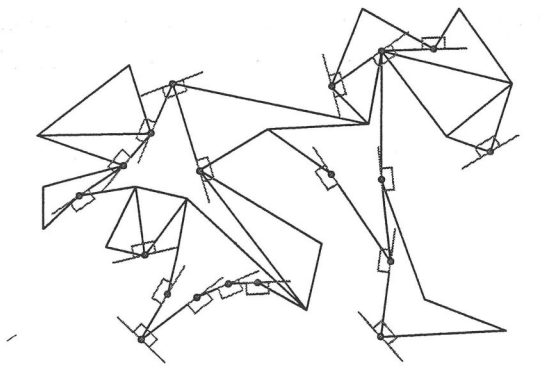


**Figure 5: Guard allocation according to the vertex assignment depicted in Fig. 4 after the second step.**

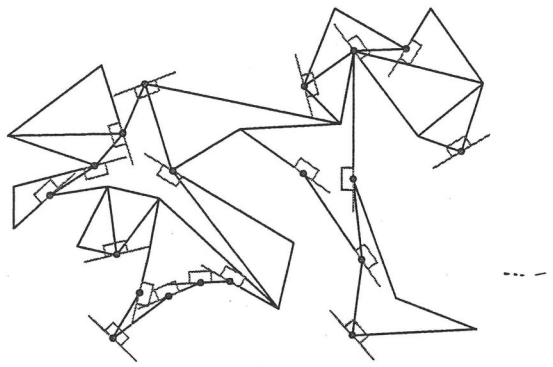


**Figure 6: Guard allocation according to the vertex assignment depicted in Fig. 4 after rotation.**

Finally, we show that the total number of guards is $2n-k$. We use $2\ell_1 - 3$ guards on the boundary of the pseudo-triangle $T_1$ corresponding to the root of $D_{xy}$, where $\ell_1$ is the number of vertices of $T_1$. For all other $T_i \in D_{xy}$ with $\ell_i$ vertices, two guards out of the $2\ell_i - 3$ are already counted on the boundary of the ancestor of $T_i$. (This can be easily checked for all possible mutual positions of two adjacent pseudo-triangles: i.e., the common vertices are both corners, both reflex, or one is a corner and the other one is reflex.) Hence all three guard allocations use a total of $2 + \sum_{i=1}^{p}(2\ell_i - 5) = 2 - 5p + 2\sum_{i=0}^{p} \ell_i$ guards where $p$ denotes the number of of pseudo-triangles, i.e. $p = k-2$. Using the fact that $n - 2 = \sum_{i=1}^{p}(\ell_i - 2) = (\sum_{i=1}^{p} \ell_i) - 2p$, we obtain $2n - 2 - p = 2n - k$ guards in total. □

## 4 $\pi$-guards in one pseudo-triangle

**Lemma 5** *A pseudo-triangle $abc$ with $\ell$ vertices can be monitored by $\lfloor \ell/2 \rfloor$ vertex $\pi$-guards such that there is at most one inward-facing $\pi$-guard, which lies on a prescribed chain, say, on $\overline{bc}$, and all other $\pi$-guards at reflex vertices are outward-facing.*

**Proof (sketch):** We propose two possible guard allocations for $abc$ with a total of at most $\ell$ guards, such that all $\pi$-guards at reflex vertices are outward-facing, except for at most one guard on the chain $\overline{bc}$. We consider two cases:

**Case 1:** There is a vertex $v \in \overline{bc}$ visible from $a$ – see Fig. 7.

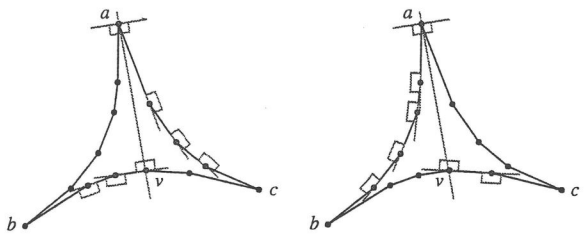



**Figure 7: The two guard allocations in Case 1.**

**Case 2:** There is no vertex on $\overline{bc}$ visible from $a$ – see Fig. 8.

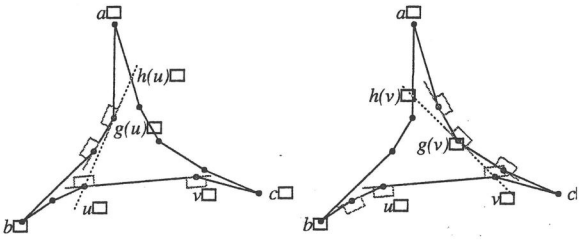



**Figure 8: The two guard allocations in Case 2.**

□

## 5 Upper bound for the general model

To establish the upper bound in the general model, we rely on a special pseudo-triangulation of $P$, which corresponds to the so-called *pulling triangulation* of convex polygons. A *pulling pseudo-triangulation* of a simple polygon is obtained by choosing (i.e. *pulling*) a convex vertex $x$ and connecting it to all other convex vertices of $P$ via geodesic paths (see Fig. 9). The pulling pseudo-triangulation has the special property that every pseudo-triangle $T \in D_{xy}$, with the exception of the root, has at least one corner adjacent to its parent, i.e. at least one of the two vertices which $T$ shares with its parent is a corner with respect to $T$.

**Proof [Theorem 1] (sketch):** The basic idea of the proof is to construct a partition $\mathcal{F}$ of the vertices $V_P$ of $P$ and to place vertex $\pi$-guards at at most $\frac{1}{2}|F|$ vertices for each $F \in \mathcal{F}$. It follows that the total number of vertex $\pi$-guards will be at most $\lfloor n/2 \rfloor$.

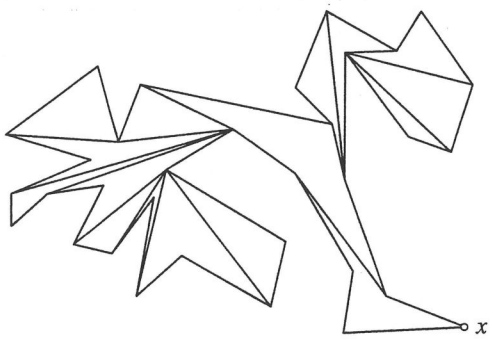



**Figure 9: A pulling pseudo-triangulation with respect to $x$.**

In order to define the partition $\mathcal{F}$, consider an ordered pulling pseudo-triangulation $D_{xy}$. We assign a subset $\Psi_T \subset V_T$ to each pseudo-triangle $T \in D_{xy}$ such that $\mathcal{F} := \{\Psi_T : T \in D_{xy}\}$ is a partition of $V_P \setminus \{x\}$. Then we place vertex $\pi$-guards at some vertices $\Phi_T \subset \Psi_T$.

The pseudo-triangles of $D_{xy}$ are processed from bottom to top, that is, $T \in D_{xy}$ is processed when all its successors have already been processed. Processing $T \in D_{xy}$ involves the following three tasks:

1. defining $\Psi_T \subset V_T$
2. specifying $\Phi_T \subset \Psi_T \subset V_T$ with $|\Phi_T| \leq \frac{1}{2}|\Psi_T|$
3. verifying that
   (a) the vertex $\pi$-guards placed on the boundary of $T$ (not only those in $\Phi_T$) can be oriented such that they collectively monitor $T$
   (b) for each vertex $v$, the orientation of a $\pi$-guard at $v$ is consistent in every $T' \in D_{xy}$ adjacent to $v$

The consistency of the orientation relies heavily on the *pointedness* of our pseudo-triangulation: a $\pi$-guard at a vertex $v$ can monitor all convex angular domains adjacent to $v$ and at the same time it can be an outward-facing $\pi$-guard with several possible orientations in a pseudo-triangle where $v$ is a reflex vertex. This means that if we include at one step in the construction a vertex $v$ in $\Phi_T$ where $v$ is a corner in $T$ then we are free to choose the orientation of the $\pi$-guard at $v$ whenever we are processing a pseudo-triangle $T'$ in which $v$ is a reflex vertex as long as the guard remains out-ward-facing in $T'$. □

## References


[1] V. Brumberg, S. Ramaswami, and D. Souvaine, Experimental results on upper bounds for vertex pi-lights, in: *Proc. 11th Ann. Fall Workshop on Comput. Geom. (Brooklyn, NY, 2001).*

[2] V. Estivill-Castro, J. O'Rourke, J. Urrutia, and D. Xu, Illumination of polygons with vertex guards, *Inform. Process. Lett.* **56** (1995) 9–13.

[3] M. Pachiola and G. Vegter, Pseudo-triangulations: Theory and applications, in: *Proc. 12th ACM Symp. Comput. Geom. (Philadelphia, PE, 1996)*, ACM-Press, 291–300.

[4] J. O'Rourke, Vertex $\pi$-lights for monotone mountains, in: *Proc. 9th Canadian Conf. Comput. Geom. (Kingston, ON, 1997)*, 1–5.

[5] I. Streinu, A combinatorial approach to planar non-colliding robot arm motion planning, in: *41st Symp. Foundations of Comp. Sci. (Redondo Beach, CA, 2000)*, IEEE Comp. Soc., 443–453.

[6] Cs. D. Tóth, Illuminating polygons with $\pi$-floodlights, in: *Proc. Int. Conf. on Comput. Sci. (San Francisco, CA, 2001) Part I*, Lecture Notes on Comp. Sci. Vol. 2073, Springer, Berlin, 2001, 772–781.

[7] J. Urrutia, Art Gallery and Illumination Problems, in: *Handbook on Computational Geometry (J. R. Sack, J. Urrutia, eds.)*, Elsevier, Amsterdam, 2000, 973–1027.

[8] J. Urrutia, personal communication, 2001.

# The Orthogonal Fortress Problem with Floodlights

Andreas Spillner
Institut für Informatik
Friedrich Schiller Universität Jena
07743 Jena, Germany
spillner@minet.uni-jena.de

Hans-Dietrich Hecker
Institut für Informatik
Friedrich Schiller Universität Jena
07743 Jena, Germany
hdh@mipool.uni-jena.de

## ABSTRACT

We study the problem of illuminating the exterior of a polygon with floodlights, which is known as the *fortress problem*. For some classes of rectilinear polygons we give bounds for the number of floodlights that are always sufficient for illumination.

## 1. INTRODUCTION

We suppose the reader to be familiar with the concepts of polygons, simple polygons and rectilinear polygons. Given a simple polygon $P$ we denote the interior with $int(P)$, the exterior with $ext(P)$ and the boundary with $bd(P)$. Given a point $x$ we say that a point $y$ is visible from the point $x$, if the segment $\overline{xy}$ is completely contained in $int(P) \cup bd(P)$ or in $ext(P) \cup bd(P)$. A floodlight is a light source, that projects light inside a cone $C$. If the size of $C$ is $\alpha$, we will call $C$ an $\alpha$-floodlight. If the apex of $C$ is located at a vertex of a polygon, we will call $C$ a vertexlight. We do not allow two floodlights to share a common apex. For convenience we identify a floodlight with the cone it projects light in. Given a floodlight $F$ with apex $a$ we say that a point $y$ is illuminated by $F$, if $y$ is visible from $a$ and $y \in F$. We would like to refer the reader to [3] for a survey on art gallery and illumination problems. There one can also find a section about floodlight illumination problems.

We now introduce some commonly used concepts in the world of rectilinear polygons. First for ease of discussion we will call a vertical edge $E$ of a rectilinear polygon $P$ a West-edge or W-edge for short, if $int(P)$ is to the right of $E$. In a similar manner we will speak about North-edges, East-edges and South-edges. A simple rectilinear polygon $P$ will be called $x$-monotone, if for every vertical straight line $L$ the set $L \cap P$ is convex. It will be called $y$-monotone, if for every horizontal straight line $L$ the set $L \cap P$ is convex. A simple rectilinear polygon is called orthogonally convex if it is $x$-monotone and $y$-monotone.

Our research was motivated by the work of J. Urrutia and others presented in [1]. There they show that for every $\alpha \in [0, \frac{\pi}{2})$ there exists a simple rectilinear polygon the interior of which can not be illuminated by $\alpha$-vertexlights. On the other hand for every $\alpha \in [\frac{\pi}{2}, \frac{3\pi}{2})$ they are able to prove that for a rectilinear polygon $P$ with $n$ vertices and $h$ holes $\lfloor \frac{3n+4(h-1)}{8} \rfloor$ $\alpha$-vertexlights are always sufficient and somtimes necessary to illuminate the interior of $P$. In the *fortress problem*, which requires us to illuminate the exterior of $P$, they show that $\frac{n}{2} + 2$ $\frac{\pi}{2}$-vertexlights are always sufficient and sometimes necessary. When we allow the apices of the floodlights to be placed on the boundary of $P$ then the bound for illuminating the interior reduces to $\lfloor \frac{n+2h}{4} \rfloor$ while the bound for illuminating the exterior does not change.

The *fortress problem* was independently posed by D. Wood and J. Malkelwich. In the original setting the illumination has to be achieved with vertexguards, which in our context are $2\pi$-vertexlights. It is known, that the exterior of every simple polygon with $n$ vertices can be illuminated by $\lceil \frac{n}{2} \rceil$ $2\pi$-vertexlights. Furthermore the exterior of every simple rectilinear polygon with $n$ vertices can be illuminated by $\lceil \frac{n}{4} \rceil + 1$ $2\pi$-vertexlights. These two bounds are tight and can be found in [2]. It is not hard to see, that in the case of simple rectilinear polygons $\alpha$-vertexlights with $\alpha \in [\frac{3\pi}{2}, 2\pi)$ are as powerfull as $2\pi$-vertexlights. On the other hand for every $\alpha \in [0, \frac{\pi}{2})$ there is a simple rectilinear polygon the exterior of which can not be illuminated by $\alpha$-vertexlights.

We study the *fortress problem* for rectilinear polygons too, and try to find out, if the upper bound of $\frac{n}{2} + 2$ given in [1] can be improved, when we use $\alpha$-vertexlights with $\alpha \in (\frac{\pi}{2}, \frac{3\pi}{2})$. As long as no ambiguity can arise, we will occasionally omit the size of the floodlights used.

## 2. UPPER BOUNDS

First we consider the class of orthogonally convex polygons.

THEOREM 1. *Let $P$ be an orthogonally convex polygon and $\alpha \in [\frac{5\pi}{4}, \frac{3\pi}{2})$. Then we can illuminate the exterior of $P$ with $\lfloor \frac{n}{3} \rfloor + 4$ $\alpha$-vertexlights.*

PROOF. W.l.o.g. we consider $\frac{5\pi}{4}$-vertexlights. Let $E_N$ be a N-edge of $P$, such that $P$ is contained in the halfplane bounded by the straight line containing $E_N$. Analogously we define an E-edge $E_E$, a S-edge $E_S$ and a W-edge $E_W$. Since $P$ is an orthogonally convex polygon these edges are unique. Please compare figure 1.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

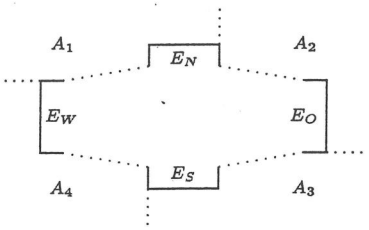



**Figure 1: An orthogonally convex polygon.**

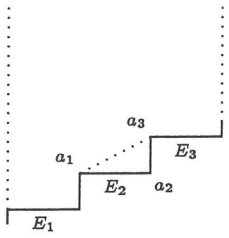



**Figure 2: A section of three N-edges in region $A_1$.**

Lets have a closer look at the region $A_1$. We consider three N-edges $E_1$, $E_2$ and $E_3$ as displayed in figure 2. By $D$ we denote the triangle with vertices $a_1$, $a_2$ and $a_3$. In case the size of the interior angle of $D$ at $a_1$ is less than $\frac{\pi}{4}$ we place two floodlights as indicated in figure 3. Otherwise we place two floodlights as shown in figure 4. Floodlights are indicated by shaded wedges. It is easy to see, that in either case the whole region above the edges $E_1$, $E_2$ and $E_3$ is illuminated. So roughly spoken we need two floodlights per six vertices of $P$, which gives the desired upper bound. □

On the basis of this result we can prove an upper bound for the class of all simple rectilinear polygons. The proof employs standard tools such as the orthoconvex hull of a simple rectilinear polygon. However it consists in a rather lengthy case analysis, so that we will omit the proof here.

THEOREM 2. *Let $P$ be a simple rectilinear polygon with $n$ vertices and $\alpha \in [\frac{5\pi}{4}, \frac{3\pi}{2})$. Then we can illuminate the exterior of $P$ with $\lfloor \frac{3n}{8} \rfloor + 8$ $\alpha$-vertexlights.*

On the other hand we can generalize theorem 1 in the follwing way:

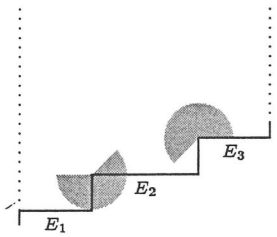



**Figure 3: One possible placement of the floodlights.**

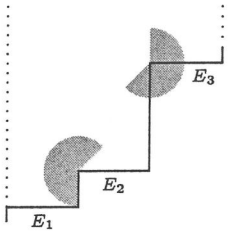



**Figure 4: Another possible placement for the floodlights.**

THEOREM 3. *Let $P$ be an orthogonally convex polygon with $n$ vertices, $k \in \mathbb{N} \setminus \{0,1,2\}$ and $\varepsilon \in [\frac{\pi}{2(k-1)}, \frac{\pi}{2(k-2)})$. Then the exterior of $P$ can be illuminated with $\lfloor \frac{k-1}{2k} n + \frac{6(k-1)}{k} \rfloor$ vertexlights the size of which is $\pi + \varepsilon$.*

The idea for the proof is basically the same as in the proof of theorem 1. But instead of only three N-edges we consider $k$ consecutive N-edges.

## 3. CONCLUDING REMARKS

We first remark, that we were able to prove that the upper bound of $\frac{n}{2} + 2$ given in [1] is even tight for $\alpha$-vertexlights with $\alpha \in [\frac{\pi}{2}, \pi]$. Furthermore we can show that the upper bounds given in theorem 3 are asymptotically tight at least for $k = 3$ and $k = 4$. Finally we can prove that the upper bound in theorem 2 is asymptotically tight. This proof is based on a construction presented in [1].

On the other hand we were able to show that for every $\alpha \in [\frac{\pi}{2}, 2\pi]$ the problem of finding the minimum number of $\alpha$-vertexlights sufficient to illuminate the exterior of a given simple rectilinear polygon is NP-hard.

## 4. REFERENCES

[1] J. Abello, V. Estivill-Castro, T. Shermer, and J. Urrutia. Illumination with orthogonal floodlights. In *LNCS*, volume 1004, pages 362–371, 1995.

[2] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[3] J. Urrutia. Art gallery and illumination problems. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry.* North Holland, 1998.

# Rectilinear Trees under Rotation and Related Problems

[Extended Abstract]

Benny K. Nielsen
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
Denmark
benny@diku.dk

Pawel Winter
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
Denmark
pawel@diku.dk

Martin Zachariasen
Dept. of Computer Science
University of Copenhagen
DK-2100 Copenhagen Ø
Denmark
martinz@diku.dk

## ABSTRACT

We consider the problem of finding a minimum spanning tree for a set of points in the plane where the orientations of edges are restricted to $\lambda$ uniformly distributed orientations, $\lambda = 2, 3, 4, \ldots$, and where the coordinate system can be rotated around the origo by an arbitrary angle. The most important case with applications in VLSI design arises when $\lambda = 2$; in this, so-called rectilinear case, the edges have to be parallel with the $x$- or $y$-axis. We suggest a straightforward algorithm to solve this problem. We also discuss how to solve the rectilinear Steiner tree problem in the rotational setting. Finally, we provide some computational results indicating the average savings for different values of $n$ and $\lambda$ both for spanning and Steiner trees.

## 1. INTRODUCTION

Suppose that we are given a set $P$ of $n$ points in the plane. We are interested in finding a minimum spanning tree (MST) or a Steiner minimum tree (SMT) for $P$ under the assumption that the edges are permitted to have a limited number of orientations. In fact, we will assume that these orientations are evenly spaced. More specifically, let $\lambda$ be a non-negative integer, $\lambda \geq 2$. Let $\omega = \pi/\lambda$. Permissible directions are then defined by the rays initiating from the origo and making angles $i\omega$, $i = 0, 1, ..., 2\lambda - 1$ with the positive $x$-axis. Finding such an MST is not a challenging problem since it can be defined as a minimum spanning tree problem in a complete graph with appropriate edge lengths.

Suppose that we are permitted to rotate the coordinate system. Rotation by $\omega$ (or a multiple of $\omega$) will have no impact on the lengths of the edges. But for any angle $\alpha \in [0, \omega[$, the edge lengths will change. What is the value of $\alpha$ minimizing the length of an MST for $P$? Once $\alpha$ is fixed, finding an MST is straightforward. Determining similar SMTs seems to be more complicated. However, for the most important rectilinear case, we will show that the search for such SMTs can be reduced to $O(n^2)$ rotational angles.

Our interest in rotated MSTs and SMTs with bounded orientations was motivated by recent developments in VLSI technology. It will soon be possible to manufacture chips with wires running in more than two directions. This makes the case $\lambda = 4$ important in practice. Furthermore, additional length savings seem to be available when the coordinate system can be rotated. This is in particular useful for small values of $\lambda$ and for nets with limited number of terminals. As $\lambda$ grows, the edge length variations become smaller. As the number of terminals increases, savings along some edges are "eaten" up by increased lengths of other edges.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

## 2. LENGTH OF A SEGMENT

Let us first examine the situation where there are only two points $a = (a_x, a_y)$ and $b = (b_x, b_y)$. It is obvious that the best minimum spanning tree is obtained when the coordinate system is rotated so that the segment $ab$ overlaps with one of the permissible orientations. So our problem is trivial. However, it is still interesting to determine how the length of $ab$ changes as the coordinate system is rotated.

Assume that $a$ and $b$ are on the horizontal $x$-axis and $a_x < b_x$. When the coordinate system is rotated by $\alpha$, $0 \leq \alpha \leq \omega$, then the length of $ab$, denoted by $|ab|_\alpha$, changes. It increases until $\alpha = \omega/2$, and then decreases until $\alpha = \omega$ (Fig. 1). More specifically,

$$|ab|_\alpha = |ab| \frac{\sin\alpha + \sin(\omega - \alpha)}{\sin(\omega)}$$

In particular, if $\omega = \pi/2$, then

$$|ab|_\alpha = |ab|(\sin\alpha + \cos\alpha)$$

It is obvious that the function $f_{ab}(\alpha) = |ab|_\alpha$ is periodically strictly concave (with the period $\omega$). Furthermore, the minimum is attained when the direction of the segment $ab$ overlaps with one of the direction rays.

## 3. MINIMUM SPANNING TREE

Consider a collection $S$ of segments. Their total length will be minimized when the rotation of the coordinate system forces the orientation of one of the segments to overlap with one of the direction rays. This follows immediately from the fact that segment lengths are piecewise strictly concave

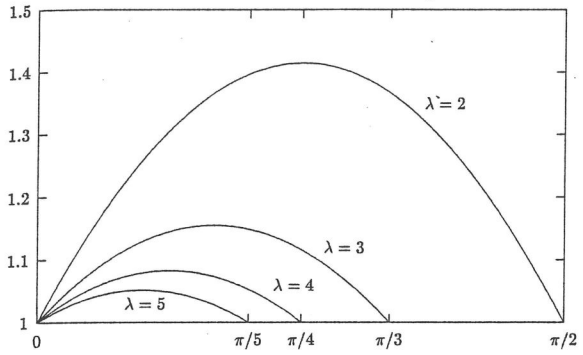



**Figure 1:** $f_{ab}(\alpha) = |ab|_\alpha$ **for** $\alpha \in [0, \frac{\pi}{\lambda}]$, $\lambda = 2, 3, 4, 5$.

functions of the rotation angle. The sum of piecewise strictly concave functions is also piecewise strictly concave.

Consider a set $P$ of $n$ points in the plane. Assume that the coordinate system has been rotated in such a way that an MST $T$ for $P$ is shortest possible. In view of the above remark, one of the edges of $T$ overlaps with one of the direction rays.

The algorithm to determine $T$ is therefore straightforward. For each pair of points $a$ and $b$ of $P$, consider the segment $ab$. Rotate the coordinate system so that the orientation of one of the direction rays overlaps with $ab$. Compute an MST for $P$ and store it provided that it is shorter than any MST found so far. Fig. 2 shows how the lengths of MSTs for a set of 10 points and $\lambda = 4$ changes when the coordinate system is rotated. The histogram was generated by computing 1000 MSTs for values of $\alpha$ evenly distributed in the interval $[0, \frac{\pi}{4}]$. The vertical dashed lines indicate MSTs in fact computed by the algorithm. The optimum is on the $x$-axis.

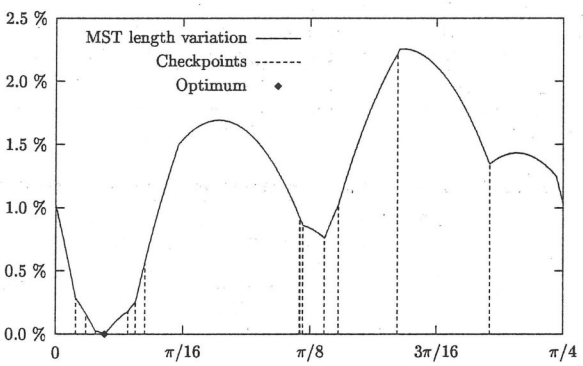



**Figure 2: MST length variation.**

For a fixed rotation of the coordinate system, an MST can be computed in $O(n \log n)$ time [6]. Since it is only necessary to consider $O(n^2)$ different rotations, the total running time for computing a rotational MST is $O(n^3 \log n)$. We will in Section 5 return to the issue of making this algorithm much more efficient in practice.

## 4. RECTILINEAR STEINER TREE

Consider the problem of interconnecting the set $P$ of points by a tree of minimum total length, but *allowing* additional junctions, so-called Steiner points. When a set of $\lambda$, $\lambda \geq 2$, uniformly spaced legal orientations is given, this is denoted the Steiner tree problem in uniform orientation metrics [1, 4]. The rectilinear ($\lambda = 2$) and Euclidean ($\lambda = \infty$) Steiner tree problems have received considerable attention in the literature [3, 5].

In this section we consider the rotational variant of the rectilinear Steiner tree problem: Find a shortest interconnection of $P$ using only two perpendicular orientations; note that the two orientations are not given, but are restricted to be perpendicular.

Assume that the coordinate system is rotated at an angle $0 \leq \alpha < \pi/2$. In this case the problem to be solved is the usual rectilinear Steiner tree problem, that is, to find a rectilinear Steiner minimum tree (RSMT). An RSMT is a union of full Steiner trees (FSTs) in which all terminals are leaves (and all interior nodes are Steiner points). Hwang [2] proved that there exists an RSMT for which every FST has a particular shape: The FST consists of a *backbone*, which is just a shortest path between two terminals, say $a$ and $b$, using at most one vertical and at most one horizontal line segment (in Fig. 3 the backbone consists of the line segments $ac$ and $cb$, where $c$ is the corner point of the backbone); all the remaining terminals in the FST are connected directly to the backbone using exactly one line segment (e.g., segment $ts$ in Fig. 3). Furthermore, none of the remaining terminals are connected to the backbone via the corner point of the backbone.

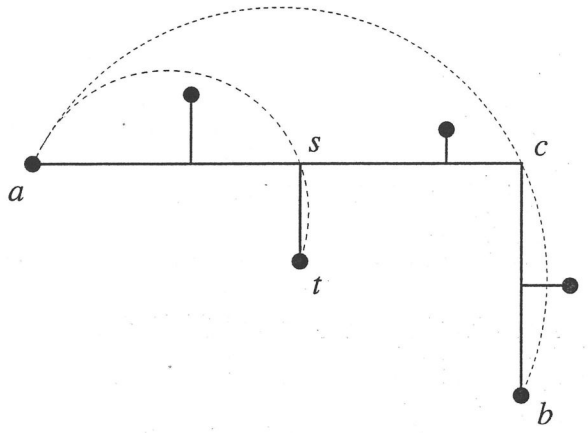



**Figure 3: Rotating a rectilinear FST with one corner point.**

Now let us assume that the rotational problem has an optimal solution for a given angle $0 \leq \alpha^* < \pi/2$. W.l.o.g. this optimal solution — which is an RSMT for the orientations given by $\alpha^*$ — consists of FSTs having the shape described above. We will now show that at least one of the FSTs in the RSMT must have a backbone that consists of *one* line segment, i.e., the backbone has no corner point.

Assume to the contrary that every FST has a backbone with a corner point. Let $F$ be an FST and let $|F|_\alpha$ be the length of the FST as a function of $\alpha$ (for $\alpha^* - \epsilon < \alpha < \alpha^* + \epsilon$ where $\epsilon > 0$ is sufficiently small). Consider again the FST in Fig. 3. The corner point $c$ will move along the indicated half-circle, such that the angle of the segments $ac$ and $cb$ remains at $\pi/2$. Thus the length of the backbone is a strictly concave function of $\alpha$ as shown in Section 2. Similarly, the length of a segment that connects one of the remaining terminals to the backbone will be a strictly concave function. Therefore, $|F|_\alpha$ is a strictly concave function of $\alpha$, and the same will hold for the sum of all FST lengths. Therefore, the RSMT can in fact be shortened, which is a contradiction.

Since at least one of the FSTs must have a backbone without a corner point, the orientation of the backbone line segment will overlap with the orientation given by its two endpoints. The optimal solution to the rotational rectilinear Steiner tree problem can therefore be found using an algorithm similar to the one described in Section 3: For each pair of points $a$ and $b$ in $P$, rotate the coordinate system so that one of the direction rays overlaps with the segment $ab$. Compute an RSMT for this direction and repeat this procedure for all pairs of points; the shortest RSMT computed will be an optimal solution to the rotational rectilinear Steiner tree problem.

For $\lambda > 2$ we conjecture that a similar reduction of the necessary angles can be obtained, but it will not be polynomial in the number of terminals as for the rectilinear Steiner tree problem; it is not enough only to consider the orientations given by pairs of points in $P$.

## 5. COMPUTATIONAL RESULTS

In this section we give some computational results indicating the effect of allowing rotations of the coordinate system when computing MSTs and rectilinear SMTs. We used two sets of problem instances: VLSI design instances and randomly generated instances (uniformly distributed in a square). The VLSI instances were made available by courtesy of IBM and Research Institute for Discrete Mathematics, University of Bonn. In this study we have focused on one particular chip from 1996.

### 5.1 Minimum Spanning Tree

Generally most of the edges for a given point set will not be part of a MST for any rotation angle. It is a waste of time to 'straighten' these edges and compute MSTs. Luckily most of these edges can be pruned by using so-called bottleneck Steiner distances.

Consider the complete graph $K_n$ where the vertices represent the points in the plane. Define the weight of the edge between vertices $a$ and $b$ to be $|ab|_{\omega/2}$. In other words, this weight is equal to the maximum distance between $a$ and $b$ when the coordinate system is rotated. Compute bottleneck Steiner distances in $K_n$. It is defined as the minimum of the longest edges encountered in all paths between $a$ and $b$ in $K_n$, and can be computed in time $O(n^2)$ for all pairs of points; see [3] for details. Let $B_{ab}$ denote the bottleneck Steiner distance between $a$ and $b$. Whenever $|ab| > B_{ab}$, then the edge $ab$ cannot be in any minimum spanning tree generated during the rotation of the coordinate system. Fig. 4 indicates how many edges survive the pruning for $n = 50$ and $\lambda = 2, 3, 4, 5$.

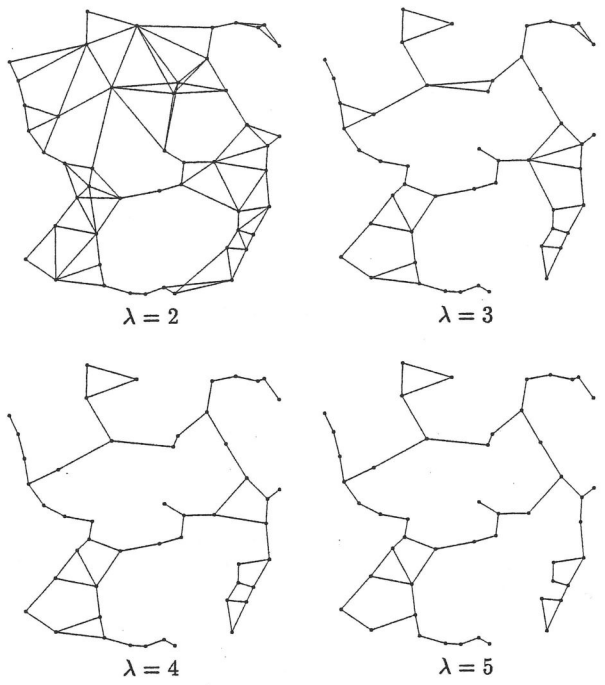



**Figure 4: Edges surviving pruning of $K_{50}$ for $\lambda = 2, 3, 4, 5$.**

| $\lambda$ | n=3 | n=4 | n=5 | n=10 | n=20 | n=50 | n=100 |
|---|---|---|---|---|---|---|---|
| 2 | 2.76 | 4.86 | 7.38 | 20.12 | 44.77 | 122.09 | 253.84 |
| 3 | 2.34 | 3.84 | 5.50 | 12.62 | 26.77 | 71.43 | 146.81 |
| 4 | 2.22 | 3.44 | 4.77 | 10.77 | 23.05 | 60.51 | 123.05 |
| 5 | 2.13 | 3.34 | 4.54 | 10.17 | 21.28 | 56.11 | 113.59 |
| 6 | 2.07 | 3.27 | 4.38 | 9.83 | 20.54 | 53.89 | 108.57 |
| 7 | 2.06 | 3.21 | 4.28 | 9.54 | 20.05 | 52.50 | 105.88 |
| 8 | 2.03 | 3.12 | 4.18 | 9.36 | 19.76 | 51.74 | 104.49 |
| 9 | 2.02 | 3.09 | 4.14 | 9.32 | 19.64 | 51.09 | 103.29 |
| 10 | 2.02 | 3.07 | 4.10 | 9.29 | 19.56 | 50.69 | 102.49 |
| 16 | 2.01 | 3.03 | 4.03 | 9.08 | 19.17 | 49.73 | 100.33 |
| 32 | 2.00 | 3.00 | 4.00 | 9.01 | 19.03 | 49.14 | 99.41 |

**Table 1: Number of surviving edges after pruning for randomly generated instances (averages over 100 runs).**

This pruning turns out to be extremely efficient in general. Table 1 shows the number of edges that survive for different values of $\lambda$ and $n$. Each entry is an average over 100 runs.

The extraordinary efficiency of pruning makes it possible to solve the problem of finding the best rotated MST much faster (especially for higher values of $\lambda$). It will on average require $O(n^2 \log n)$ time. However, it should be noted that it is possible to construct point sets of any size where the number of edges after pruning is $\Omega(n^2)$.

Table 2 shows the MST improvement for various values of $n$ and $\lambda$ where each entry is an average over 100 runs. The table contains two measures of improvement. The first one is calculated as $1 - \frac{|T_{\min}|}{|T_{\max}|}$ in percent. $T_{\min}$ is the shortest MST while $T_{\max}$ is the longest MST taken over 600 uniformly distributed values of $\alpha$ in the interval $[0, \omega[$. In the second measure of improvement the value $T_{\max}$ has been replaced by the length of the MST without rotating the points ($\alpha = 0$); a natural alternative.

## 5.2 Rectilinear Steiner Tree

We used GeoSteiner [5] to compute RSMTs for each of the $O(n^2)$ orientations given by the pairs of terminals (where $n$ is the number of terminals). The RSMT improvement obtained by rotating the coordinate system can be seen in Table 3. The values are percentages which express the improvement compared to not rotating at all. Results for both random and VLSI instances are given. For small problem instances the improvements are highest for randomly generated instances, while for large instances, the VLSI problems result in a higher improvement.

# 6. RELATED AND OPEN PROBLEMS

There are several other geometric combinatorial optimization problems which require a selection of a subset of edges and can in the rotational setting be approached in the same way as the MST problem. The travelling salesman problem and matching are probably the most well-known. Another straightforward generalization occurs when the orientations are fixed but not necessarily evenly spaced. Determination of rotated MSTs in higher dimensions seems also to require a straightforward generalization of the 2-dimensional case.

There are several research directions which deserve more attention in the future. First of all, it remains open if the rotated MSTs can be determined more efficiently by some direct methods that do not involve enumeration of a (limited) number of MSTs. Also, the problem of determining rotated SMTs for other than the rectilinear case is completely open.

# 7. CONCLUDING REMARKS

We addressed the problem of determining MSTs and rectilinear SMTs when edge directions are limited to uniformly distributed orientations and where the coordinate system is permitted to rotate by any angle. We suggested a simple polynomial algorithm to solve the MST problem. We also provided some computational results indicating how big the savings can be. As it could be expected, the savings become negligible when $\lambda$ and $n$ grows. On the other hand, for all practical applications, $\lambda$ is very small. Nets occurring in VLSI design are also rather small (in terms of the number of terminals involved). However, when many nets are to be routed, the overall savings will not be as impressive as for small isolated nets.

*Acknowledgments*

The authors would like to thank IBM and the Research Institute for Discrete Mathematics, University of Bonn, for providing industrial benchmarks circuits. The research was partially supported by the Danish Natural Science Research Council (contract number 56648).

# 8. REFERENCES

[1] M. Brazil, D. A. Thomas, and J. F. Weng. Minimum Networks in Uniform Orientation Metrics. *SIAM Journal on Computing*, 30:1579–1593, 2000.

[2] F. K. Hwang. On Steiner Minimal Trees with Rectilinear Distance. *SIAM Journal on Applied Mathematics*, 30:104–114, 1976.

[3] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.

[4] B. K. Nielsen, P. Winter, and M. Zachariasen. An Exact Algorithm for the Uniformly-Oriented Steiner Tree Problem. *Submitted*, 2001.

[5] D. M. Warme, P. Winter, and M. Zachariasen. GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), `http://www.diku.dk/geosteiner/`, 2001.

[6] P. Widmayer, Y. F. Wu, and C. K. Wong. On Some Distance Problems in Fixed Orientations. *SIAM Journal on Computing*, 16(4):728–746, 1987.

| | $n=3$ | | $n=4$ | | $n=5$ | | $n=10$ | | $n=20$ | | $n=50$ | | $n=100$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ | $T_{\max}$ | $\alpha=0$ |
| 2 | 21.86 | 14.48 | 17.42 | 9.67 | 15.86 | 8.38 | 10.29 | 4.89 | 7.47 | 3.19 | 5.03 | 2.12 | 3.46 | 1.51 |
| 3 | 8.86 | 5.34 | 7.38 | 4.34 | 6.37 | 3.69 | 4.31 | 2.52 | 3.01 | 1.47 | 1.70 | 0.78 | 1.23 | 0.61 |
| 4 | 5.55 | 3.48 | 4.32 | 2.50 | 3.81 | 2.14 | 2.37 | 1.31 | 1.72 | 0.93 | 1.02 | 0.51 | 0.72 | 0.39 |
| 5 | 3.40 | 2.10 | 2.85 | 1.66 | 2.42 | 1.39 | 1.59 | 0.89 | 1.09 | 0.58 | 0.72 | 0.39 | 0.47 | 0.25 |
| 6 | 2.36 | 1.46 | 1.88 | 1.17 | 1.55 | 0.97 | 1.05 | 0.67 | 0.73 | 0.39 | 0.44 | 0.21 | 0.30 | 0.16 |
| 7 | 1.76 | 1.12 | 1.41 | 0.86 | 1.21 | 0.74 | 0.78 | 0.42 | 0.57 | 0.32 | 0.35 | 0.19 | 0.24 | 0.14 |
| 8 | 1.36 | 0.85 | 1.11 | 0.66 | 0.94 | 0.53 | 0.61 | 0.33 | 0.44 | 0.24 | 0.25 | 0.13 | 0.18 | 0.09 |
| 9 | 1.04 | 0.66 | 0.80 | 0.49 | 0.73 | 0.43 | 0.49 | 0.28 | 0.30 | 0.16 | 0.20 | 0.11 | 0.15 | 0.08 |
| 10 | 0.88 | 0.52 | 0.70 | 0.40 | 0.58 | 0.31 | 0.40 | 0.21 | 0.26 | 0.13 | 0.18 | 0.09 | 0.11 | 0.05 |
| 16 | 0.32 | 0.20 | 0.26 | 0.16 | 0.22 | 0.14 | 0.16 | 0.10 | 0.11 | 0.06 | 0.07 | 0.03 | 0.04 | 0.02 |
| 32 | 0.08 | 0.05 | 0.07 | 0.04 | 0.06 | 0.04 | 0.04 | 0.02 | 0.03 | 0.01 | 0.02 | 0.01 | 0.01 | 0.01 |

**Table 2: MST improvement in percent in relation to two values: The first one is the length of the worst case MST ($T_{\max}$) which is the longest MST found among 600 uniformly distributed values of $\alpha$. The second one is the length of the MST when there is no rotation at all ($\alpha = 0$). All values are averages on 100 random instances.**

| | n=2 | n=3 | n=4 | n=5 | n=10 | n=20 | n=50 | n=100 |
|---|---|---|---|---|---|---|---|---|
| Random | 21.21 | 11.46 | 8.19 | 7.00 | 3.17 | 2.17 | 1.28 | 0.92 |
| VLSI | 14.61 | 7.01 | 5.96 | 7.62 | 4.88 | 2.90 | - | - |

**Table 3: RSMT improvement in percent obtained by rotating as compared to not rotating at all. Random: Randomly generated instances (averages over 100 runs). VLSI: VLSI design instances (averages over 100 runs).**

# Alternating Paths through Disjoint Line Segments

Michael Hoffmann and Csaba D. Tóth*
Institute for Theoretical Computer Science
ETH Zürich, CH-8092 Zürich, Switzerland
{hoffmann,toth}@inf.ethz.ch

## ABSTRACT

It is shown that every line segment endpoint visibility graph on $n$ disjoint line segments in the plane admits an alternating path of length $\Theta(\log n)$, answering a question of Bose [2, 3]. Moreover, we give bounds on the constants hidden by the asymptotic notation.

## 1. INTRODUCTION

Consider a set $S$ of $n$ disjoint obstacles, represented by line segments, in the Euclidean plane. A mobile agent wishes to visit a maximal number of vertices (i.e., segment endpoints) under various constraints. We are specifically studying simple paths where the agent moves along a straight line segment between two vertices, and it cannot cross a segment (although it may walk along a segment from one of its endpoint to the other). In contrast to the *traveling salesman problem*, the question is not the minimal distance necessary, but rather whether there exist a path through all vertices, or what is the maximal number of vertices that can be visited.

It has been recently shown [4] that there exist a simple Hamiltonian circuit for a set $S$ of $n$ disjoint line segment, if they are not all collinear. Urabe and Watanabe [7] gave a construction where $S$ does not always have a simple Hamiltonian circuit circumventing $\bigcup L$, i.e. a *circumscribing polygon*. Earlier, Rappaport [5] showed that $S$ does not always have a simple Hamiltonian circuit through all segments of $S$. Note that in such a circuit, every second segment must belong to $S$, because the circuit is simple, once an endpoint of an $\ell \in L$ is reached, the path must continue along $\ell$.

An *alternating path* is a simple polygonal chain where every other segment belongs to $S$ but none of the segments cross any segment in $S$. An *alternating circuit* is a closed alternating path.

It is easy to see that not every set of disjoint line segments admits an alternating Hamiltonian circuit (there is an example on three segments already). Rappaport [5] proved that it is NP-complete to decide whether a given set $S$ of line segments admits an alternating Hamiltonian circuit. But if each segment in $S$ has at least one endpoint on the convex hull of $\bigcup S$, one can decide in $O(n \log n)$ time whether an alternating Hamiltonian circuit exists [6]. We answer a question posed by Bose [2, 3]: how short is the longest alternating path that every set of $n$ disjoint line segments has? (The *length* of a path is the number of vertices visited).

**Theorem 1** *Let $S$ be a set of $n$ disjoint closed line segments in the plane. Then for any $s \in S$, there is an alternating path of length $2\log_2(n+1)$ passing through $s$.*

Apart from a constant factor, this bound is best possible: there are sets of disjoint line segments where the longest alternating path has length $\frac{12}{\log_2 3} \log_2 n - 17$, as we show in Section 4.

It is easy to turn our proof into an $O(n \log n)$ algorithm to compute an alternating path of length at least $2\log_2(n+1)$. But note that this path might not be the longest alternating path for the given set of line segments. The optimization problem might have much larger complexity.

We derive Theorem 1 as an easy consequence of the following.

**Theorem 2** *For any set $S$ of $n$ disjoint closed line segments and for any pair $s_1, s_2 \in S$, there exists an alternating path from $s_1$ to $s_2$.*

Our proof relies on a recent result which states that segment endpoint visibility graphs are Hamiltonian. It allows us to handle the alternating paths of disjoint line segments in an abstract setting: in a graph which is a union of a Hamiltonian circuit and a complete matching on a same set of $2n$ vertices.

## 2. PRELIMINARIES

*Supported by the joint Berlin-Zürich graduate program "Combinatorics, Geometry, and Computation", financed by the German Science Foundation (DFG) and ETH Zürich.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

Consider a set $S$ of $n$ disjoint closed line segments in the plane and denote by $P_S$ the set of the $2n$ segment endpoints. The *segment endpoint visibility graph* $Vis_S = (P_S, E_S \cup E_V)$ is defined as follows. Two nodes $u, v \in P_S$ are connected by a

- *segment edge*, if and only if the corresponding line segment $\overline{uv}$ is in $S$,

while $u$ and $v$ are connected by a

- *visibility edge*, if and only if the corresponding line segment $\overline{uv}$ does not cross any segment from $S$.

Two line segments cross if they have at least one common point which lies in the relative interior of both segments. Denote by $E_S$ the set of segment edges, and by $E_V$ the set of visibility edges. See the example in Figure 1(a) where the visibility edges are shown as dotted lines. A simple path $P = (p_1\, p_2, \ldots, p_k)$ in $Vis_S$ is called *alternating path* if it consists of segment edges and visibility edges in alternating order, or formally, $p_{2i-1}\, p_{2i} \in E_S$ for every $i = 1, \ldots, \lfloor k/2 \rfloor$ and $p_{2i}\, p_{2i+1} \in E_V$ for every $i = 1, \ldots, \lfloor (k-1)/2 \rfloor$.

It has been shown recently that segment endpoint visibility graphs are Hamiltonian [4]. Moreover, there is a Hamiltonian circuit $H_S$ in $Vis_S$ that corresponds to a simple polygon in the plane. But $H_S$ is not necessarily alternating, as it might contain two or more visibility edges in a row (Figure 1(b)). $H_S$ may even consist exclusively of visibility edges.

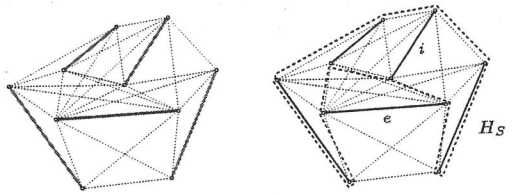



**Figure 1: Segment endpoint visibility graph and a Hamiltonian circuit**

A segment $s \in S$, which is not in $H_S$, is necessarily a diagonal of $H_S$. For example in Figure 1(b), the segment $i$ is an internal diagonal, while the segment $e$ is an external diagonal of $H_S$. The main idea of our proof is the following. We build an alternating path recursively starting with an arbitrary directed edge $\overrightarrow{e}$ of $H_S$. Follow $H_S$ in the direction of $\overrightarrow{e}$ until an endpoint $u$ of a segment diagonal $uv$ is reached. If there is no segment diagonal, then $H_S$ is an alternating circuit of length $2n$. At $u$, the path traverses the edge $uv$ and continues from $v$ along $H_S$ arbitrarily. Since the two edges incident to $v$ in $H_S$ are visibility edges (the segments in $S$ are disjoint), we obtain a path where segment edges and visibility edges alternate. Observe that this path has no self-crossings, because the polygon $H_S$ together with all segment diagonals is a planar graph.

We will show in the subsequent sections, that one can always produce a *simple* path of length $\Omega(\log n)$ in this manner.

# 3. THE LOWER BOUND

The key fact in our argument is stated in Lemma 1 below. The proof of Theorems 1 and 2 then follows by elementary graph theoretic arguments.

Denote by $D(H_S)$ the Hamiltonian cycle $H_S$ together with all its segment diagonals. Observe that $D(H_S)$ is planar and its maximal degree is three.

**Lemma 1** *For any directed edge $\overrightarrow{e}$ and any edge $f$ of $H_S$, there is a directed alternating path from $\overrightarrow{e}$ to $f$ in $D(H_S)$.*

Each vertex of $D(H_S)$ has degree 2 or 3. If $deg(v) = 2$ for some vertex $v$, then no segment diagonal is adjacent to $H_S$ at $v$, therefore $v$ is adjacent to a visibility edge and a segment edge. If $deg(v) = 3$ then $v$ is adjacent to a segment diagonal $uv$ and to two visibility edges along $H_S$.

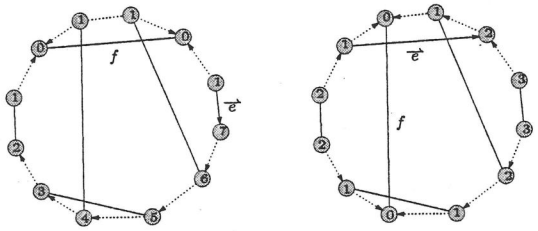



**Figure 2: The vertices can be labeled by distance to $f_0$.**

Let $\overrightarrow{e} = \overrightarrow{(e_0, e_1)}$ and $f = (f_0, f_1)$. $e_1$ and $f_1$ partition the circuit $H_S$ into two edge-disjoint arcs. Each vertex $v \in D(H_S) \setminus \{e_1\}$ has a well-defined distance $d(v)$ to $f_1$ along its proper arc in $H_S$, and let $d(e_1) = max\{d(v) : v \neq e_1\}$. Visibility edges of $D(H_S) \setminus e_1$ can be oriented such that an edge $uv$ is directed from $u$ to $v$, if $d(u) > d(v)$. Orient one visibility edge incident to $e_1$ from $e_1$ to its neighbor. (See an examples in Figure 2 where the vertices are labeled with $d(.)$). With respect to the described orientation, every vertex except for $f_1$ has at least one outgoing and at most one incoming visibility edge.

Let $G = (V, E)$ be an undirected graph and let $P = (p_1, \ldots, p_n)$ $(n \in \mathbb{N})$ be a path in $G$. For an edge $\{p_n, x\} \in E$ we write $P \circ x$ to denote the path $(p_1, \ldots, p_n, x)$.

**Proof.** (of Lemma 1) Let $R(\overrightarrow{e})$ be the set containing $e_1$ and all vertices of $D(H_S)$ that can be reached from $\overrightarrow{e}$ by alternating paths of both even and odd length (i.e. where the last edge is a segment edge or a visibility edge, respectively).

If an endpoint of $f$ is in $R(\overrightarrow{e})$ then the proof is complete. Assuming the contrary, let $y_0 \in R(\overrightarrow{e})$ be a vertex for which $\min\{d(r) : r \in R(\overrightarrow{e})\}$ is attained. Let $P_0$ be an alternating path from $\overrightarrow{e}$ to $y_0$ ending with a segment edge. We show that the path $P_0$ can be extended to an alternating path from $\overrightarrow{e}$ to $f$ by repeating the following step:

> *Given $P_i$ ending with $y_i$, let $\overrightarrow{y_i x_{i+1}}$ be the unique visibility edge leaving $p_i$, and let $x_{i+1} y_{i+1}$ be the*

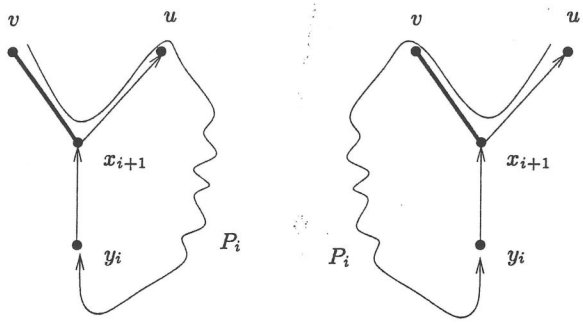



**Figure 3: The two possible orientation of $P_i$ through $x_{i+1}$.**

*unique segment edge incident to $x_{i+1}$. Put $P_{i+1} = P_i \circ x_{i+1} \circ y_{i+1}$.*

Clearly $x_1 \notin R(\overrightarrow{e})$. We show that $x_i \notin R(\overrightarrow{e})$ implies $y_i \notin R(\overrightarrow{e})$. If we assume $y_i \in R(\overrightarrow{e})$ then $x_i$ can be reached on both odd and even alternating cycles: $P_i - y_i$ and $P' \circ x_i$ where $P'$ is a path to $y_i \in R$ ending with a visibility edge.

We show next that $y_i \notin R(\overrightarrow{e})$ implies $x_{i+1} \notin R(\overrightarrow{e})$. If we assume $x_{i+1} \in R(\overrightarrow{e})$ then $y_i \in R(\overrightarrow{e})$ because it can be reached on both odd and even alternating cycles: $P_i$ and $P' \circ y_i$ where $P'$ is a path to $x_{i+1} \in R$ ending with a visibility edge.

We observe that $y_i \notin R(\overrightarrow{e})$ for all $i > 0$. It rests to prove that $P_i$ is a *simple* path for all $i$. Then it follows immediately that $P_i$ eventually reaches $f$, since in each step $P_i$ covers two more vertices.

Suppose that for a $P_i$, $i \geq 0$, the extension $P_{i+1}$ reaches itself, i.e., $x_{i+1} \in P_i \setminus R(\overrightarrow{e})$. (Observe that if $y_{i+1}$ is a common point of $P_{i+1}$ and $P_j$, $j \leq i$, then already $x_{i+1}$ is a common point of the two paths.) Clearly, $deg(x_{i+1}) = 3$. Vertex $x_{i+1}$ is incident to the visibility edge $y_i x_{i+1}$, denote the second incident visibility edge by $x_{i+1}u$ and the unique segment edge by $x_{i+1}v$. (See Fig. 3.) By the choice of $x_{i+1}$, edge $\overrightarrow{y_i x_{i+1}}$ enters $x_{i+1}$. Hence $\overrightarrow{x_{i+1}u}$ leaves $x_i$ and $P_i$ passes through $u, x_i$, and $v$ in this order, i.e., $x_{i+1} = y_j$, $0 < j \leq i$.

Here, $j \neq i$ because $x_{i+1} = y_j$ and $y_j \neq y_i$, hence $i > 0$. This implies that $y_i$ can be reached from $\overrightarrow{e}$ by an odd and an even alternating path: $P_i$ ending with a segment edge and $P_j \circ y_i$ ending with a visibility edge. This gives $y_i \in R(\overrightarrow{e})$, contradiction. □

It is worthwhile to note here that we do not use in the above proof that $D(H_S)$ is planar. The proof of Theorem1 and 2 is completed by the following, elementary, argument.

**Lemma 2** *Let $G = (V, E)$ be a planar subgraph of a segment endpoint visibility graph $Vis_S = (P_S, E_S \cup E_V)$ such that $E \supset E_S$ and the maximum vertex degree in $G$ is $B \in \mathbb{N}$. If for any two segment edges $s, t \in E_S$ there is an alternating path from $s$ to $t$, then there is a an alternating path of length $2\log_{B-1}((B-2)n+1)$ starting from any segment edge $s \in S$.*

We can, basically, repeat the standard existence proof of paths of length $\log_{B-1}((B-2)n+1)$ in a connected graph with $n$ nodes if the maximal degree is $B \in \mathbb{N}$.

**Proof.** For a directed alternating path $P$ starting with a segment edge $\overrightarrow{e}$ and ending with another segment edge, let $f(P, \overrightarrow{e})$ denote the number of segment edges $t$ such that there is a directed alternating path from $\overrightarrow{e}$ to $t$ whose initial section is $P$.

The path $P$ can be extended by a visibility edge of $G$ in less than $B$ different ways, and in each case a unique segment edge follows. The possible resulting paths are not all necessarily alternating, since some vertices of $P$ could be reached again. By all means, summing for all possible alternating path extensions $R$ of $P$ we get

$$\sum_{R, \text{ s.t. } R \text{ extends } P} f(R, \overrightarrow{e}) \geq f(P, \overrightarrow{e}) - 1 .$$

The $-1$ term appears, because we cannot reach the last edge of $P$ by any extension. The direction of $e$ can be chosen such that $f\left(\left(\overrightarrow{e}\right), \overrightarrow{e}\right) \geq n - 1$, since any segment can be accessed from $\overrightarrow{e}$ on an alternating path.

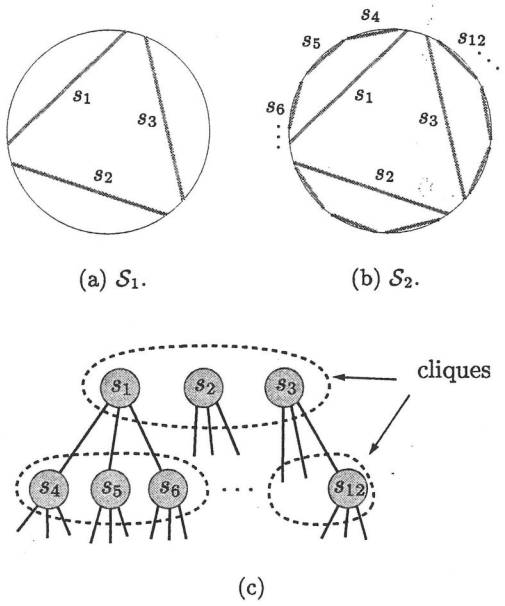
Each time we extend the path $P$ by a new edge, $f(P, \overrightarrow{e})$ decreases by at most a factor of $B$. The lower bound $\lceil \log_{B-1}((B-2)n+1) \rceil$ on the number of segment edges on this path follows from the depth of a balanced $(B-1)$-ary tree. □

## 4. UPPER BOUND

Complementing the results from the previous section, we will now show an asymptotically matching lower bound, that is construct sets $\mathcal{S}_k$, $k \in \mathbb{N}$ of line segments that do not have long alternating paths.

**Theorem 3** *There are sets of $n$ disjoint line segments in the plane that do not admit an alternating path of length greater than $\frac{12}{\log_2 3} \log_2 n - 17 < 7.57 \log_2 n$.*

**Proof.** Let $\lambda_k$ be the length of a longest alternating simple path in $\mathcal{S}_k$. The set $\mathcal{S}_k$ is constructed recursively as follows. All line segments are chords of a circle $c$. $\mathcal{S}_1$ consists of three segments arranged in a triangular fashion, i.e. such that $Vis_{\mathcal{S}_1} \cong K_6$. The endpoints of the chords partition $c$ into arcs. $\mathcal{S}_k$ is obtained from $\mathcal{S}_{k-1}$ by inserting a sequence of three segments (i.e. a copy of $\mathcal{S}_1$) on every arc of $c$ that is bounded by only one segment from $\mathcal{S}_{k-1}$. Figure 4 shows $\mathcal{S}_1$ and $\mathcal{S}_2$.




(a) $\mathcal{S}_1$. (b) $\mathcal{S}_2$.

(c)

**Figure 4: The construction of $\mathcal{S}_k$.**

Note that there is no difference in visibility between the two endpoints of any segment in $\mathcal{S}_k$. If one considers the segments only, one can interpret $\mathcal{S}_k$ as complete ternary tree of depth $k-1$ where each node is formed by a clique of three segments (Figure 4(c)). Since the longest simple path in a tree of depth $k$ has length $2k+1$ (remember that we count the number of vertices) and since visiting a 3-clique of segments means visiting 6 vertices, we can conclude that $\lambda_k = 12k - 6$.

$\mathcal{S}_k$ contains exactly $n_k := 3^{k+1} - 3$ vertices. Hence,

$$\begin{aligned}\lambda_k &= 12\left(\log_3 n_k - 1 + \log_3 \frac{3^k}{3^k-1}\right) - 6 \\ &= \frac{12}{\log_2 3}\log_2 n_k - 18 + 12\log_3 \frac{3^k}{3^k-1}\end{aligned}$$

and the claimed result follows, since the last term is less than one for $k \geq 3$. □

Note that an $\Omega(\log n)$ bound was already known due to Urrutia [1], but it is unpublished, and we do not know about the precise constants achieved.

## 5. OPEN QUESTIONS

There is a straightforward way to find a Hamiltonian polygon for the lower bound construction from the previous section. Visiting vertices along the circle $c$ gives a *circumscribing* polygon (i.e., where all segments are sides or internal diagonals). We note here that it is considerably easier to establish our Theorem 1 for sets of segments which have a circumscribing polygon.

We note also that maximal alternating paths are in the graph $D(H_S)$ for this example. Our algorithm described in the proof of Lemma 2 always gives a path of length at least $4\log_2(n+3) - 7$, starting form a segment $s$. Actually, for $(n+3)/2$ segments, it outputs an alternating path of maximum length.

An evident question is: what is the exact length of a maximal alternating path one can find for any set of disjoint line segments. We could show that it is between $2\log_2(n+1)$ and $7.57\log_2 n - 17$. It is not clear that our approach in its abstract setting cannot give the exact result.

Let $H$ and $M$ be a Hamiltonian circuit and a complete matching on the same set $V$ of $2n$ vertices. What is the longest path in the abstract graph $(V, H \cup M)$ in which every second edge belongs to $M$. For this problem, we know only the same bounds as for alternating paths in the segments endpoint visibility graph so far.

## 6. REFERENCES


[1] BOSE, P. personal communication, August 2001.

[2] BOSE, P., HOULE, M. E., AND TOUSSAINT, G. Every set of disjoint line segments admits a binary tree. In *Proc. 5th Annu. Internat. Sympos. Algorithms Comput.* (1994), vol. 834 of *Lecture Notes Comput. Sci.*, Springer-Verlag, p. ??

[3] DEMAINE, E. D., AND O'ROURKE, J. Open problems from CCCG'99. In *Electronic Proc. 11th Canadian Conf. on Comput. Geom.* (Vancouver, 1999).

[4] HOFFMANN, M., AND TÓTH, CS. D. Segment endpoint visibility graphs are Hamiltonian. In *Proc. 13th Canad. Conf. Comput. Geom.* (Waterloo, 2001). to appear.

[5] RAPPAPORT, D. Computing simple circuits from a set of line segments is NP-complete. *SIAM J. Comput. 18*, 6 (1989), 1128–1139.

[6] RAPPAPORT, D., IMAI, H., AND TOUSSAINT, G. T. Computing simple circuits from a set of line segments. *Discrete Comput. Geom. 5*, 3 (1990), 289–304.

[7] URABE, M., AND WATANABE, M. On a counterexample to a conjecture of mirzaian. *Comput. Geom* **2**, 1 (1992), 51–53.

# A simple kinetic visibility polygon

Samuel Hornus
iMAGIS-GRAVIR/IMAG-INRIA
Samuel.Hornus@imag.fr

Claude Puech
iMAGIS-GRAVIR/IMAG-INRIA
Claude.Puech@imag.fr

## ABSTRACT

*Given a set of moving obstacles in the plane, we propose a method for maintaining efficiently the visibility polygon of a (possibly moving) viewpoint. We consider both smooth-convex, and simply-polygonal obstacles.*

## Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Non-numerical Algorithms and Problems*

## Keywords

kinetic data structure, visibility polygon

## 1. INTRODUCTION

Visibility computations are central in many computer graphics algorithms and in robot motion planning. A very useful tool is the determination of the objects visible from a viewpoint. Thus, in the plane, the visibility polygon is an important visibility structure. It is a star shaped polygon centered at the viewpoint, whose edges are the visible parts of the objects in the scene, and whose interior intersects no object.

Efficient algorithms exist to compute the visibility polygon in the static case, but applications of this problem may apply to moving objects. Computing the visibility polygon at various times on a static " snapshot " of the scene is inefficient, since we do not take into account the temporal coherence that arise from the continuity of the movements of the objects (and possibly the viewpoint): if the time step is too small, we will compute many times the very same (combinatorial) visibility polygon.

We use the *kinetic data structure* framework introduced by Guibas [2, 1] to propose a *simple* algorithm that maintains the visibility polygon of a view point in a 2-dimensional scene when all objects may move. The structure maintained is in fact a weak radial decomposition of the scene. Section 2 treats the case of smooth convex obstacles. Section 3 examines the case of simple (non auto-crossing) polygons.

Kinetic data structures (KDS) are a way to efficiently and accurately maintain an attribute built on top of continuously moving items (e.g. a convex hull). In order to maintain an attribute $\mathcal{A}$ over a set of moving items (items are generally points), each test in the proof of correctness of the construction of $\mathcal{A}$ is analyzed to detect the time at which it will fail. The idea is that maintaining the validity of all those tests (called *certificates*) guarantees that the attribute $\mathcal{A}$ is maintained also, since the certificates provide a proof of correctness. Certificates are ordered in a priority queue, according to their failure time. When the simulation time passes above the first certificate's failure time, the attribute is modified, and the proof is updated (i.e. some certificates disappear, others are created, and their failure time is computed). This method only requires that the motion of the items be known in the short term. For short, one could say that kinetic data structures get rid of step-by-step simulations, and implement in fact time sweep algorithms.

We now get interested in maintaining the visibility polygon of a scene. Here, in the KDS terminology, the items are the convex smooth objects, or the polygons' vertices. We maintain a weak radial decomposition of the scene, thus, the certificates we use take care of the well ordering (i.e. cyclically sorted) of the segments in the decomposition. Finally the radial decomposition of the scene allows us to quickly build the visibility polygon.

## 2. CONVEX OBSTACLES

Let $\mathcal{O}$ be a set of $n$ convex obstacles in the plane. Let $\mathcal{F}$ be the " free space ": the complement of the union of the obstacles in the plane. Let $V$ be a point in $\mathcal{F}$. We aim at maintaining the visibility polygon of $V$ when $V$ and elements of $\mathcal{O}$ move in the plane. We assume we can compute in constant time the visibility tangents of an obstacle, that are defined as the two tangents to the object passing through the view point $V$. Let $\mathcal{T} = \{t_0, t_1, \ldots, t_{2n-1}\}$ the t_uple of the visibility tangents, sorted in the counter-clockwise order.

Let $u \in \mathcal{S}^1$ be a direction. We denote by $V(u)$ the obstacle seen by $V$ in the direction $u$. $V(u)$ can possibly be the " blue sky " that we denote $\infty$. One important observation is that $V(u)$ is constant between two consecutive visibility tangents. Thus, a way to define the visibility polygon around $V$, is to

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

see it as the function $\mathcal{P} : \mathcal{T} \mapsto \{\mathcal{O} \cup \infty\}$ so that $\mathcal{P}(t_i) = V(t_i^+) = V(t_{i+1}^-)$ where $t_i$ is seen as a direction pointing away from the view point $V$.

## 2.1 Kinetic visibility polygon

The visibility polygon (VP) changes only when two visibility tangents (VT) cross each other (but two VTs may cross each other without affecting the VP). Thus, we can maintain the VP by detecting when two consecutive VTs will cross, then updating the VP according to the kind of both VT, and swapping the two VTs involved to keep them sorted in counter-clockwise order.

However, having computed the VP at a given time is not sufficient to maintain it efficiently when obstacles move. We need some additional data that will have to be maintained also: for each VT $t_i$, we maintain its *hit-item*, which is the obstacle that is hit by the VT beyond the tangency point (it can be $\infty$). In fact, we maintain a weak radial decomposition of the scene, where only the far object hit by a VT is recorded and not the near object.

For each crossing, the update of the visibility polygon is done in constant time, by distinguishing 8 cases. First, we need to characterize the VTs. Half of them will be *Left* if (seen from $V$), they pass to the left of the obstacle. The other half will be *Right* visibility tangents.

We explain the naming of the crossing events with an example. Figure 1a shows an $\widehat{L}R$ and an $\widehat{L}L$ crossing events (from left to right, the figure presents the obstacles involved in the event, just before, " during " and after the crossing). $\widehat{L}R$ means that the first VT (in counter-clockwise order) is a *Left* tangent, the consecutive VT is a *Right* tangent, and the hat on $\widehat{L}$ means that, when the crossing occurs, the tangency point of the *Left* tangent is farther from $V$ than the tangency point of the *Right* tangent. Hence, the 8 cases are named $L\widehat{L}$, $\widehat{L}L$, $L\widehat{R}$, $\widehat{L}R$, $R\widehat{L}$, $\widehat{R}L$, $R\widehat{R}$, $\widehat{R}R$.

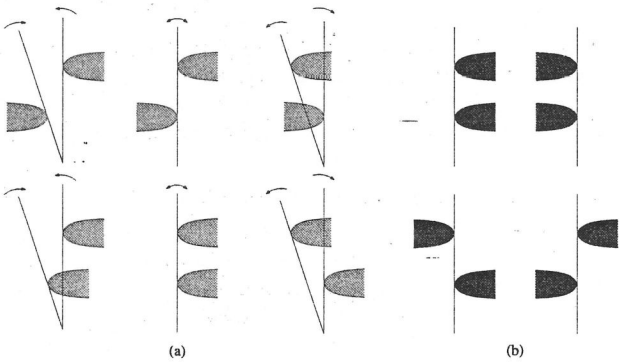



**Figure 1: (a) example of a $\widehat{L}R$ event (up) and a $\widehat{L}L$ event (down); (b) all events**

Now we need to update the VP and the *hit-items* when the $\widehat{L}R$ crossing occurs, see Figure 2. The VTs that cross each other are consecutive in our t_uple of ordered VTs. Let $t$ be the time at which the crossing occurs. Then at times $t^-$ and $t^+$, no other VT can lie between the two VTs we are interested in. Therefore we can be sure that any other obstacle (different from G or D in Figure 2) either completely crosses the angular section E, or has no intersection with it. This ensures the correctness of the update process.

First we check if an object C exists between the two points of tangency at time $t$. To do so, we just need to check whether `g.hit-item` (the hit-item of the *Right* tangent of obstacle G) is the same as `d.hit-item` or not. If so, then C does not exists, else, C exists. Note that `{g,d}.hit-item` can be $\infty$. This information is enough to update the *hit-items*.

Seen from $V$, the foremost obstacle among those in the figure is G. Therefore, if there is a change in the visibility for E, this change makes G visible. To know whether G becomes the visible object in E, we simply check whether there exists another obstacle in front of G at time $t$, by comparing the obstacle visible in E to C or S (depending on the existence of C).

The algorithm is described in Figure 3.

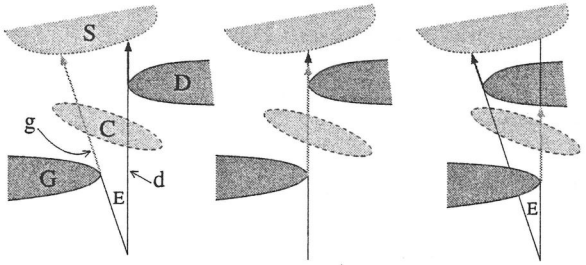



**Figure 2: Update of a $\widehat{L}R$ event, the arrows point to the *hit-items* in the case C does not exists**

The seven other cases are processed in the very same way, by just changing the roles in Figure 3.

```
g.tangent-item == G;
d.tangent-item == D;
--------------------
if ( G.hit-item == S ) {
    // C does not exists
    g.hit-item = d.tangent-item;
    if ( E.vis-item == S )
        E.vis-item = g.tangent-item;
}
else { // C exists, C == g.hit-item
    if ( E.vis-item == g.hit-item )
        E.vis-item = g.tangent-item;
}
```

**Figure 3: Processing an $\widehat{L}R$ event**

We know how to maintain the weak radial decomposition of our scene. Without more work, we can compute the visibility polygon only in linear time, by looking at the visible item between each visibility tangent, and " merging " the same consecutive values. This is not very efficient, but we easily remove this problem by performing a first " run-length-encoding " of the consecutive visible items, and by maintaining this encoding each time the VP is changed when a crossing-event occurs. This is done in constant time.

## 2.2 Complexity

We express the complexity of this kinetic data structure using terms proposed by Guibas and Basch [2, 1]. Our data structure is optimal in size since it is linear in the size of the scene (the set of all obstacles). It is *responsive*, meaning that the cost of processing a certificate failure is small: constant-time in our case. This KDS is *local*, meaning that the number of certificates that involve a single object is small; it is $O(1)$ in our case, with max. 4 certificates per obstacle.

However, it is not optimal since we may have to update many certificates in a move while none of these affect the visibility polygon. Imagine lots of small discs vertically aligned above a big disc, and the view point traversing the plane horizontally under the big disc. Using Guibas and Basch terminology, our KDS is not *efficient*, since the total number of events processed may be of a higher order as the number of changings in the VP. An optimal algorithm would update as many certificates as there are changes in the visibility polygon during the animation. Hall-holt and Rusinkiewicz [3, 4] propose such an algorithm, but are limited to convex smooth obstacles. They do process only one certificate failure for each change in the VP, but the cost of processing one event is not constant in time. However, the overall cost of processing all events for a simple motion (of the observer only) is significantly better in their algorithm.

# 3. SIMPLE POLYGONAL OBSTACLES

We now present an adaptation of the method to simple polygonal obstacles. A simple polygon can be concave, but none of its edges cross one another. We consider that obstacles in the initial set are in general position, meaning that no pair of vertices is aligned with the observer $V$.

The basic idea is the same. For each vertex $v$, we keep track of the ray starting at $V$ and passing through $v$, which makes, by a slight abuse of language, our new visibility tangent. For each vertex (or VT, since vertices and VTs are in bijection) we also keep track of its *hit-item* and of the type of the vertex. Here, the type of a VT is a bit more complex: Figure 4 shows how we name the type of a vertex depending on the position of both its adjacent edges relatively to the VT passing through it.

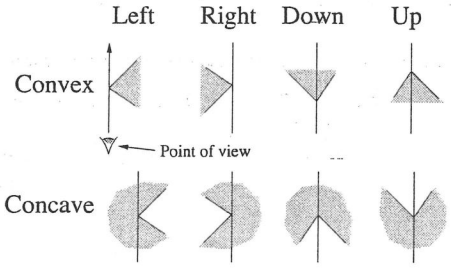



**Figure 4: Various types of vertices**

Note that the *hit-item* of a vertex $v$ can be irrelevant (and even wrong) if the part of its VT beyond $v$ goes through the interior of the adjacent polygon of $v$. It is yet correctly updated when the VT gets anew in free space.

The certificates that must stay true are the same as in the case of convex smooth obstacles: we schedule a crossing event when two consecutive vertices (which are kept sorted in counter-clockwise order around the view point) get aligned with the observer $V$. In Section 2, both VTs of the same obstacle could not cross each other. This is not the case here, since two consecutive vertices (consecutive in the cyclic order and on a polygon boundary) can get aligned with $V$.

Thus, we have two kinds of update when a certificate fails. If the certificate concerns vertices consecutive on the border of a polygon, then we have to update their type, and possibly their *hit-items* and the VP; this again, is done in constant time. In other cases, the update is similar to those in Section 2, with some more cases because we have to take into account other types for a vertex, namely *Up* and *Down*.

Figure 5 shows how the type of a vertex changes when $V$ crosses the supporting line of one of its adjacent edges. Edges of a polygon are oriented so that the inside of the polygon lies to the left of its edges. When the crossing occurs, one vertex is nearer to $V$ than the other: it will be said *near*, and the other, *far*; one vertex is following the other on the polygon's boundary: it will be said *next*, and the other, *prev*. This is the terminology used in Figure 5 to decide which transition we should target.

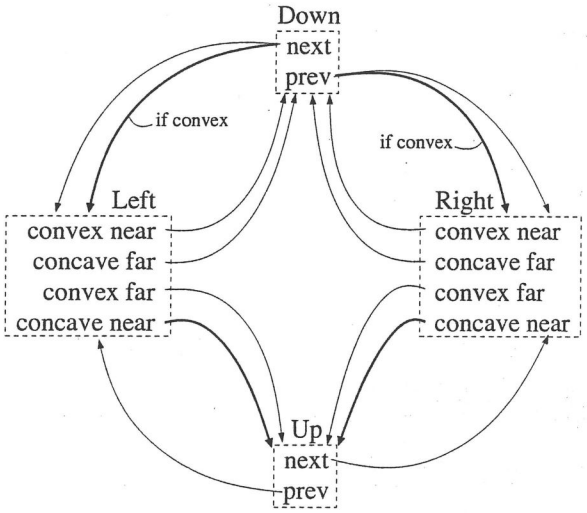



**Figure 5: Updating a vertex's type. Fat arrows represent transitions where *hit-item* must be updated**

The complexity of the structure for polygonal objects is the same as for convex smooth objects.

Hall-Holt proposes a refinement of this method, which processes exactly as many events as there are changes in the VP. The cost of processing one event is larger, but for the same motion of all items (obstacles and the viewpoint), his algorithm is less costly in time because of a relaxation on the constraints on the shape of the scene decomposition. However, he designed his algorithm only for convex obstacles.

The algorithm proposed by Hall-Holt can in fact be adapted to simple polygonal obstacles using a radial decomposition

of the polygonal scene where each edge of a polygon is considered separated of the others, so that the radial segments would lie " in " the polygons as well, and not only in free space.

## 4. CONCLUSION AND FUTURE WORK

We have presented a simple kinetic data structure that maintains the visibility polygon of a moving point in a planar scene of moving obstacles (convex-smooth or simply-polygonal). A change in the visibility polygon is processed in constant time. The size of the structure is optimal (linear in the size of the scene). However it processes too many events: among all the events processed, lots can have no effect on the VP. However the number of events processed remains optimal if the scene is sparse, because nearly all obstacles become visible.

These algorithms could perhaps be accelerated by representing polygons with various level of details (perhaps even aggregating polygons that are closed to each other and far away from the view point), and using a sufficiency criteria to increment or decrement the LOD for some (groups of) polygons.

We may also want to describe a 3-dimensional visibility polyhedron, and extend it to the KDS framework. This looks much more difficult than in the 2d case.

## 5. REFERENCES


[1] J. Basch. *Kinetic Data Structures.* PhD thesis, Stanford University, June 1999.

[2] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th Symposium on Discrete Algorithms (SODA'97)*, pages 747–756, 1997.

[3] O. Hall-Holt. Kinetic visible set maintenance in the plane. submitted for publication, 2001.

[4] O. Hall-Holt and S. Rusinkiewicz. Visible zone maintenance for real-time occlusion culling. submitted for publication, 2001.

# A kinetic view of the shooter problem

Jerzy W. Jaromczyk
Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA
jurek@cs.uky.edu

Mirosław Kowaluk*
Institute of Informatics
Warsaw University
Warsaw, Poland
kowaluk@mimuw.edu.pl

## ABSTRACT

The *generalized shooter* in $R^2$ is an optimization problem in the family of the transversal problems, with the objective to locate a point (shooter) such that the number of half-lines (rays) starting at this point needed to stab the $n$ given line segments (targets) is minimized. We cast this problem in a kinetic fashion where we are interested in the number of rays necessary for shooting at any given point for the shooter moving along an arbitrary continuous path. It appears that this is equivalent to maintaining the size of a minimum clique cover ($MCC$) for the intersection graph of a set of moving arcs on a circle. By designing a new data structure we are able to follow the topological changes affecting the size of the $MCC$ of this graph in time $O(\log^2 n)$ per event. This leads to $O(n^4 \log^2 n)$ solution to the shooter problem, which is the best up to date.

## 1. INTRODUCTION

The transversal problems, that is, intersecting a family of objects such as segments or polygons with a line, lines, or half-lines, play an important role both in theoretical and applied computational geometry.

In this paper we study a related problem called a *generalized shooter problem*. The problem, introduced in [NMB96] as an extension to questions studied in [LSW90,NMB96,WZ97], has a simple geometric formulation.

Given is a set $S$ of segments in $R^2$ and a point $p$ we say that a half-line starting at $p$ shoots at $s$ in $S$ if it intersects $s$. A set of half-lines shoots at $S$ if every segment in $S$ is shot at by some half-line in this set. The objective of the shooting problem is to find $sn(S,p)$, the minimum number of half-lines starting at $p$ that shoot at $S$. In the generalized shooter problem we ask for $sn(S) = \min_p\{sn(S,p)\}$ and/or for the locus of points $p$ that minimize this number of the shooting half-lines for $S$.

*Partially funded by grant KBN 8T11C03915

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

This problem has been intensively studied in recent years and there is a sequence of algorithms with incremental improvements in the asymptotic complexity.

An $O(n^5 \log n)$ deterministic algorithm was presented by Wang and Zhu [WZ] who also gave an $O(n^2)$ 2-approximation solution (the number of shots at most twice the optimum). Chaudhuri and Nandy [CN99] presented an improved solution with an $O(n^5)$ pessimistic time but with a better practical performance. Another improvement was provided recently by Katz, Nielsen and Segal [KNS00], who designed a $(1+\epsilon)$- approximation solution with an $O((n^4 \log n)/(\epsilon^3 r^*))$ worst-case running time; $r^*$ is the number of optimal solutions. The asymptotic cost of the algorithm depends on the quality of the approximation.

Here we present an $O(n^4 \log^2 n)$ deterministic solution to the generalized shooter problem. The algorithm is capable of finding all of the optimal positions and its cost is within a poly-logarithmic factor to the number of $O(n^4)$ possible solutions in the worst case. The solution is via a kinematic view of the problem and its connection to the Minimum Clique Cover ($MCC$) for arc-graphs. Specifically, we will show how to efficiently maintain the information about the shooting number when the shooter moves on the plane.

Although a connection between the $MCC$ and the shooter problem has been known for some time, its utilization was limited to a direct application of the Hsu algorithm [HT91] for finding $MCC$ of a static arc-graph. As one of the main contributions of this paper, we show how to maintain the size of the $MCC$ when the arcs move in a continuous fashion on the circle, which effectively leads to a data structure that we call *kinetic arc-graphs*. In addition to the generalized shooter problem our method applies to efficient solutions for related problems such as a *dynamic unit-arc graphs* and a *shooting to half-transparent segments* that we will sketch in the last section. Furthermore, the same data structure of kinetic arc-graphs can be used to maintain the maximum independent sets and the minimum dominating sets in the circular arc-graphs induced by a point moving among the segments in $S$.

## 2. KINETIC ARC-GRAPHS

The relation between the shooter problem and arc-graphs is intuitive and simple. A shooter positioned at point $p$ viewes the segments as angular ranges. These ranges can be represented as arcs on a small enough circle centered at $p$.

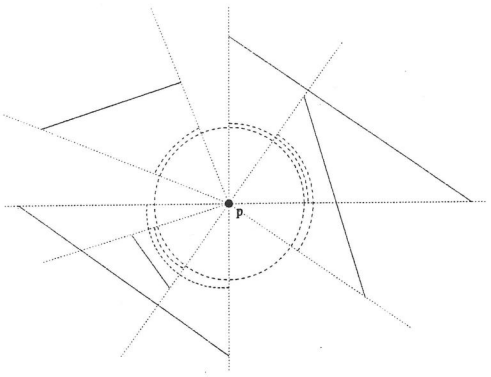



**Figure 1: A set of segments and the corresponding set of arcs.**

An example is illustrated in Figure 1.

DEFINITION 2.1. *A circular arc-graph $Arcs(S)$ is a graph whose veritices correspond to the arc projections of the segments in $S$ projected on the circle centered at $p$. Edges in $Arcs(S)$ correspond to overlapping arcs is called a* circular arc-graph.

A set of nodes $V^*$ in $Arcs(S)$ is said to form a *linear clique* if all the corresponding arcs have a common point. So for example, a set of three evenly distributed arcs of length $2\pi/3 + \epsilon$ that is a clique is not a linear clique in the corresponding graph. The importance of linear cliques to our shooting problem is obvious; all of the segments whose arcs participate in such a clique can be shot with just one half-line.

We need a few definitions and results from [HT91] regarding circular arc-graphs.

Let us order a set $F$ of arcs on a circle in the clockwise orientation by their starting endpoints. Then arc $i$ can be represented by a pair $(h(i), t(i))$, where $h(i)$ is the head (beginning) and $t(i)$ is the tail (end) of the arc. Hsu and Tsai [HT1] defined on $F$ a function $next(.)$ and an independent set of arcs $GD(.)$.

DEFINITION 2.2 (HT91). *For each arc $i$, define $next(i)$ to be arc $j$ in $F$ whose head is contained in $(t(i), h(i))$ and whose tail is the first encountered tail in a clockwise traversal from $t(i)$. Define $GD(i)$ to be the maximal independent set of the form $\{i_1(= i), i_2, ..., i_k\}$, where $i_t = next(i_{t-1})$, $t = 2, ..., k$. Finally, $last(i)$ is defined as $next(i_k)$ (clearly, $last(i)$ overlaps $i$).*

Function $next(.)$ defines a directed graph whose components are cycles with attached trees (the out-degree for each node is 1). For that reason the graph's components are called $c$-trees. The cycles in $c$-trees play an important role in finding the $MCC$ as it will be explained below. An example presenting a set of arcs, and the corresponding circular arc-graph and $c$-trees is presented in Figure 1.

Denote by $LQ(i)$ the (maximal) linear clique consisting of $i$ and all arcs that contain $t(i)$.

Furthermore, let $i$ be an arc in $F$ whose node is in a cycle in the $c$-trees for $F$. Let $\alpha(F)$ denote the size of a maximum independent set in the circular-arc graph for $F$ and $GD(i) = \{i_1(= i), .., i_{\alpha(F)}\}$. Furthermore, the greedy clique cover $GD_q(i)$ is defined to be $\{LQ(i_1), ..., LQ(i_{\alpha(F)}\}$ if $last(i) = i$ and $\{LQ(i_1), ..., LQ(i_{\alpha(F)+1}\}$, otherwise.

Hsu and Tsai [HT91] prove the following theorem that we present here with our notations.

THEOREM 2.1 (HT91). *Assume that the set of arcs $F$ on a circle is not a clique. Let $i$ be any arc in any cycle in the $c$-trees for $F$. Then $GD_q(i)$ is a $MCC$ for $F$. Moreover, it can be found in an $O(n)$ time assuming that the circular arc-graph with ordered arcs is given.*

The significance of the above theorem is in providing a connection between a cycle in the $c$-tree and the size of the minimal (linear) clique cover. Translated into simple terms, in order to find the $MCC$, first we need to identify a sequence of arcs whose chain of the *next* relations forms a cycle, that is, a sequence of the arcs that wraps around the circle once or more times. Then, in such a sequence we want to find the longest segment of independed arcs, that is, to stop before the arcs start wrapping around. The size depends on the *next* value for the last element in the sequence. This is the essence of the algorithm.

Note that $F$ is a non-linear clique in a very special case.

COROLLARY 2.1. *The solution to the shooting problem for a given position $p$ of the shooter is equivalent to finding the $MCC$ in the $Arcs(S)$.*

## 3. SHOOTING ALGORITHM

In this section we will show the main ingredients of the algorithm leaving details to the full version of the paper.

Let $\mathcal{A}(\mathcal{S})$ be the arrangement induced by lines passing through the endpoints of the segments in $S$. The sizes of the $MCC$ are the same for all points belonging to the same cell as the corresponding circular arc-graphs are isomorphic.

Hence, the general shooting problem is equivalent to finding the smallest $MCC$ over the entire arrangement. Direct application of Corollary 2.1, as it was done in the previous papers on shooting, leads to an $O(n^5)$ solution. We will present an efficient representation for $c$-trees that will allow us to maintain the size of the $MCC$ with a poly-logarythmic cost per event when moving from a cell to an adjacent cell in a continuous motion. When the point moves between adjacent cells in the $\mathcal{A}(\mathcal{S})$ there are changes in the *next* function that lead in turn to two kinds of changes in the underlying $c$-trees: a subtree is reattached from one node to another, and the nodes in the cycle (as well as the cycle size) change, see Figure 2. Monitoring these changes and updating $c$-trees require efficient data structures. Changes in *next* will be maintained in an interval tree-like structure, and changes in

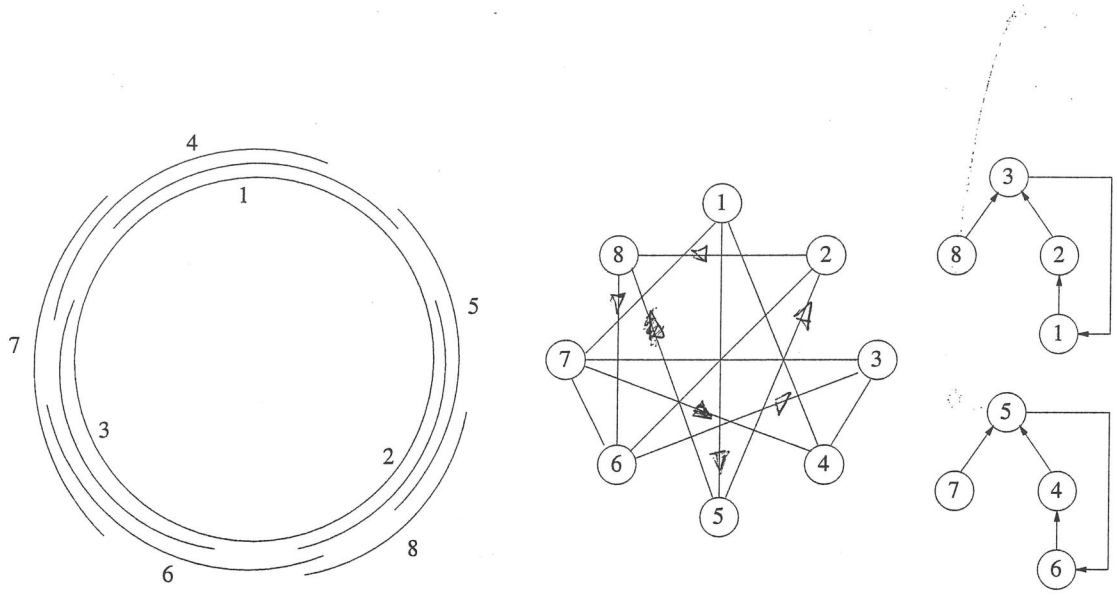



**Figure 2: A set of arcs, the corresponding circular arc-graph and $c$-trees**

the $c$-tree will be maintained in a data structure developed for this paper and called a *ptb*-tree.

A *pbt*-tree is built on a $c$-tree and it consists of two structures that are meshed together:

- Each node in the $c$-tree is replaced with a $B$-tree of its children. This will allow us to change in the logarithmic time pointers between children and their parents while updating the path decomposition (see below).
- A path decomposition of paths in the $c$-tree. Each path is organized as a $B$-tree and links nodes into longest paths. It is constructed as follows: while moving upwards from the leaves to the root of the $c$-tree, at each node we continue the longest path of its children and terminate all other paths reaching this node. This decomposition allows us to quickly find the length of the cycle in the $c$-tree.

A *ptb*-tree is illustrated in Figure 4. The size of the *ptb*-tree is $O(n)$. Although we are not using randomization it is useful to think that the paths in the *ptb*-tree are organized similarly to skip lists [Pu90].

Our solution is based on the following elements that we will list below.

1. To find $sn(S)$ it is sufficient to visit the arrangement $\mathcal{A}(\mathcal{S})$ induced by the lines passing through the endpoints of $S$ and to maintain the current value of the $MCC$. The size of the arrangement is $O(n^4)$.

2. The size of the $MCC$ can be found based on the $c$-tree by analyzing the cycle in this $c$-tree.

3. There is an efficient representation for $c$-trees, called a *ptb*-tree that supports the following three operations (it helps to think about *ptb*-tree as a rooted tree with the root pointing to some internal node):

The semantics of Jump$(v, k)$ is to find node $v'$ between $v$ and the root that is $k$ levels above $v$.

The semantics of JumpToRoot$(v)$ is to find the distance in $c$-tree from $v$ to the root.

LEMMA 3.1. *The worst-case complexity of operations* Reattach *is* $O(\log n)$ *and the cost of* Jump *and* JumpToRoot *is* $O(\log^2 n)$.

The operations will be used to modify the $c$-tree after changes to the *next* value of some arcs, to monitor the length of the cycle, and to quickly move from one node to another. One of useful and quite tricky applications of Jump is to verify if a given node $v$ is located on the cycle in a given $c$-tree. To this end, we store the length $sc$ of the cycle, computed at the beginning with Hsu-Tsai linear time algorithm and then recomputed with the JumpToRoot operation. With JumpToRoot$(v)$ we can compute the distance $sv$ from $v$ to the root. Then Jump$(root, sc - sv)$ should end up in $v$, if $v$ is on the cycle.

4. There is a dynamic data structure that maintains the arcs on a circle and allows to find the value of *next* for each arc in time $O(\log n)$.

This structure, called *Pos*, is similar to the interval tree [dBvKOS97] for a set of segments and arcs can be deleted and inserted in *Pos* in $O(\log n)$ time. *Pos* will be used to find changes in the *next* function when two endpoints of arcs swap positions as a result of the shooter motion.

5. The value of the $MCC$ changes by at most 1 between the cells in $\mathcal{A}(\mathcal{S})$. Indeed, let $mcc(c)$ denote the size of this cover.

LEMMA 3.2. *If* $c_1$, $c_2$ *are the adjacent cells of* $\mathcal{A}(\mathcal{S})$ *that do not share the boundary that is one of the input segments then* $|mcc(c_1) - mcc(c_2)| \leq 1$.

PROOF: Moving from $c_1$ to $c_2$ corresponds to swapping

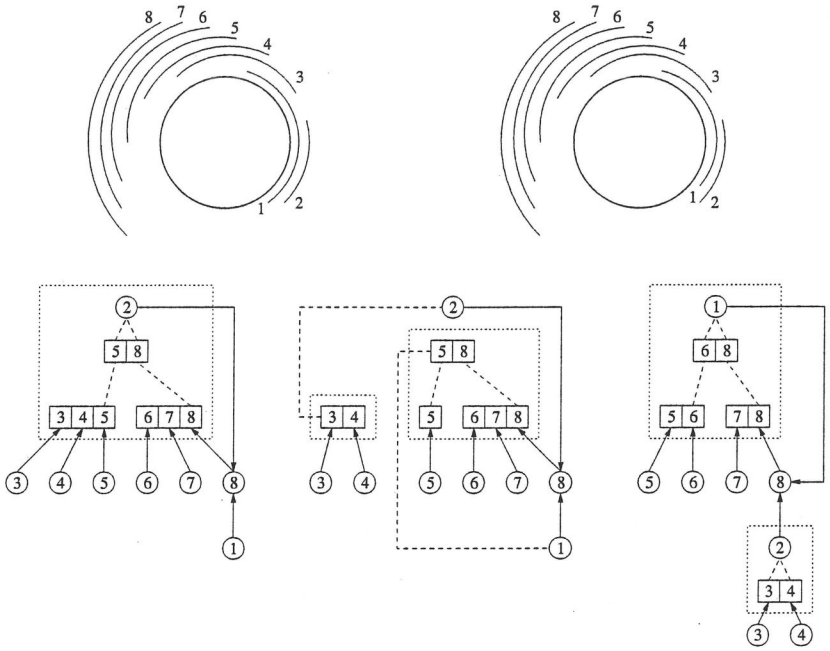


**Figure 3: One of the cases - arcs 1 and 2 swap positions**

two neighbor endpoints of some two arcs in $\widetilde{Arcs}(c_1)$. In terms of the circular-arc graph for $Arcs(c_1)$ it corresponds to adding or removing one edge. This may change the number of cliques in the current minimal cover; after removing an edge one of the cliques may break into two, or after adding an edge two cliques may become a larger clique in $Arcs(c_2)$. If the number of cliques increases, we have a cover that shows that $mcc(c_1) \leq mcc(c_2) \leq mcc(c_1) + 1$. If the number of cliques decreases then removing the newly added edge reduces the problem to the previous case and we have $mcc(c_2) \leq mcc(c_1) \leq mcc(c_2) + 1$, which ends the proof. □

This lemma is actually more important than it appears at first glance. As we remember, the size of the $MCC$ is the longest independent set in a cycle in the $c$-tree. Thus the lemma allows to avoid traversing the entire cycle to find where the sequence of *next* starts wrapping around the circle. Instead, it can be accomplished by jumping from arc $i$ to the node $j$ in the cycle that is at the distance of the old value of *mcc* minus 1 and then checking the overlap between arc $i$ and the *next* of just three arcs, $j$, $next(j)$ and $next(next(j))$. The gain is in reducing the cost to $O(\log^2 n)$.

To solve the generalized shooter problem we start with computing the arrangement $\mathcal{A}(S)$ and then for an arbitrarily selected cell we initialize the *ptb*-tree and compute the value of the $MCC$ using the linear time algorithm of Hsu and Tsai. Then, using a schedule (such as a spanning tree of the cells) we visit the cells in the arrangement. When the shooter crosses the boundary between two cells then two endpoints of some two arcs in the circular arc-graph change their position. Using *Pos* structure, in $O(\log n)$ time we find the new values of *next* for the affected arcs. The changes are propagated in $O(\log^2 n)$ time to the *ptb*-tree and the new value of the *mcc* is recorded. We can keep all the values in a priority queue represented with a binary heap.

After visiting all the cells we can read off the solution to the shooter problem from the heap.

The correctness of the algorithm follows from the following lemma.

LEMMA 3.3. *The algorithm correctly updates ptb-trees representing the c-trees after changes to the next values.*

PROOF: The *ptb* is initialized correctly to represent the initial $c$-tree. By case analysis of possible changes in positions between a pair of arcs when the boundary between cells is crossed, and from the correctness of operations on *ptb*-tree we demonstrate the claim. □

It leads to the following

THEOREM 3.1. *Algorithm* Shooting *is correct, i.e., it finds the minimal number of halflines necessary to shoot the $n$ given segments. The cost of the algorithm is $O(n^4 \log^2 n)$.*

PROOF: From Lemma 3.3 we know that the *ptb*-tree correctly represents the *next* relation for each cell in the arrangement as we move from one cell to another and that the returned value is equal to the size of the $MCC$ for the current circular arc-graph. This implies that the algorithm finds the solution to the shooting Problem. The shooting number is found by analyzing the neighborhood of the previous shooting number in the cycle. By Lemma 3.2 only such a neighborhood needs to be analyzed.

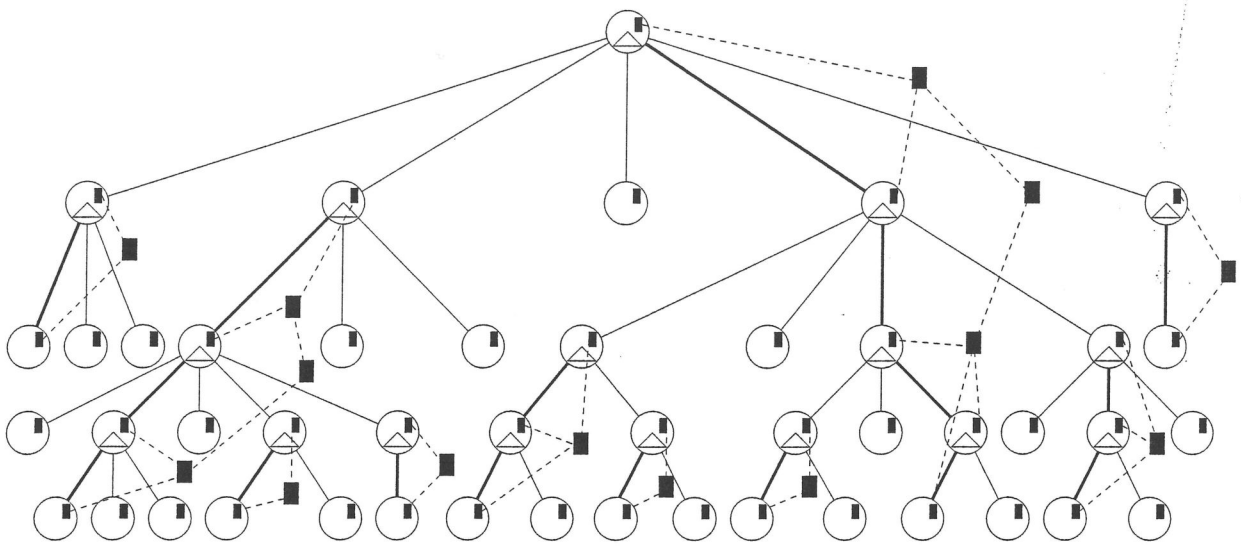


**Figure 4:** *A schematic view of a ptb-tree. B-trees of the children of nodes are represented with small triangles. The bold lines illustrate B-trees for the path decomposition.*

The initialization of the arrangement, the initialization of the data structures and $O(n^4)$ updates to the *ptb*-tree can be accomplished in $O(n^4 \log n)$ time. Moreover, after all the data structures are initialized, the cost per event is $O(\log^2 n)$. □

## 4. CONCLUSIONS

We have focused in this paper on a solution to the generalized shooter problem. By designing an efficient data representation *ptb*-tree for *c*-trees, we have been able to maintain the Minimum Clique Cover for the circular arc-graph corresponding to the segments viewed from the perspective of a moving shooter. It results in an algorithm with the best deterministic performance up-to-date. The algorithm is close to the optimal for segment arrangements with a reach set of optimal positions for the shooter.

Although the presentation assumed that the segments in $S$ are pairwise disjoint, this can be relaxed, and in particular, we can allow the shooter to cross the input segments while moving between cells. Note that when a segment is crossed the corresponding arc in the circular arc-graph jumps to the other half-circle, which makes the problem sightly more difficult.

The same approach can be used for other graph-theoretical structures such as the Maximum Dominating Sets and the Minimum Dominating Sets for circular arcs-graphs.

Other computational applications of the our approach include the $MCC$ problem for a dynamic set of union length arcs and the shooting problem for segments that are semi-transparent; selected segments in the set are invisible from one side and visible when viewed from the other.

Additionally, the *ptb*-tree tree data structure can be applied whenever there is a need to reattach groups of subtrees between nodes and to quickly jump from a node to its predecessor that is a given number of generations above in the tree. The structure borrows from B-trees and skip lists, as well as uses a technique that we call a path decomposition.

Possible furher applications may involve piercing sets and transversals for more complicated objects; polygons are good candidates here.

## 5. REFERENCES


[BMcC72] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:178–189, 1972.

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

[HT91] W.-L. Hsu and K.-H. Tsai. Linear time algorithms on circular-arc graphs. *Information Processing Letters*, 40:123–129, 1991.

[KNS00] M.J. Katz F. Nielsen and M. Segal. Shooter location through piercing sets. *Proceedings of the 16th EWCG*, 55–58, 2000.

[LSW90] D.T. Lee M. Sarrafzadeh and V.F. Wu. Minimum cuts for circular-arc graphs. *SIAM J. on Computing*, 19:1041–1050, 1990.

[NMB96] S.C Nandy K. Mukhopadhyaya and B.B. Bhattacharya. Shooter location problem. *Proceedings of the 8th Canad. Conf. Comput. Geom.*, 93–98, 1996.

[Pu90] W. Pugh. A probabilistic alternative to balanced trees. *Information Processing Letters*, 40:123–129, 1991.

[WZ97] C.A. Wang and B. Zhu. Shooter location problem revisited. *Proceedings of the 9th Canad. Conf. Comput. Geom.*, 223–228, 1997.

# Balanced Partition of Minimum Spanning Trees*

[Extended Abstract]

Mattias Andersson and Christos Levcopoulos
Department of Computer Science, Lund University
Box 118, 221 00 Lund
Sweden
christos@cs.lth.se

Joachim Gudmundsson†
Department of Computer Science, Utrecht University
PO Box 80.089, 3508 TB
Utrecht, the Netherlands
joachim@cs.uu.nl

Giri Narasimhan
School of Computer Science
Florida International University
Miami, FL 33199, USA
giri@fiu.edu

## ABSTRACT

To better handle situations where additional resources are available to carry out a task, many problems from the manufacturing industry involve "optimally" dividing a task into a constant $k$ number of smaller tasks. We consider the problem of partitioning a given set $\mathcal{S}$ of $n$ points in the plane into $k$ subsets, $\mathcal{S}_1, \ldots, \mathcal{S}_k$, such that $\max_{1 \leq i \leq k} |MST(\mathcal{S}_i)|$ is minimized. A variant of this problem arises in the shipbuilding industry[10].

Initially we show that this problem, the *k-BPMST* problem, is NP-hard, and we then continue by presenting two approximation algorithms. The first one, a straight-forward greedy algorithm, is a $k$-approximation algorithm that runs in $O(n \log n)$ time. The second algorithm, which runs in time $O(n \log n)$, is a $(4/3 + \epsilon)$-approximation algorithm for the case $k = 2$ and a $(2 + \epsilon)$-approximation algorithm for the case $k \geq 3$.

## Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*geometrical problems and computations*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

## General Terms

Algorithms, Theory

## Keywords

Minimum Spanning Tree, Balanced Partition, NP-hard, Approximation, Approximation Scheme, Computational Geometry, Travelling Salesman Problem

*To appear in the International Conference on Computational Science, April 2002, Amsterdam, The Netherlands.

†Supported by the Swedish Foundation for International Cooperation in Research and Higher Education

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

## 1. INTRODUCTION

In one interesting application from the shipbuilding industry, the task is to use a robot to cut out a set of prespecified regions from a sheet of metal while minimizing the completion time. In another application, a salesperson needs to meet some potential buyers. Each buyer specifies a region (i.e., a *neighborhood*) within which the meeting needs to be held. A natural optimization problem is to find a salesperson tour of shortest length that visits all of the buyers' neighborhoods and finally returns to his initial departure point. Both these problems are related to the problem known in the literature as the *Traveling Salesperson problem with Neighborhoods* (TSPN) and which has been extensively studied [2, 4, 6, 7, 8, 9]. The problem (TSPN) asks for the shortest tour that visits each of the neighborhoods. The problem was recently shown to be APX-hard[7].

Interesting generalizations of the TSPN problem arise when additional resources ($k > 1$ robots in the sheet cutting problem, or $k > 1$ salespersons in the second application above) are available. The $k$-TSPN problem is a generalization of the problem where we are given $k$ salespersons and the aim is to minimize the completion time, i.e., minimize the distance traveled by the salespersons making the longest journey.

The need for partitioning the input set such that the optimal substructures are balanced gives rise to many interesting theoretical problems. If we restrict our inputs to sets of points instead of regions an interesting problem is the one called the $k$-TSP problem, which is equivalent to the $k$-TSPN problem but with the regions replaced by points. In this paper, however, we consider the problem of partitioning the input so that the sizes of the minimum spanning trees of the subsets are balanced. More formally, the *Balanced Partition Minimum Spanning Tree problem* ($k$-BPMST) is stated as follows:

PROBLEM 1. *Given a set of $n$ points $\mathcal{S}$ in the plane, partition $\mathcal{S}$ into $k$ sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the*

*largest minimum spanning tree,*

$$W = \max_{1 \le i \le k} (|M(\mathcal{S}_i)|)$$

*is minimized. Here $M(\mathcal{S}_i)$ is the minimum spanning tree of the subset $\mathcal{S}_i$ and $|M(\mathcal{S}_i)|$ is the weight of the minimum spanning tree of $\mathcal{S}_i$.*

Note that a $c$-approximation for the $k$-BPMST problem immediately yields a $2c$-approximation for the $k$-TSP problem, by traversing the produced MST's.

The paper is organized as follows. We first show that the problem is NP-hard. We then present an approximation algorithm with approximation factor $4/3 + \epsilon$ for the case $k = 2$, and with an approximation factor $2 + \epsilon$ for the case $k \geq 3$. The algorithm runs in time $O(n \log n)$.

## 2. NP HARDNESS

In this section we show that the $k$-BPMST problem is NP-hard. In order to do this we need to state the recognition version of the $k$-BPMST problem:

PROBLEM 2. *Given a set of $n$ points $\mathcal{S}$ in the plane, and a real number $\mathcal{L}$, does there exist a partition of $\mathcal{S}$ into $k$ sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest minimum spanning tree,*

$$W = \max_{1 \le i \le k} (|M(\mathcal{S}_i)|) \leq \mathcal{L}?$$

In a computational model in which we can handle square roots in polynomial time, such as the real-RAM model (which will be used for simplicity), this formulation of the problem is sufficient in order to show that the $k$-BPMST problem is NP-hard. Note, however, that it may be inadequate in more realistic models, such as the Turing model, where efficient handling of square roots may not be possible. The computation of roots is necessary to determine the length of edges between points, which, in turn, is needed in order to calculate the weight of a minimum spanning tree. So in a realistic computational model the hardest part may not be to partition the points optimally, but instead to calculate precisely the length of the MST's. Thus, in these more realistic computational models we would like to restrict the problem to instances where the lengths of MST's are easy to compute. For example, this can be done by modifying the instances created in the reduction below, by adding some points so that the MST's considered only contain vertical and horizontal edges.

The proof is done (considering the real-RAM model) by a polynomial reduction from the following recognition version of PARTITION.

PROBLEM 3. *Given integers $a = \{a_1 \leq \ldots \leq a_n\}$, the recognition version of the partition problem is: Does there exist a subset $P \subseteq I = \{1, 2, \ldots, n\}$ such that*

$$\#P = \#I/P \quad \text{and} \sum_{j \in P} a_j = \sum_{j \in I/P} a_j$$

We will denote $\#P$ by $h$, $h = n/2$. This version of PARTITION is NP-hard [5].

LEMMA 1. *The $k$-BPMST problem is NP-hard.*

PROOF. The reduction is done as follows. Given a PARTITION instance we create a 2-BPMST instance, in polynomial time, such that it is a yes-instance if, and only if, the PARTITION-instance is a yes-instance. Obviously PARTITION then polynomially reduces to 2-BPMST. Given that the PARTITION-instance contains $n$ integers $a_1, \ldots, a_n$, we create the following 2-BPMST instance. A set of points $\mathcal{S}$, as shown in Figure 1 (a) is created, with inter point distances as shown in Figure 1 (b). A closer description of these points and some additional definitions is given below:

- $a' = \{a'_1, \ldots, a'_n\}$, where $a'_i = (0, i\lambda)$,
- $l = \{l_1, \ldots, l_n\}$, where $l_i = (-\delta - a_i, i\lambda)$,
- $r = \{r_1, \ldots, r_n\}$, where $r_i = (\delta + a_i, i\lambda)$,
- $l' = \{l'_1, \ldots, l'_{n-1}\}$, where $l'_i$ is the midpoint on the line between $l_i$ and $l_{i+1}$, and
- $r' = \{r'_1, \ldots, r'_{n-1}\}$, where $r'_i$ is the midpoint on the line between $r_i$ and $r_{i+1}$

For any given partition $P = \{P[1], P[2], \ldots, P[h]\} \subseteq \{1, 2, \ldots, n\}$, we define $a^* = \{a_{P[1]}, \ldots, a_{P[n]}\}$. Further, let $\lambda = 11n(a_n + n)$ and let $\delta = 7n(a_n + n)$. Note that $\lambda_i^2 \leq \lambda^2 + a_n^2$ which implies that $\lambda_i \leq 12n(a_n + n)$, which means that $\gamma_i = \lambda_i/2 \leq 6n(a_n + n)$. Finally let (see definition 2)

$$\mathcal{L} = (\sum_{i \in I} a_i)/2 + n/2 \cdot \delta + \sum_{i=1}^{n-1} \lambda_i$$

Since the number of points in $\mathcal{S}$ is polynomial it is clear that this instance can be created in polynomial time. Next we consider the "if", and the "only if" parts separately.

*If* If P exists and we have a yes PARTITION-instance it is clear that the corresponding 2-BPMST instance is also a yes-instance. This follows when the partition $\mathcal{S}'_1 = a^* + l + l'$, $\mathcal{S}'_2 = \mathcal{S} - \mathcal{S}_1$ (a class 1 partition, as defined below) is considered. The general appearance of $M(\mathcal{S}'_1)$ and $M(\mathcal{S}'_2)$ (see Figure 1 (c)) is determined as follows. The points $l + l'$ and the points $r + r'$ will be connected as illustrated in Figure 1 (c), which follows from the fact that $\gamma_i < \delta < \delta + a_1$. Next consider the remaining points $a'$. Any point $a'_i$ will be connected to either $l_i$ (in $M(\mathcal{S}'_1)$) or $r_i$ (in $M(\mathcal{S}'_2)$), since $r_i$ and $l_i$ are the points located closest to $a'_i$ (follows since $\lambda > \delta + a_n$). Thus,

$$|M(\mathcal{S}'_1)| = |M(\mathcal{S}'_2)| = (\sum_{i \in I} a_i)/2 + n/2 \cdot \delta + \sum_{i=1}^{n-1} \lambda_i$$

and we have that the created instance is a yes-instance.

*Only if* We have that $P$ does not exist and we therefore want to show that the created 2-BPMST is a no-instance. For this two classes of partitions will be examined:

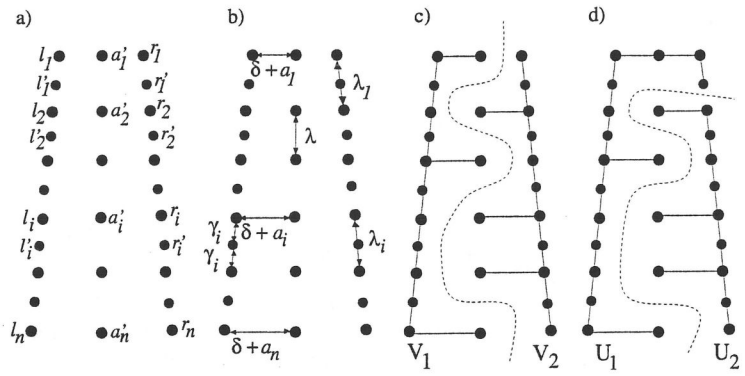



**Figure 1: The set of points $\mathcal{S}$ created for the reduction. In Figure (a) all notations for the points are given. Similarly, in Figure (b) the notations for the distances between points are given. Figure (c) illustrates a class 1 partition, and (d) illustrates a class 2 partition.**

- All partitions $\mathcal{V}_1, \mathcal{V}_2$ such that $l + l' \subseteq \mathcal{V}_1$ and $r + r' \subseteq \mathcal{V}_2$
- All other partitions $\mathcal{U}_1, \mathcal{U}_2$ not belonging to class 1.

We start by examining the first class (illustrated by Figure 1c). Note that an optimal MST will contain the edges in $M(\mathcal{V}_1)$ and $M(\mathcal{V}_2)$ plus the edge between $a'_1$ and $l_1$ or $r_1$, hence $|M(\mathcal{S})| = |M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| + \delta + a_1$. Note also that $|M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| = 2 \cdot \mathcal{L}$. For all partitions $\mathcal{V}'_1 \subseteq \mathcal{V}_1, \mathcal{V}'_2 \subseteq \mathcal{V}_2$ such that each subset $\mathcal{V}'_1, \mathcal{V}'_2$ contains exactly $|a'|/2$ points from the set $a'$ it is clear, since $P$ does not exist, that $max\{|M(\mathcal{V}'_1)|, |M(\mathcal{V}'_2)|\} > \mathcal{L}$. This is true also for the partitions $\mathcal{V}^*_1 \subseteq \mathcal{V}_1, \mathcal{V}^*_2 \subseteq \mathcal{V}_2$ such that each subset does *not* contain exactly $|a'|/2$ points from the set $a'$. To see this consider any such partition and the corresponding subset $\mathcal{V}^*_i$ such that $|\mathcal{V}^*_i| = \max\{|\mathcal{V}^*_1|, |\mathcal{V}^*_2|\}$. We have that

$$|M(\mathcal{V}^*_i)| \geq \delta + n/2 \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > (\sum_{i \in I} a_i) + n/2 \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > \mathcal{L}$$

This implies that $\max\{|M(\mathcal{V}^*_1)|, |M(\mathcal{V}^*_2)|\} > \mathcal{L}$.

Next consider the class 2 partitions (illustrated by Figure 1d). There is always an edge of weight $\gamma_i$ $(1 \leq i \leq n)$ connecting the two point sets of any such partition. This means that there can not exist a class 2 partition $\mathcal{U}_1, \mathcal{U}_2$ such that $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} \leq \mathcal{L}$, because we could then build a tree with weight at most $2 \cdot \mathcal{L} + \gamma_i < |M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| + \delta + a_1 = |M(\mathcal{S})|$, which is a contradiction. Thus, $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} > \mathcal{L}$, which concludes this lemma. □

# 3. A $(2+\epsilon)$ APPROXIMATION

In this section a $(2+\epsilon)$-approximation algorithm is presented. Note also that a straight-forward greedy algorithm, that partitions $M(\mathcal{S})$ into $k$ sets by removing the $k-1$ longest edges gives an approximation of $k$. The main idea of the $(2+\epsilon)$-approximation algorithm is to partition $\mathcal{S}$ into a constant number of small components, test all valid combinations of these components and give the best combination as output. In order to do this efficiently, as will be seen later, one will need an efficient partitioning algorithm. Tree partitioning algorithms have been previously studied [3]. Thus, we can (by studying $M(\mathcal{S})$ and using similar principles) easily construct a partitioning algorithm, denoted VALIDPARTITION or VP for short, such that Lemma 2 can be shown. A partition of a point set $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is said to be valid if $\max(|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|) \leq 2/3 \cdot |M(\mathcal{S})|$.

LEMMA 2. *[1] Given a set of points $\mathcal{S}$,* VP *divides $\mathcal{S}$ into two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that (i)* $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq \frac{2}{3}M(\mathcal{S})$, *and (ii)* $|M(\mathcal{S}_1)| + |M(\mathcal{S}_2)| \leq |M(\mathcal{S})|$. *If* VP *is given a MST of $\mathcal{S}$ as input then it holds that the time needed for* VP *to compute a valid partition is $O(n)$.*

## 3.1 Repeated ValidPartition

An algorithm denoted REPEATEDVALIDPARTITION, or RVP for short, will use VP repeatedly in order to create the many small components mentioned in the introduction of this section. RVP, given $M(\mathcal{S})$ and an integer $m$, first divides $M(\mathcal{S})$ into two components using VP. Then RVP repeatedly partitions the largest component created thus far (again using VP) until $m$ components have been created. The following lemma expresses an important characteristic of RVP.

LEMMA 3. *[1] Given a minimum spanning tree of a set of points $\mathcal{S}$ and an integer $m$,* RVP *will partition $\mathcal{S}$ into $m$ components $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that*

$$\max(|M(\mathcal{S}_1)|, \ldots, |M(\mathcal{S}_m)|) \leq \tfrac{2}{m}|M(\mathcal{S})|.$$

## 3.2 The approximation algorithm

Now we are ready to state the algorithm CA. As input we are given a set $\mathcal{S}$ of $n$ points, an integer $k$ and a positive real constant $\epsilon$. The algorithm differs in two separate cases, $k = 2$ and $k \geq 3$. First $k = 2$ is examined, in which the following steps are performed:

**step 1:** Divide $M(\mathcal{S})$ into $\frac{4}{\epsilon'}$ components, using RVP, where $\epsilon' = \frac{\epsilon}{4/3+\epsilon}$. The reason for the value of $\epsilon'$ will become clear below. Let $W$ denote the heaviest component created and let $w$ denote its weight.

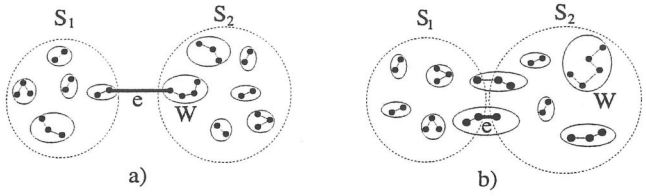



**Figure 2: The two cases for CA, $k = 2$. The edge $e$ (marked) is the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$.**

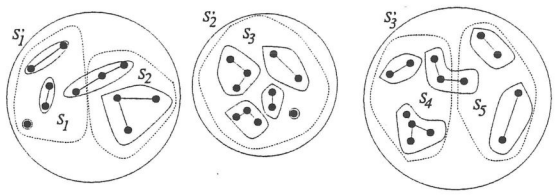



**Figure 3: $\mathcal{S}_1, \ldots, \mathcal{S}_k$ is an optimal partition of $\mathcal{S}$. All subsets that can be connected by edges of length at most $w$ are merged, thus creating the new set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$.**

**step 2:** Combine all components created in step 1, in all possible ways, into two groups.

**step 3:** For each combination tested in step 2, compute the MST for each of its two created groups.

**step 4:** Output the best tested combination

THEOREM 1. *For $k = 2$ the approximation algorithm* CA *produces a partition which is within a factor $\frac{4}{3} + \epsilon$ of the optimal in time $O(n \log n)$.*

PROOF. Let $V_1$ and $V_2$ be the partition obtained from CA. Assume that $\mathcal{S}_1$ and $\mathcal{S}_2$ is the optimal partition, and let $e$ be the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$. According to Lemma 3 it follows that $w \leq 2/(4/\epsilon')|M(\mathcal{S})| = \frac{\epsilon'}{2}|M(\mathcal{S})|$. We will have two cases, $|e| > w$, and $|e| \leq w$, which are illustrated in Figure 2 (a) and Figure 2 (b), respectively. In the first case every component is a subset of either $\mathcal{S}_1$ or $\mathcal{S}_2$. This follows since a component consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$ must include an edge with weight greater than $w$. Thus, no such component can exist among the components created in Step 1. Further, this means that the partition $\mathcal{S}_1$ and $\mathcal{S}_2$ must have been tested in Step 2 of CA and, hence, the optimal solution must have been found.

In the second case, $|e| \leq w$, there may exist components consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$, see Fig. 2. To determine an upper bound of the approximation factor we start by examining an upper bound of CA. The dividing process in Step 1 of CA starts with $M(\mathcal{S})$ being divided into 2 components $M(\mathcal{S}'_1)$ and $M(\mathcal{S}'_2)$, such that $\max(|M(\mathcal{S}'_1)|, |M(\mathcal{S}'_2)|) \leq \frac{2}{3}|M(\mathcal{S})|$. These two components are then divided into several smaller components. This immediately reveals an upper bound of $|CA| \leq \frac{2}{3}|M(\mathcal{S})|$. Next the lower bound is examined. We have:

$$|opt| \geq \frac{|M(\mathcal{S})| - |e|}{2} \geq \frac{|M(\mathcal{S})|}{2} - \frac{\epsilon' \cdot M(\mathcal{S})}{2} \geq (1-\epsilon')\frac{M(\mathcal{S})}{2}.$$

Then, if the upper and lower bound are combined we get:

$$|CA|/|opt| \leq \frac{\frac{2}{3}|M(\mathcal{S})|}{(1-\epsilon')\frac{M(\mathcal{S})}{2}} \leq \frac{4/3}{1-\epsilon'} \leq 4/3 + \epsilon.$$

In the third inequality we used the fact that $\epsilon' \leq \frac{\epsilon}{4/3+\epsilon}$.

Next consider the complexity of CA. In Step 1 $M(\mathcal{S})$ is divided into a constant number of components using VP. This takes $O(n)$ time. Then, in Step 2, these components are combined in all possible ways. This takes $O(1)$ time since there are a constant number of components. For each tested combination there is a constant number of MST's to be computed in Step 3. Further, since there are a constant number of combinations and $M(\mathcal{S})$ takes $O(n \log n)$ to compute, Step 3 takes $O(n \log n)$ time. □

Next consider $k \geq 3$. In this case the following steps are performed:

**step 1:** Compute $M(\mathcal{S})$ and remove the $k-1$ heaviest edges $e_1, \ldots, e_{k-1}$ of $M(\mathcal{S})$, thus resulting in $k$ separate trees $M(U'_1), \ldots, M(U'_k)$.

**step 2:** Divide each of the trees $M(U'_1), \ldots M(U'_k)$ into $\frac{k \cdot C}{\epsilon'}$ components, using RVP. $C$ is a positive constant and $\epsilon' = \frac{\epsilon}{2+\epsilon}$. The reason for the value of $\epsilon'$ will become clear below. Denote the resulting components $M(U_1), \ldots, M(U_r)$, where $r = \frac{k \cdot C}{\epsilon'} \cdot k$. Further set $w = \max\{|M(U_1)|, \ldots, |M(U_r)|\}$.

**step 3:** Combine $U_1, \ldots, U_r$ in all possible ways into $1, \ldots, k$ groups.

**step 4:** For each such combination do:

- Compute the MST for each of its corresponding groups.
- Divide each such MST in all possible ways, using RVP. That is, each MST is divided into $1, \ldots, i (i \leq k)$ components, such that the total number of components resulting from all the divided MST's equals $k$. Each such division defines a partition of $\mathcal{S}$ into $k$ subsets.

**step 5:** Of all the tested partitions in step 4, output the best.

THEOREM 2. *For $k \geq 3$ the approximation algorithm CA produces a partition which is within a factor of $2 + \epsilon$ of the optimal in time $O(n \log n)$*

PROOF. A constant number of components are created which means that the time complexity is the same as for the case $k = 2$, $O(n \log n)$. To prove the approximation factor we first give an upper bound on the weight of the solution produced by CA and then we provide a lower bound for an optimal solution. Combining the two results will conclude the theorem.

Consider an optimal partition of $\mathcal{S}$ into $k$ subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Merge all subsets that can be connected by edges of length at most $w$. From this we obtain the sets $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$, where $k' \leq k$ (see Figure 3). Let $m'_i$ denote the number of elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$ included in $\mathcal{S}'_i$. The purpose of studying these new sets is that every component created in Step 2 of CA belongs to exactly one element in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. A direct consequence of this is that a combination into $k'$ groups equal to $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$ must have been tested in Step 3.

Step 4 guarantees that $M(\mathcal{S}'_1), \ldots, M(\mathcal{S}'_{k'})$ will be calculated, and that these MST's will be divided in all possible ways. Thus, a partition will be made such that each $M(\mathcal{S}'_i)$ will be divided into exactly $m'_i$ components. This partitions $\mathcal{S}$ into $k$ subsets $\mathcal{V}_1, \ldots, \mathcal{V}_k$. Let $\mathcal{V}$ be a set in $\mathcal{V}_1, \ldots, \mathcal{V}_k$ such that $|M(\mathcal{V})| = \max_{1 \leq i \leq k}(|M(\mathcal{V}_i)|)$. We wish to restrict our attention to exactly one element of the set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. Thus, we note that $\mathcal{V}$ is a subset of exactly one element $\mathcal{S}'$ in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. Assume that $M(\mathcal{V})$ was created in Step 4 when $M(\mathcal{S}')$ was divided into $m'$ components using RVP. Thus, $M(\mathcal{V}) \leq \frac{2}{m'}|M(\mathcal{S}')|$, according to Lemma 3. Since the partition $\mathcal{V}_1, \ldots, \mathcal{V}_k$ will always be tested we have that $|CA| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$.

Next a lower bound of an optimal solution is examined. Let $|opt'|$ be the value of an optimal solution for $\mathcal{S}'$ partitioned into $m'$ subsets. Note that $\mathcal{S}'$ consists of $m'$ elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Assume w.l.o.g that $\mathcal{S}' = \mathcal{S}_1 + \ldots + \mathcal{S}_{m'}$. This means that $\mathcal{S}_1, \ldots, \mathcal{S}_{m'}$ is a possible partition of $\mathcal{S}'$ into $m'$ subsets. Thus, $|opt| \geq \max_{1 \leq i \leq m'}(|M(S_i)|) = |opt'|$. Assume w.l.o.g. that $e'_1, \ldots, e'_{m'-1}$ are the edges in $M(\mathcal{S})$ connecting the components in $\mathcal{S}'$. We have:

$$|opt| \geq |opt'| \geq \frac{1}{m'}(|M(S')| - \sum_{i=1}^{m'-1} |e'_i|) \geq$$
$$\geq \frac{1}{m'}(|M(S')| - (m'-1)w) \qquad (1)$$

To obtain a useful bound we need an upper bound on $w$. Consider the situation after Step 1 has been performed. We have

$\max_{1 \leq i \leq k}(|M(U'_i)|) \leq |M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|$. Since each $U'_i$ is divided into $\frac{k \cdot C}{\epsilon'}$ components we have that the resulting components, and therefore also $w$, have weight at most $2/(\frac{k \cdot C}{\epsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)$, according to Lemma 3. Using the above bound gives us:

$$\frac{w}{|opt|} \leq \frac{2/(\frac{k \cdot C}{\epsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)}{\frac{1}{k}(|M(\mathcal{S})| - \sum_{i=1}^{k-1} |e_i|)} \leq$$
$$\leq \frac{2 \cdot \epsilon'}{C} \Rightarrow w \leq \frac{2 \cdot \epsilon'}{C}|opt| \qquad (2)$$

Note that $|opt| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$. Further, setting $C \geq 4$ and combining 1 and 2 gives us:

$$|opt| \geq \frac{1}{m'}\left(|M(S')| - (m'-1)\frac{2 \cdot \epsilon'}{C}|opt|\right) \geq (1-\epsilon')\frac{|M(\mathcal{S}')|}{m'}.$$

Combining the two bounds together with the fact that $\epsilon' \leq \epsilon/(2+\epsilon)$ concludes the theorem.

$$|CA|/|opt| \leq \frac{\frac{2}{m'}|M(\mathcal{S}')|}{(1-\epsilon')\frac{|M(\mathcal{S}')|}{m'}} \leq \frac{2}{1-\epsilon'} \leq 2+\epsilon.$$

□

The results in this paper are shown in a geometric setting but can, under some minor restrictions, be shown in a metric setting also.

# 4. CONCLUSION

In this paper it was first shown that the $k$-BPMST problem is NP-hard. After this was established, the next step was to design approximation algorithms for the problem. The algorithm is based on partitioning the point set into a constant number of smaller components and then trying all possible combinations of these small components. This approach revealed a $(4/3+\epsilon)$-approximation in the case $k = 2$, and a $(2+\epsilon)$-approximation in the case $k \geq 3$. The time complexity of the algorithm is $O(n \log n)$.

# 5. REFERENCES


[1] M. Andersson. *Balanced Partition of Minimum Spanning Trees.* LUNDFD6/NFCS-5215/1–30/2001, Master thesis, Department of Computer Science, Lund University, Sweden, 2001.

[2] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1994.

[3] R. I. Becker and S. R. Schach. A shifting algorithm for min-max tree partitioning. *Journal of the Association for Computing Machinery*, 29(1):58–67, January 1982.

[4] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[6] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods. *Nordic Journal of Computing*, 6:469–488, 1999.

[7] J. Gudmundsson and C. Levcopoulos. *Hardness Result for TSP with Neighborhoods.* Technical report, LU-CS-TR:2000-216, Department of Computer Science, Lund University, Sweden, 2000.

[8] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1995.

[9] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[10] B. Shaleooi. *Algoritmer för plåtskärning* (Eng. transl. *Algorithms for cutting sheets of metal).* LUNDFD6/NFCS-5189/1–44/2001, Master thesis, Department of Computer Science, Lund University, Sweden, 2001.

# Approximating the Geometric Minimum-Diameter Spanning Tree

Joachim Gudmundsson*
Dept. of Comp. Science
Utrecht University
The Netherlands
joachim@cs.uu.nl

Herman Haverkort†
Dept. of Comp. Science
Utrecht University
The Netherlands
herman@cs.uu.nl

Sang-Min Park
Dept. of Comp. Science
KAIST
Korea
smpark@jupiter.kaist.ac.kr

Chan-Su Shin
School of Electr. and Inform. Engineering
Hankuk University of Foreign Studies
Korea
cssin@hufs.ac.kr

Alexander Wolff
Institut für Mathematik und Informatik
Universität Greifswald
Germany
awolff@uni-greifswald.de

## ABSTRACT

Let $P$ be a set of $n$ points in the plane. The geometric minimum-diameter spanning tree (MDST) of $P$ is a tree that spans $P$ and minimizes the Euclidian length of the longest path. It is known that there is always a mono- or a dipolar MDST, i.e. a MDST whose longest path consists of two or three edges, respectively. The more difficult dipolar case can so far only be solved in $O(n^3)$ time. In this paper we give an $O(n \log n)$-time approximation scheme for the MDST.

## 1. INTRODUCTION

The geometric minimum-diameter spanning tree (MDST) can be seen as a network without cycles that minimizes the maximum travel time between any two sites connected by the network. This is of importance e.g. in communication systems where the maximum delay in delivering a message is to be minimized. Ho et al. showed that there always is a mono- or a dipolar MDST [4]. For a different proof, see [3]. Ho et al. gave an $O(n \log n)$-time algorithm for the monopolar and an $O(n^3)$-time algorithm for the dipolar case [4]. In the more difficult dipolar case the objective is to find a minimum-diameter dipolar spanning tree (MDdST), i.e. a tree with two roots $x, y \in P$ such that the function $r_x + |xy| + r_y$ is minimized, where $|xy|$ is the Euclidean distance of $x$ and $y$, and $r_x$ and $r_y$ are the radii of two disks centered at $x$ and $y$ that cover $P$.

We show that there is a fast approximation scheme for the MDdST. More precisely, given a set $P$ of $n$ points and some $\varepsilon > 0$ we show how to compute in $O(n \log n)$ time and space a dipolar tree whose diameter is at most $(1+\varepsilon)$ times as long as the diameter of a MDdST. Our approximation scheme is based on a well-separated pair decomposition [1] of $P$, a very powerful tool that we will briefly review. This decomposition makes it possible to consider only a linear number of pairs of points on the search for the two poles of an approximate MDdST. The main novelty of our scheme is a discretization of the orientations of line segments to a constant number of orientations. We only consider projections of $P$ onto those directions, which gives us fast access to farthest neighbors. For the full paper, see [2].

## 2. THE WELL-SEPARATED PAIR DECOMPOSITION

Our algorithm uses the well-separated pair decomposition (WSPD) of Callahan and Kosaraju [1] that we now review.

DEFINITION 1. *Let $\tau > 0$ be a real number, and let $A$ and $B$ be two finite sets of points in $\mathbb{R}^d$. We say that $A$ and $B$ are* well-separated w.r.t. $\tau$, *if there are two disjoint $d$-dimensional balls $C_A$ and $C_B$ both of radius $r$ such that $A \subset C_A$, $B \subset C_B$, and the distance between $C_A$ and $C_B$ is at least equal to $\tau r$.*

The parameter $\tau$ will be referred to as the *separation constant*. The following lemma follows easily from Definition 1.

LEMMA 1. *Let $A$ and $B$ be two finite sets of points that are well-separated w.r.t. $\tau$, let $x$ and $p$ be points of $A$, and let $y$ and $q$ be points of $B$. Then (i) $|xy| \leq (1 + 2/\tau) \cdot |xq|$, (ii) $|xy| \leq (1 + 4/\tau) \cdot |pq|$, (iii) $|px| \leq (2/\tau) \cdot |pq|$, and (iv) the angle between the line segments $pq$ and $py$ is at most* $\arcsin(2/\tau)$.

*Supported by the Swedish Foundation for International Cooperation in Research and Higher Education.

†Supported by the Netherlands' Organization for Scientific Research.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

DEFINITION 2. *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and $\tau > 0$ a real number. A* well-separated pair decomposition *for $P$ w.r.t. $\tau$ is a sequence $(A_1, B_1), (A_2, B_2), \ldots, (A_\ell, B_\ell)$ of pairs of non-empty subsets of $P$ such that each pair is well-separated w.r.t. $\tau$ and for any $p \neq q \in P$, there is exactly one pair $(A_i, B_i)$ with $p \in A_i$ and $q \in B_i$ or vice versa.*

Callahan and Kosaraju show that a WSPD with $\ell = O(\tau^2 n)$ can be computed using $O(n \log n + \tau^2 n)$ time and space.

# 3. A STRAIGHT-FORWARD APPROXIMATION SCHEME

The approximation algorithm consists of two subalgorithms: the first algorithm computes a minimum-diameter monopolar spanning tree and the second computes an approximation of the MDdST. When these two trees have been computed we output the one with smaller diameter. According to [4] there exists a MDST that is either a monopolar or a dipolar tree. The minimum-diameter monopolar tree can be computed in time $O(n \log n)$, hence we will focus on the problem of computing a MDdST. Let $d_{\min}$ be the diameter of a MDdST and let $\mathcal{T}_{pq}$ denote a spanning tree with dipole $\{p, q\}$ whose diameter is minimum among all such trees. For any dipolar spanning Tree $\mathcal{T}$ with dipole $\{u, v\}$ let $r_u(\mathcal{T})$ ($r_v(\mathcal{T})$) be the length of the longest edge of $\mathcal{T}$ incident to $u$ ($v$) without taking into account the edge $uv$. When it is clear which tree we refer to, we use $r_u$ and $r_v$.

OBSERVATION 1. *Let $(A_1, B_1), \ldots, (A_\ell, B_\ell)$ be a well-separated pair decomposition (WSPD) of $P$ with separation constant $\tau$, and let $p$ and $q$ be any two points in $P$. Then there is a pair $(A_i, B_i)$ such that for every point $u \in A_i$ and every point $v \in B_i$ the inequality* diam $\mathcal{T}_{uv} \leq (1 + 8/\tau) \cdot$ diam $\mathcal{T}_{pq}$ *holds.*

PROOF. According to Definition 2 there is a pair $(A_i, B_i)$ in the WSPD such that $p \in A_i$ and $q \in B_i$. If $u$ is any point in $A_i$ and $v$ is any point in $B_i$, then let $\mathcal{T}$ be the tree with poles $u$ and $v$ where $u$ is connected to $v$, $p$ and each neighbor of $p$ in $\mathcal{T}_{pq}$ except $q$ is connected to $u$, and $q$ and each neighbor of $q$ in $\mathcal{T}_{pq}$ except $p$ is connected to $v$. By Lemma 1(ii) $|uv| \leq (1 + 4/\tau)|pq|$ and by Lemma 1(iii) $r_u \leq |up| + r_p \leq 2|pq|/\tau + r_p$. Since diam $\mathcal{T} = r_u + |uv| + r_v$ we have

$$\text{diam } \mathcal{T} \leq \left(r_p + 2\frac{|pq|}{\tau}\right) + \left(|pq| + 4\frac{|pq|}{\tau}\right) + \left(r_q + 2\frac{|pq|}{\tau}\right) < \left(1 + \frac{8}{\tau}\right) \cdot \text{diam } \mathcal{T}_{pq}.$$

The lemma follows since $\mathcal{T}_{uv}$ has minimum diameter among all dipolar spanning trees with dipole $\{u, v\}$. □

A first algorithm is now obvious. For each pair $(A_i, B_i)$ in the WSPD of $P$ pick *any* vertex $p \in A_i$ and *any* vertex $q \in B_i$, and compute $\mathcal{T}_{pq}$. This is done by checking every possible radius of a disk centered at $p$ as in [4] in $O(n \log n)$ time. Setting $\tau$ to $8/\varepsilon$ yields:

LEMMA 2. *A dipolar tree $\mathcal{T}$ with* diam $\mathcal{T} \leq (1 + \varepsilon) \cdot d_{\min}$ *can be computed in $O(\frac{n}{\varepsilon^2} + n^2 \log n)$ time using $O(\frac{n}{\varepsilon^2} + n \log n)$ space.*

# 4. A FAST APPROXIMATION SCHEME

Now we describe a faster algorithm and show its time complexity; we will prove its correctness in Section 5.

THEOREM 1. *A dipolar tree $\mathcal{T}$ with* diam $\mathcal{T} \leq (1+\varepsilon) \cdot d_{\min}$ *can be computed in $O(\frac{n}{\varepsilon^3} + \frac{n}{\varepsilon} \log n)$ time using $O(\frac{n}{\varepsilon^2} + n \log n)$ space.*

The idea of the algorithm is again to check a linear number of pairs of points, using the WSPD, but to speed up the computation of the disks around the two poles. Note that we need to find a close approximation of the diameters of the disks to be able to guarantee a $(1 + \varepsilon)$-approximation of the MDdST. Obviously we cannot afford to try all possible disks for all possible pairs of poles. Instead of checking the disks we will show in the analysis that it suffices to check a constant number of partitions of the points among the poles. The partition of points is done by cuts that are orthogonal to the line through the poles. We cannot afford to do this for each possible pair. Instead we select a constant number of orientations and use a constant number of orthogonal cuts for each orientation. For each cut we calculate for each point in $P$ the approximate distance to the farthest point on each side of the cut. Below we give a more detailed description of the algorithm. For its pseudocode refer to Algorithm 1.

**Phase 1: Initializing.** Choose an auxiliary positive constant $\kappa < \min\{0.9\epsilon, 1/2\}$. As will be clear later, this parameter can be used to fine-tune which part of the algorithm contributes how much to the uncertainty and to the running time. In phase 3 the choice of the separation constant $\tau$ will depend on the value of $\kappa$ and $\varepsilon$.

DEFINITION 3. *A set of points $P$ is said to be $l$-ordered if the points are ordered with respect to their orthogonal projection onto the line $l$.*

Let $l_i$ be the line with angle $\frac{i\pi}{\gamma}$ to the horizontal line, where $\gamma = \lceil 4/\kappa \rceil$. This implies that for an arbitrary line $l$ there exists a line $l_i$ such that $\angle l_i l \leq \frac{\pi}{2\gamma}$. For each $i$, $1 \leq i \leq \gamma$, sort the input points with respect to the $l_i$-ordering. We obtain $\gamma$ sorted lists $F = \{F_1, \ldots, F_\gamma\}$. Each point $p$ in $F_i$ has a pointer to itself in $F_{(i \bmod \gamma)+1}$. The time to construct these lists is $O(\gamma n \log n)$.

For each $l_i$, rotate $P$ and $l_i$ such that $l_i$ is horizontal and consider the orthogonal projection of the points in $P$ onto $l_i$. For simplicity we denote the points in $P$ from left to right on $l_i$ by $p_1, \ldots, p_n$. Let $d_i$ denote the horizontal distance between $p_1$ and $p_n$. Let $b_{ij}$, $1 \leq j \leq \gamma$, be the point on $l_i$ at distance $\frac{jd_i}{\gamma+1}$ to the right of $p_1$. Let $L_{ij}$ and $R_{ij}$ be the set of points to the left and to the right of $b_{ij}$ respectively.

For each point $b_{ij}$ on $l_i$ we construct $\gamma$ pairs of lists, denoted $L'_{ijk}$ and $R'_{ijk}$, where $1 \leq k \leq \gamma$. A list $L'_{ijk}$ ($R'_{ijk}$) contains the set of points in $L_{ij}$ ($R_{ij}$) sorted according to the $l_k$-ordering. Such a list can be constructed in linear time since the ordering is given by the list $F_k$. (Actually it is not necessary to store the lists $L'_{ijk}$ and $R'_{ijk}$: we only need to store the first and the last point in each list.) Hence the total time complexity needed to construct the lists is

**Algorithm 1** Approx-MDdST($P, \varepsilon$)

**Ensure:** diam $\mathcal{T} \leq (1+\varepsilon)\, d_{\min}$
**Phase 1:** initializing
1: choose $\kappa \in (0, \min\{0.9\varepsilon, 1/2\})$
2: $\gamma \leftarrow \lceil 4/\kappa \rceil$
3: **for** $i \leftarrow 1$ **to** $\gamma$ **do**
4: $\quad l_i \leftarrow$ line with angle $i\frac{\pi}{\gamma}$ to the horizontal
5: $\quad F_i \leftarrow l_i$-ordering of $P$
6: **end for**
7: **for** $i \leftarrow$ **to** $\gamma$ **do**
8: $\quad$ rotate $P$ and $l_i$ such that $l_i$ is horizontal
9: $\quad$ let $p_1, \ldots, p_n$ be the points in $F_i$ from left to right
10: $\quad d_i \leftarrow |p_1.x - p_n.x|$
11: $\quad$ **for** $j \leftarrow 1$ **to** $\gamma$ **do**
12: $\quad\quad b_{ij} \leftarrow$ point on $l_i$ at dist. $j\frac{d_i}{\gamma+1}$ to the right of $p_1$
13: $\quad\quad$ **for** $k \leftarrow 1$ **to** $\gamma$ **do**
14: $\quad\quad\quad L'_{ijk} \leftarrow l_k$-ordered subset of $F_k$ to the left of $b_{ij}$
15: $\quad\quad\quad R'_{ijk} \leftarrow l_k$-ordered subset of $F_k$ to the right of $b_{ij}$
16: $\quad\quad$ **end for**
17: $\quad$ **end for**
18: **end for**
**Phase 2:** computing approximate farthest neighbors
19: **for** $i \leftarrow 1$ **to** $\gamma$ **do**
20: $\quad$ **for** $j \leftarrow 1$ **to** $\gamma$ **do**
21: $\quad\quad$ **for** $k \leftarrow 1$ **to** $n$ **do**
22: $\quad\quad\quad N(p_k, i, j, L) \leftarrow p_k$ (dummy)
23: $\quad\quad\quad$ **for** $l \leftarrow 1$ **to** $\gamma$ **do**
24: $\quad\quad\quad\quad p_{\min} \leftarrow$ first point in $L'_{ijl}$
25: $\quad\quad\quad\quad p_{\max} \leftarrow$ last point in $L'_{ijl}$
26: $\quad\quad\quad\quad$ **if** $|p_k p_{\min}| > |p_k p_{\max}|$ **then**
27: $\quad\quad\quad\quad\quad f \leftarrow p_{\min}$
28: $\quad\quad\quad\quad$ **else**
29: $\quad\quad\quad\quad\quad f \leftarrow p_{\max}$
30: $\quad\quad\quad\quad$ **end if**
31: $\quad\quad\quad\quad$ **if** $|p_k f| > |p_k N(p_k, i, j, L)|$ **then**
32: $\quad\quad\quad\quad\quad N(p_k, i, j, L) \leftarrow f$
33: $\quad\quad\quad\quad$ **end if**
34: $\quad\quad\quad$ **end for**
35: $\quad\quad$ **end for**
36: $\quad\quad$ repeat lines 21–35 with $R$ instead of $L$
37: $\quad$ **end for**
38: **end for**
**Phase 3:** testing pole candidates
39: $\tau = 8(\frac{1+\varepsilon}{(1+\varepsilon-(1+\kappa)(1+\kappa/24)} - 1)$
40: build WSPD for $P$ with separation constant $\tau$
41: $d \leftarrow \infty$ {smallest diameter so far}
42: **for** each pair $(A, B)$ in WSPD **do**
43: $\quad$ choose any two points $u \in A$ and $v \in B$
44: $\quad D \leftarrow \infty$ {approx. diam. of tree with poles $u$ and $v$}
45: $\quad l_{(u,v)} \leftarrow$ the line through $u$ and $v$
46: $\quad$ find $l_i$ such that $\angle l_i l_{(u,v)}$ is minimized
47: $\quad$ **for** $j \leftarrow 1$ **to** $\gamma$ **do**
48: $\quad\quad D \leftarrow \min\{D,$
$\quad\quad\quad\quad |N(u,i,j,L)u| + |uv| + |vN(v,i,j,R)|,$
$\quad\quad\quad\quad |N(u,i,j,R)u| + |uv| + |vN(v,i,j,L)|\}$
49: $\quad$ **end for**
50: $\quad$ **if** $D < d$ **then**
51: $\quad\quad u' \leftarrow u$ and $v' \leftarrow v$
52: $\quad\quad d \leftarrow D$
53: $\quad$ **end if**
54: **end for**
55: compute $\mathcal{T} \leftarrow \mathcal{T}_{u'v'}$
56: **return** $\mathcal{T}$

$O(\gamma^3 n + \gamma n \log n)$, see lines 1–18 in Algorithm 1. These lists will help us to compute an approximate farthest neighbor in $L_{ij}$ and $R_{ij}$ for each point $p \in P$ in time $O(\gamma)$, as will be described below.

**Phase 2: Computing approximate farthest neighbors.** Compute, for each point $p$, an approximate farthest neighbor in $L_{ij}$ and an approximate farthest neighbor in $R_{ij}$, denoted $N(p, i, j, L)$ and $N(p, i, j, R)$ respectively. This can be done in time $O(\gamma)$ by using the lists $L'_{ijk}$ and $R'_{ijk}$: just compute the distance between $p$ and the first respectively the last point in each list. There are $\gamma$ lists for each pair $(i, j)$ and given that at most two entries in each list have to be checked, an approximate farthest neighbor can be computed in time $O(\gamma)$. Hence the total time complexity of this phase is $O(\gamma^3 n)$, as there are $O(\gamma^2 n)$ triples of type $(p, i, j)$. The error we make by using approximate farthest neighbors is small:

OBSERVATION 2. *If $p$ is any point in $P$, $p_L$ the point in $L_{ij}$ farthest from $p$ and $p_R$ the point in $R_{ij}$ farthest from $p$, then (a) $|pp_L| \leq (1+\kappa/24) \cdot |pN(p,i,j,L)|$ and (b) $|pp_R| \leq (1+\kappa/24) \cdot |pN(p,i,j,R)|$.*

PROOF. Due to symmetry it suffices to check (a). If the algorithm did not select $p_L$ as farthest neighbor it holds that for each of the $l_i$-orderings there is a point further from $p$ than $p_L$. Hence $p_L$ must lie within a symmetric $2\gamma$-gon whose edges are at distance $|pN(p,i,j,L)|$ from $p$. This implies that $|pN(p,i,j,L)| \geq |pp_L| \cos(\pi/(2\gamma)) \geq |pp_L|/(1+\kappa/24)$, using some basic calculus and $\kappa \leq 1/2$. □

**Phase 3: Testing pole candidates.** Compute the WSPD for $P$ with separation constant $\tau$. To be able to guarantee a $(1+\varepsilon)$-approximation algorithm the value of $\tau$ will depend on $\varepsilon$ and $\kappa$ as follows:

$$\tau = 8\left(\frac{1+\epsilon}{1+\epsilon-(1+\kappa)(1+\kappa/24)} - 1\right).$$

Note that the above formula implies that there is a trade-off between the values $\tau$ and $\kappa$, which can be used to fine-tune which part of the algorithm contributes how much to the uncertainty and to the running time. Setting for instance $\kappa$ to $0.9\varepsilon$ yields for $\varepsilon$ small $16/\varepsilon + 15 < \tau/8 < 32/\varepsilon + 31$, i.e. $\tau = \Theta(\frac{1}{\varepsilon})$. For each pair $(A, B)$ in the decomposition we select two arbitrary points $u \in A$ and $v \in B$. Let $l_{(u,v)}$ be the line through $u$ and $v$. Find the line $l_i$ that minimizes the angle between $l_i$ and $l_{(u,v)}$. That is, the line $l_i$ is a close approximation of the direction of the line through $u$ and $v$. From above we have that $l_i$ is divided into $\gamma+1$ intervals of length $\frac{d_i}{\gamma+1}$. For each value of $j$, $1 \leq j \leq \gamma$, compute $\min(|N(u,i,j,L)u| + |uv| + |vN(v,i,j,R)|, |N(u,i,j,R)u| + |uv| + |vN(v,i,j,L)|)$. The smallest of these $O(\gamma)$ values is saved, and is a close approximation of the diameter of $\mathcal{T}_{uv}$, which will be shown below.

The number of pairs in the WSPD is $O(\tau^2 n)$, which implies that the total running time of the central loop of this phase (lines 42–54 in Algorithm 1) is $O(\gamma \cdot \tau^2 n)$. Building the WSPD and computing $\mathcal{T}_{u'v'}$ takes an extra $O(\tau^2 n + n \log n)$ time. Thus the whole algorithm runs in $O(\gamma^3 n + \gamma\tau^2 n + \gamma n \log n)$ time and uses $O(n \log n + \gamma^2 n + \tau^2 n)$ space. Setting

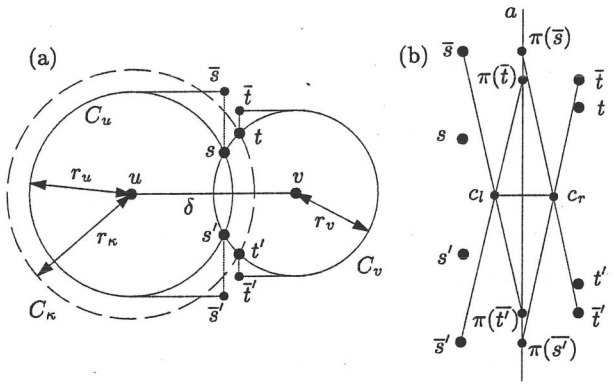



**Figure 1: A valid cut.**

$\kappa = 0.9\varepsilon$ yields $\gamma = O(\frac{1}{\varepsilon})$ and $\tau = O(\frac{1}{\varepsilon})$ and thus the time and space complexities we claimed. It remains to prove that the diameter of the dipolar tree that we compute is indeed at most $(1+\varepsilon) \cdot d_{\min}$.

## 5. THE PROOF OF CORRECTNESS

From Observation 1 we know that we will test a pair of poles $u$ and $v$ for which diam $\mathcal{T}_{uv} \leq (1 + 8/\tau)\ d_{\min} = \frac{1+\epsilon}{(1+\kappa)(1+\kappa/24)}\ d_{\min}$. The equality actually explains our choice of $\tau$. In this section we will prove that our algorithm always computes a dipolar tree whose diameter is at most $(1+\kappa)(1+\kappa/24)$ diam $\mathcal{T}_{uv}$ and thus at most $(1+\epsilon)\ d_{\min}$.

Consider the tree $\mathcal{T}_{uv}$. For simplicity we rotate $P$ such that the line $l$ through $u$ and $v$ is horizontal and $u$ lies to the left of $v$, as illustrated in Figure 1a. Let $\delta = |uv|$. Our aim is to prove that there exists an orthogonal cut that splits the point set $P$ into two sets such that the tree obtained by connecting $u$ to all points to the left of the cut and connecting $v$ to all points to the right of the cut will give a tree whose diameter is a $(1+\kappa)$-approximation of diam $\mathcal{T}_{uv}$. Since the error introduced by approximating the farthest neighbor distances is not more than a factor of $(1+\kappa/24)$ according to Observation 2, this will prove the claim in the previous paragraph.

Denote by $C_u$ and $C_\kappa$ the circles with center at $u$ and with radius $r_u$ and $r_\kappa = r_u + \kappa z$ respectively, where $z =$ diam $\mathcal{T}_{uv} = \delta + r_u + r_v$. Denote by $C_v$ the circle with center at $v$ and with radius $r_v$. Let $s$ and $s'$ ($t$ and $t'$) be two points on $C_u$ ($C_v$) such that if $C_u$ ($C_\kappa$) and $C_v$ intersect then $s$ and $s'$ ($t$ and $t'$) are the two intersection points, where $s$ ($t$) lies above $s'$ ($t'$). Otherwise, if $C_u$ ($C_\kappa$) and $C_v$ do not intersect, then $s = s'$ ($t = t'$) is the intersection of the line segment $(u, v)$ and $C_u$ ($C_\kappa$), see Figure 1a.

We say that a cut with a line $l_\kappa$ is *valid* iff all points in $P$ to the left of $l_\kappa$ are contained in $C_\kappa$ and all points of $P$ to the right of $l_\kappa$ are contained in $C_v$. A valid cut guarantees a dipolar tree whose diameter is at most $\delta + r_\kappa + r_v = (1+\kappa) \cdot$ diam $\mathcal{T}_{uv}$.

We will prove that the algorithm above always considers a valid cut. For simplicity we assume that $r_u \geq r_v$. We will show that there always exists a point $b_{ij}$ on $l_i$ such that cutting $l_i$ orthogonally through $b_{ij}$ is valid. Actually it is enough to show that the two requirements below are valid for any $\mathcal{T}_{uv}$. For a point $p$, denote the $x$-coordinate and the $y$-coordinate of $p$ by $p.x$ and $p.y$, respectively. For simplicity we set $u = (0, 0)$.

$$\text{(i)} \qquad \frac{z}{\gamma+1} \cdot \frac{1}{\cos\frac{\pi}{2\gamma}} \leq \frac{1}{2}(t.x - s.x), \quad \text{and}$$

$$\text{(ii)} \qquad \tan\frac{\pi}{2\gamma} \leq \frac{t.x - s.x}{2(r_u + r_v)}.$$

The reason for this will now be explained. First we need to define some additional points. The reader is encouraged to study Figure 1 for a visual description. Let $\overline{s} = (s.x, r_u)$, $\overline{s'} = (s'.x, -r_u)$, $\overline{t} = (t.x, r_v)$ and $\overline{t'} = (t'.x, -r_v)$. Let $a$ be the perpendicular bisector of the projections of $s$ and $t$ on the $x$-axis and let $\pi$ be the orthogonal projection of the plane on $a$. Now we can define $c_l$ to be the intersection point of the lines $(\overline{s}, \pi(\overline{t'}))$ and $(\overline{s'}, \pi(\overline{t}))$, and $c_r$ to be the intersection point of the lines $(\overline{t}, \pi(\overline{s'}))$ and $(\overline{t'}, \pi(\overline{s}))$.

It now follows that any bisector $l'$ that intersects the three line segments $(\overline{s}, \overline{t})$, $(c_l, c_r)$ and $(\overline{s'}, \overline{t'})$, will be a valid cut. This follows since all points to the left of $l'$ will be connected to $u$ and all points to the right of $l'$ will be connected to $v$, and the diameter of that tree will, obviously, be bounded by $\delta + (r_u + \kappa z) + r_u$ which is a $(1+\kappa)$-approximation of diam $\mathcal{T}_{uv}$.

From the algorithm we know that (a) there is a line $l_i$ such that $\angle(l_i, l_{(u,v)}) \leq \pi/(2\gamma)$, and that (b) there are $\gamma$ orthogonal cuts of $l_i$ that define equally many partitions of $P$. The distance between two adjacent orthogonal cuts of $l_i$ is at most $z/(\gamma+1)$. This implies that the length of the largest interval on $l_{(u,v)}$ that is not intersected by any of these orthogonal cuts is at most

$$\frac{1}{\cos\frac{\pi}{2\gamma}} \cdot \frac{z}{\gamma+1}.$$

Hence requirement (i) ensures that for every $\mathcal{T}_{uv}$ the distance $|c_l c_r| = (t.x - s.x)/2$ must be large enough to guarantee that there is an orthogonal cut of $l_i$ that intersects it.

An orthogonal cut of $l_i$ has an angle of at least $\pi/2 - \pi/(2\gamma)$ to $l_{(u,v)}$. To ensure that an orthogonal cut of $l_i$ that intersects the line segment $\overline{c_l c_r}$ also passes between $\overline{s}$ and $\overline{t}$ and between $\overline{s'}$ and $\overline{t'}$ it suffices to add requirement (ii).

It remains to prove the following lemma which implies that for every $\mathcal{T}_{uv}$ there is a valid orthogonal cut.

LEMMA 3. *For any $u, v \in P$ ($u \neq v$) the tree $\mathcal{T}_{uv}$ fulfills the requirements (i) and (ii).*

PROOF. The tree $\mathcal{T}_{uv}$ can be characterized by the relationship of the two ratios

$$\alpha := \frac{\delta}{r_u + r_v} \quad \text{and} \quad E := \frac{1 + \kappa/2}{1 - \kappa/2}.$$

We distinguish three cases: (1) $\alpha < 1$, (2) $1 \leq \alpha \leq E$, and (3) $\alpha > E$. For each of these three cases we will show that $\mathcal{T}_{uv}$ fulfills the two requirements.

**Case 1:** Using the following two straight-forward equalities, $s.x^2 + s.y^2 = r_u^2$ and $(\delta - s.x)^2 + s.y^2 = r_v^2$, we obtain that $s.x = (\delta^2 + r_u^2 - r_v^2)/(2\delta)$. A similar calculation for $t.x$ yields $t.x = (\delta^2 + r_\kappa^2 - r_v^2)/(2\delta)$. Inserting these values gives $t.x - s.x = (\kappa^2 z^2 + 2\kappa z r_u)/(2\delta)$. The fact that $\alpha \leq E$ allows us to further simplify the expression for $t.x - s.x$ by using the following two expressions:

$$\frac{z}{\delta} = \frac{\delta + r_u + r_v}{\delta} = 1 + \frac{r_u + r_v}{\delta} \geq \frac{2}{1 + \kappa/2}, \quad \text{and}$$

$$\frac{r_u}{\delta} \geq \frac{1 - \kappa/2}{2(1 + \kappa/2)}.$$

From this we obtain that

$$t.x - s.x = \frac{\kappa z}{2}\left(\frac{\kappa z}{\delta} + \frac{2r_u}{\delta}\right) > \frac{\kappa z}{2}.$$

This fulfills requirement (i) since

$$\frac{z}{\gamma + 1} \cdot \frac{1}{\cos\frac{\pi}{2\gamma}} \leq \frac{\kappa z}{4} \leq \frac{1}{2}(t.x - s.x). \qquad (1)$$

For requirement (ii) note that $\tan \pi/(2\gamma) \leq 2\kappa \tan \pi/16 < 2\kappa/5$. Since $\kappa \leq 1/2$ we get that $z/\delta \geq 2/(1 + \kappa/2) \geq 8/5$. Combining this inequality, Equality 1, and our assumption that $r_u \geq r_v$ shows that requirement (ii) is also fulfilled:

$$\frac{t.x - s.x}{2(r_u + r_v)} \geq \frac{\kappa z}{4\delta}\left(\frac{2r_u + \kappa z}{r_u + r_v}\right) \geq \frac{\kappa z}{4\delta} \geq \frac{2\kappa}{5}.$$

**Case 2:** In this case we argue in the same manner as in the previous case. Using the fact that $s.x = r_u$ and $t.x = (\delta^2 + r_\kappa^2 - r_v^2)/(2\delta)$ yields

$$t.x - s.x \geq \frac{\kappa z}{2}\left(\frac{\kappa z}{\delta} + \frac{2r_u}{\delta}\right) > \frac{\kappa z}{2}.$$

The rest of the proof is exactly as in case 1.

**Case 3:** The first requirement is already shown to be fulfilled since $t.x - s.x \geq \delta - r_u - r_v \geq \kappa z/2$, hence it remains to show requirement (ii). We have

$$\frac{t.x - s.x}{2(r_u + r_v)} \geq \frac{\delta - (r_u + r_v)}{2(r_u + r_v)}$$

plugging in the values gives $\kappa/(2-\kappa)$, which is at least $2\kappa/5$. The lemma follows. □

The lemma says that for every dipole $\{u, v\}$ there exists a line $a$ such that the dipolar tree obtained by connecting all the points on one side of $a$ to $u$ and all the points on the opposite side to $v$, is a $(1 + \kappa)$-approximation of $\mathcal{T}_{uv}$.

## Conclusions

While it seems to be hard to reduce the running time for computing an exact MDST to $o(n^3)$, we have given a fast PTAS for approximating the MDST. It computes in $O(n \log n)$ time and space a tree whose diameter is at most $(1+\varepsilon)$ times that of a MDST. The runtime dependency on $\varepsilon$ is moderate. The PTAS also works for higher dimensional point sets, but the running time increases exponentially with the dimension. A similar scheme yields an $O(n \log n)$-time PTAS for the discrete two-center problem.

The main novelty of our scheme is a discretization of the orientations of line segments to a constant number of orientations. We only consider projections of the input points onto those directions, which gives us fast access to farthest neighbors.

## 6. REFERENCES


[1] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to $k$-nearest-neighbors and $n$-body potential fields. *Journal of the ACM*, 42(1):67–90, January 1995.

[2] Joachim Gudmundsson, Herman Haverkort, Sang-Min Park, Chan-Su Shin, and Alexander Wolff. Approximating the geometric minimum-diameter spanning tree. Technical Report 4/2002, Institut für Mathematik und Informatik, Universität Greifswald, February 2002. Available at `http://www.math-inf.uni-greifswald.de/~awolff/pub/ghpsw-agmds-02t.pdf`.

[3] Refael Hassin and Arie Tamir. On the minimum diameter spanning tree problem. *Information Processing Letters*, 53(2):109–111, 1995.

[4] Jan-Ming Ho, D. T. Lee, Chia-Hsiang Chang, and C. K. Wong. Minimum diameter spanning trees and related problems. *SIAM Journal on Computing*, 20(5):987–997, October 1991.

# Drawing Free Trees Inside Rectilinear polygons Using Polygon Skeleton

[Extended Abstract]

Alireza Bagheri
Software Systems Research and Development Laboratory
Department of Computer Engineering
Amir-Kabir University of Technology, Tehran, Iran
098-021-6018208
bagheri@ce.aku.ac.ir

Mohammadreza Razzazi
Software Systems Research and Development Laboratory
Department of Computer Engineering
Amir-Kabir University of Technology, Tehran, Iran
098-021-64542733
razzazi@ce.aku.ac.ir

## ABSTRACT

In drawing a graph inside a polygon beside reduction of edge crossing, the avoidance of intersection of edges with sides of the polygon is of primary concern. In this paper we introduce a new algorithm for drawing free trees inside a rectilinear polygon by using properties of polygons to guide Simulated Annealing (SA) method.

## General Terms

Algorithms

## Keywords

Graph Drawing, Simulated Annealing, Polygon Skeleton required for Proceedings

## 1. INTRODUCTION

Trees are known structures that have many applications. Hence drawing trees "nicely" has been investigated by many researchers. There are some aesthetics for nice drawing of graphs (trees) that are mentioned in the literature. Some of the most important aesthetics are: minimizing number of edge crossing, minimizing number of bends per edge, increasing symmetry of drawing, maximizing amount of angles between two edges, and distributing vertices all over the region [8 and 17].

Most of graph drawing algorithms do drawing inside a rectangle. But in some texts it may have special effects to draw graphs inside general polygons in the middle of the text. In this paper we consider drawing of free trees inside a rectilinear polygon by means of straight skeleton of the polygon and simulated annealing (SA) method. We assume that the reader is familiar with SA method.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

Drawing of large graphs by algorithms that use clustering usually has much fewer edge crossings and are nicer than drawing results of ones that do not use it [4, 6, 7, 14, 18, 19 and 20]. The algorithms that use clustering divide vertices of graphs into some groups based on some parameters, and draw vertices of the same groups near each other [11, 12 and 13]. Our algorithm uses clustering and spreads vertices of trees all over the inner region of the given polygon by using straight skeleton of the polygon [1, 2, 3, 15, and 16]. We may have some bends in edges of the tree in our drawing.

In section 2, straight skeletons are briefly described. In section 3, our algorithm is introduced. In section 4, some drawing results of our algorithm are illustrated and compared to drawing results of [10], which uses SA method. In section 5, conclusion is stated.

## 2. STRAIGHT SKELETONS

There are two types of skeleton for simple polygons, medial axis, and straight skeleton. Medial axis of a given simple polygon, P, consists of all interior points whose closest point on the boundary of P is not unique [9]. While medial axis is a voronoi-diagram-like concept, straight skeleton is not defined using a distance function but rather by an appropriate shrinking process. Straight skeleton is defined as the union of the pieces of angular bisectors traced out by polygon vertices during the shrinking process. [1, 2, 3, 15, and 16].

Straight skeleton, in general, differs from medial axis, if P is convex then both structures are identical. Otherwise, the medial axis contains parabolically curved segments around reflex vertices of P, which are avoided by the straight skeleton. In this paper, we consider drawing trees inside rectilinear polygons, and to avoid parabolically curved segments we use straight skeleton as the skeleton of polygons. In the following, by polygon skeleton we mean straight skeleton of the polygon. The skeleton of a given polygon, P, partitions the interior of P into n connected areas which we call them faces. Each face is related to just one edge of P. Bisector pieces are called arcs, and their endpoints which are not vertices of P are called nodes of the skeleton. When P is simple, the structure is tree and arcs are straight line

segments. Figure 1 shows skeleton of a rectangle.

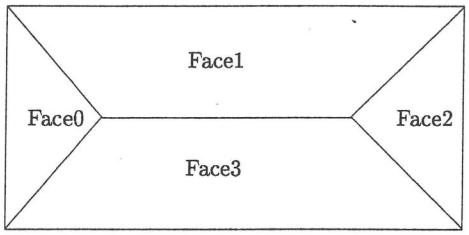



Figure 1. Straight skeleton of a rectangle

## 3. THE ALGORITHM

In this section, after giving some definitions we introduce our algorithm (From now on, for simplicity, we use trees for free trees, and polygon for rectilinear polygon).

*Definition 1.* Let FaceSet(j) be set of all faces that include node j of the given skeleton.

*Definition 2.* Let CloserSet(j, k) be set of all nodes of the given tree whose paths to node j is shorter than to node k.

Here, we describe our algorithm briefly. The main layout of the algorithm is as follows:

*Free Trees Drawing Algorithm*
*a. Compute the polygon skeleton and area of the faces.*
*b. Compute weight of the nodes of the skeleton.*
*c. Compute weights of the edges of the skeleton.*
*d. Compute weights of the edges of the tree.*
*e. Do the mapping of the tree onto the skeleton.*
*f. Remove the crossings between edges of the tree and the border of the polygon.*
*g. Draw the tree using SA method.*

In the following we describe steps of the algorithm.

**a. Computing the polygon skeleton and area of the faces.** We obtain straight skeleton of the given polygon by using [9]. Since all faces are simple polygons we can use the following formula to compute the area of the faces [5].

$$\frac{1}{2}\sum_{i=3D0}^{n-1}(X_iY_{i+1} - X_{i+1}Y_i)$$

Here $(X_j, Y_j)$ are the coordinates of vertex j ($j = 3D0..N - 1$) of the given face; $X_n = 3DX_0$ and $Y_n = 3DY_0$. To use the above formula, vertices of the faces should be ordered clockwise or counter clockwise.

**b. Computing weights of the nodes of the skeleton.** For each node j of the skeleton, we compute the following sum as the weight of node j:

$$Weight(j) = 3D \sum_{f \epsilon FaceSet(j)} Area(f)$$

In fact, this weight approximately represents the amount of area which includes the node.

**c. Computing weights of the edges of the skeleton.** For each edge (j, k) of the skeleton, we assign two weights, where each weight is assocaited to one of its endpoints.

$$W_{jk} = 3D \sum_{m \epsilon CloserSet(j,k)} Weight(m)$$

$$W_{kj} = 3D \sum_{m \epsilon CloserSet(k,j)} Weight(m)$$

$$Weight(j,k) = 3D(W_{jk}, W_{kj})$$

The weight of associated to an endpoint approximately represents the amount of area of the polygon which is closer to this endpoint than the other one.

**d. Computing weights of the edges of the tree.** For each edge (j, k) of the tree, we associate a weight computed as follows:

$$Weight(j,k) = 3D(|CloserSet(j,k)|, |CloserSet(k,j)|)$$

**e. Mapping the tree onto the skeleton.** We consider the weighted skeleton (the weighted tree), and find an edge of the skeleton (the tree0 such that the difference of the weights of its endpoints is minimum (i.e. we find the middle edge). We map the middle edge of the skeleton to the middle edge of the tree. Then we omit these two middle edges from the skeleton and the tree, this divides the tree and the skeleton respectively into two sub-trees and two sub-skeletons. Then we update the weights of the edges of the two sub-skeletons and the two sub-trees. To do this, we decrease one of the weights of the edges of each sub-skeleton (sub-tree) by the total weight of the other sub-skeleton (sub-tree). In order to determine that which weight of an edge needs to be decreased, we consider each sub-skeleton (sub-tree) as a directed tree whose root is the endpoint of the omitted middle edge that is connected to this sub- skeleton (sub-tree). (Considering the direction of each edge from the father to the child). The weight associated to the beginning point of each edge should be decreased. After updating the weights of the sub- skeletons (sub-trees), we do the mapping procedure recursively on the sub-skeletons and the sub-trees. The termination condition is satisfied when the number of the nodes of the sub-trees or the number of the edges of the sub-skeletons becomes less than one.

A skeleton or a tree may have more than one middle edge; it is easy to see that in this case these edges share an endpoint. We call this endpoint, the middle node. In this case, we omit from the tree the middle node and its incident edges, so the

tree is divided into two or more sub-trees. To update the weights associated to the edges of each sub-tree, we should decrease one of the weights of each sub-tree edge by the sum of number of nodes of all the other sub-trees plus one. But, we omit from the skeleton just the middle node, so the skeleton is divided into two or more sub-skeletons. Then we update the weights of the edges of each sub-skeleton. To do this, we should decrease one of the two weights of the sub-skeleton edges, by sum of the weights of the edges of the other sub-skeletons, which were incident to the deleted node.

In order to determine that the weight of which endpoint of each edge should be decreased, we consider each sub-skeleton (sub-tree) as a directed tree whose root is the middle node. (Considering the direction of each edge from the father to the child). The weight of the beginning point of each edge should be decreased. In each case, we divide the sub-skeletons and sub-trees into some possibly balanced groups, and for each group of the sub-skeletons, and its related group of the sub-trees, we call the mapping procedure recursively. Therefore, we do the mapping by using one of the following ways:

1. The middle node of the tree is mapped onto the middle node of the skeleton.

2. The middle node of the tree is mapped onto the middle edge of the skeleton.

3. The middle edge of the tree is mapped onto the middle edge of the skeleton.

4. The middle edge of the tree is mapped onto the middle node of the skeleton.

In case 1, the node of the tree is mapped onto the node of the skeleton. In case 2, the node of the tree is mapped onto the middle point of the edge of the skeleton. In cases 3 and 4, first we substitute the edge of the tree with a path of length 2 whose endpoints are the endpoints of the middle edge and its middle vertex is a virtual vertex. Then the virtual vertex of the tree is mapped onto the related middle node or the middle point of the related middle edge of the skeleton. Then cases 3 and 4 continues respectively as cases 2 and 1.

This mapping procedure provides a mapping list of the nodes of the tree which are mapped onto the corresponding points of the skeleton. This mapping list is used by SA method to spread the vertices of the tree all over the drawing region. After termination of SA method, all the virtual vertices are removed and the substituted tree edges are restored.

**f. Removing the crossings between edges of the tree and the border of the polygon.** As the starting configuration of the SA method, all the nodes of the tree, which are included in the mapping list, are located at related points of the skeleton. Then the remaining nodes of the tree are placed at the location of the closest located tree node. By the closest located node of a node, we mean the located node with the shortest path to the node among all the located nodes. When all the nodes of the tree are located, if the given polygon is non-convex, there is the possibility of crossing between edges of the tree and the border of the polygon. These crossings are removed by introducing some dummy nodes and bending the crossing edges of the tree. The resulting configuration is the starting configuration of SA method.

**g. Drawing the tree using SA method.** Now, we use SA method to draw the tree inside the given polygon. we try to keep the nodes of the tree close to their corresponding points of the skeleton by considering a factor in the goal function of SA method. Our algorithm guides SA method, so it converges faster than the drawing algorithm introduced in [10], which uses ordinary time consuming SA method. Let us for simplicity call the algorithm of [10] the SA algorithm.

## 4. DRAWING RESULTS

In this section, we compare some drawing results of our algorithm and the SA algorithm. Figures 2 and 3, respectively, show the drawings of a complete binary tree that consists of seven nodes by our algorithm and the SA algorithm.

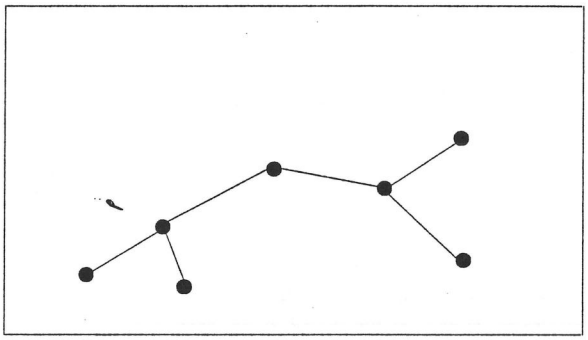


Figure 2. Drawing of a 7-nodes complete binary tree by SA algorithm

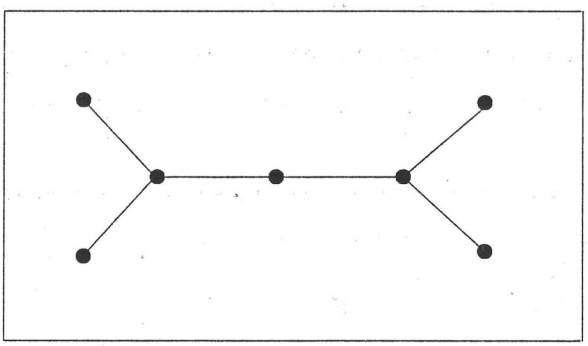


Figure 3. Drawing of a 7-nodes complete binary tree by our algorithm

The drawing of a complete binary tree with 63 vertices by the SA algorithm , inside a square, is shown in Figure 4. As can be seen, although the tree is planar the drawing of it by the SA algorithm is not planar. The drawing result of this

tree by our algorithm is shown in Figure 5.

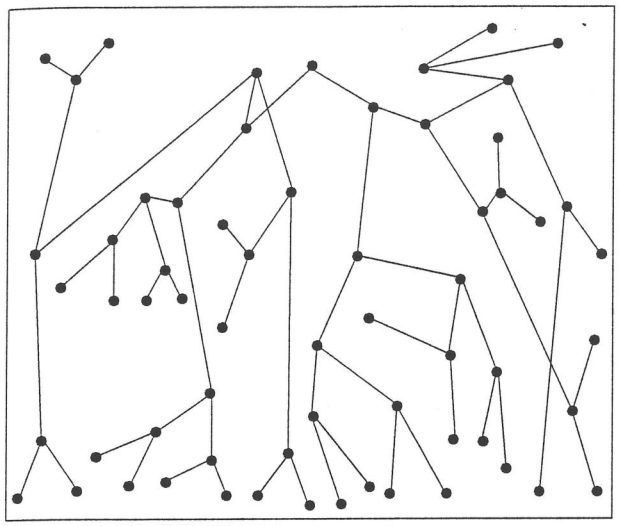


Figure 4. Drawing of a 63-nodes complete binary tree by SA algorithm

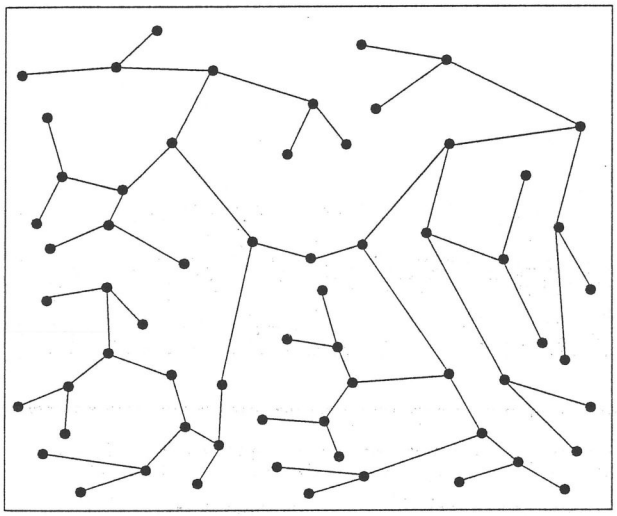


Figure 5. Drawing of a 63-nodes complete binary tree by our algorithm

In all the following examples, drawing of a 31-nodes complete binary tree inside some rectilinear polygons is presented. Figures 6, 7, and 8 illustrate the drawings of the tree respectively inside U-shape, T-shape, and S-shape rectilinear polygons by our algorithm.

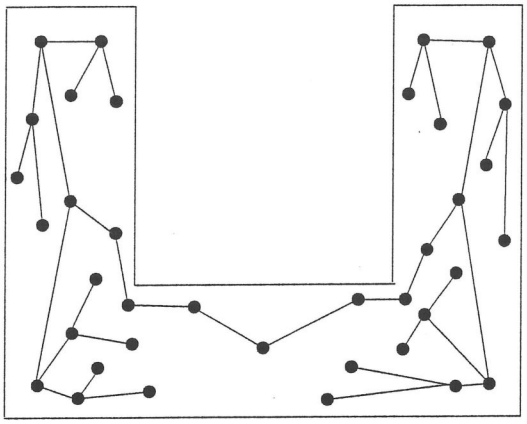


Figure 6. Drawing of a 31-nodes complete binary tree by our algorithm inside a U-shape rectilinear polygon

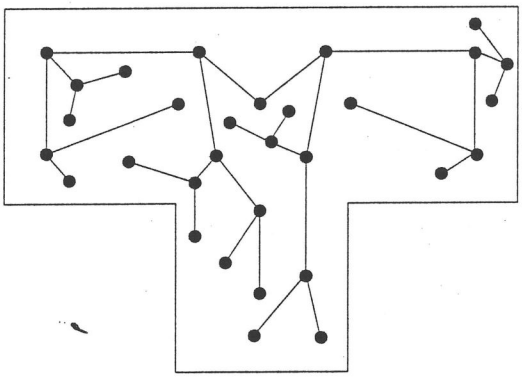


Figure 7. Drawing of a 31-nodes complete binary tree by our algorithm inside a T-shape rectilinear polygon

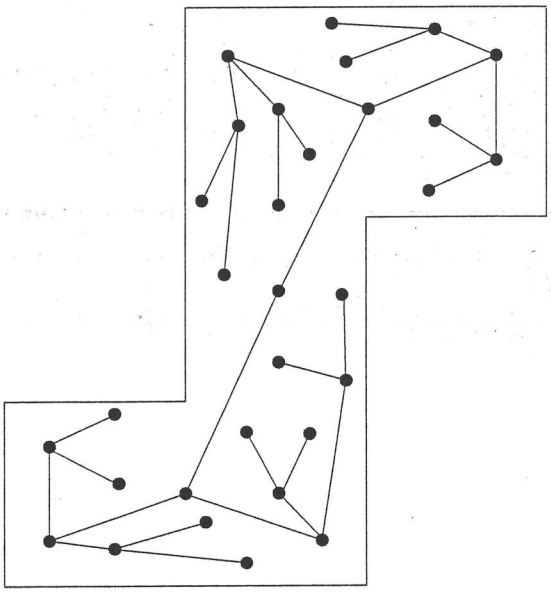


Figure 8. Drawing of a 31-nodes complete binary tree by our algorithm inside a S-shape rectilinear polygon

## 5. CONCLUSION

In this paper we introduced a new algorithm that uses straight skeletons of polygons and simulated annealing method to draw free trees inside rectilinear polygons. The drawing results show that our algorithm draws trees nicer than the previous known algorithms, even when trees are relatively large, with respect to the aesthetics that we mentioned at the introduction section.

## 6. REFERENCES


[1] O. Aichholzer and F. Aurenhammer, "Straight Skeletons for General Polygonal Figures in the Plane", Proc. 2nd Ann. Int. Conf. Computing and Combinatorics (COCOON-96), Lecture Notes in Computer Science (LNCS)1090, Springer , pp. 117-126, 1996.

[2] O. Aichholzer, F. Aurenhammer, D. Alberts and B. Gartner, "Straight Skeletons of Simple Polygons", Proceedings of the 4th International Symposium at LIESMARS-95, Wuhan/China, October 1995.

[3] O. Aichholzer, F. Aurenhammer, D. Alberts and B. Gartner, "A Novel Type of Skeleton for Polygons", Journal of Universal Computer Science 1(12), pp. 752-761, 1995.

[4] D. Behrens, E. Barke and R. Tolkiehn, "Design Driven Partitioning", Asp-DAC-97: 2nd Asian and South Pacific Design Automation Conference, Chiba, Japan, 1997.

[5] P. Bourke, "Calculating the area and centroid of a polygon", http://www.swin.edu.au/astronomy/pbourke/geometry/polyarea/, 1988.

[6] F. J. Brandenburg, "Graph Clustering I: Cycles of Cliques", Proceedings of Graph Drawing 97, LNCS 1353, 1997.

[7] F. J. Brandenburg and A. Sen, "Graph Clustering II: Trees of Cliques with Size Bounds", http://www.infosun.fmi.uni-passau.de/br/lehrstuhlMitarbeiter/Brandenburg/publikationen/, 1999.

[8] T. M. Chan, "A Near-Line Area Bound for Drawing Binary Trees", In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms, pp. 161-168, 1999.

[9] F. Chin, J. Snoeyink and C. A. Wang, "Finding the Medial Axis of a Simple Polygon in Linear Time", Proc. 6th Ann. Int. Symp. Algorithms and Computation (ISAAC 95), Lecture Notes in Computer Science 1004, pp. 383-391, 1995.

[10] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", ACM Transactions on Graphics, Vol. 15, No. 4, pp. 301-331, October 1996.

[11] P. Eades and Q. W. Feng, "Multilevel Visualization of Clustered Graphs", Symposium on Graph Drawing GD-96, ed. S. C. North, LNCS 1190, pp. 101-112, 1996.

[12] P. Eades, Q. W. Feng and X. Lin, "Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs", Symposium on Graph Drawing GD-96, ed. S. C. North, LNCS 1190, pp. 113-128, 1996.

[13] P. Eades, Q. W. Feng and H. Nagamochi, "Drawing Clustered Graphs on an Orthogonal Grid", Journal of Graph Algorithms and Applications, http://www.cs.brown.edu/publications/jgaa/, Vol. 3, No. 4, pp. 3-29, 1999.

[14] J. Edachery, A. Sen and F. J. Brandenburg, "Graph Clustering Using Distance-K Cliques", 7th International Symposium on Graph Drawing (GD-99), LNCS 1731, pp. 98-106, September 1999.

[15] D. Eppstein and J. Erickson, "Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions", 14th ACM Symposium on Computational Geometry, Minneapolis, pp. 58-67, 1998.

[16] P. Felkel and S. Obdrzalek, "Straight Skeleton Implementation", Proceedings of Spring Conference on Computer Graphics, ISBN 80-223-0837-4, pp. 210-218, http://www.cgg.cvut.cz/Publications/felkel-sccg-98.ps.gz, 1998.

[17] H. Purchase, "Which Aesthetic Has the Greatest Effect on Human Understanding?", 5th Symposium on Graph Drawing, Rome, Italy, LNCS 1353, September 18-20, pp. 248-261, 1997.

[18] T. Roxborough and A. Sen, 1997, "Graph Clustering Using Multiway Ratio Cut", Proceedings of Graph Drawing Symposium GD-97, LNCS 1353, Rome, pp. 291-296, September 1997.

[19] R. Sablowski and A. Frick, "Automatic Graph Clustering", Proceedings of Graph Drawing GD-96, Berkeley, California, LNCS 1190, pp. 396-400, September 1996.

[20] Tan X., J. Tong, P. Tan, N. Park and F. Lombardi, "An Efficient Multi-Way Algorithm for Balanced Partitioning of VLSI Circuits", Proceedings of the International Conference on Computer Design (ICCD-97), IEEE, pp. 608-613, 1997.

# Optimal Projections onto Grids

[Extended Abstract]

J.M. Díaz-Báñez
Univ. Sevilla, Spain
dbanez@us.es

F. Hurtado
Univ. Pol. Catalunya, Spain
hurtado@ma2.upc.es

M. A. López
Univ. Denver, USA
mlopez@cs.du.edu

J. A. Sellarès
Univ. Girona, Spain
sellares@ima.udg.es

## ABSTRACT

The problem of determining nice (regular, simple, minimum crossing, monotonic) and non-degenerate (with distinct $x$-coordinate, non-collinear, non-cocircular, non-parallel) orthogonal and perspective projections of a set of points or a set of disjoint line segments has been studied extensively in the literature for the theoretical case of infinite resolution images. In this paper we propose to extend the study of this type of problems to the case where the images have finite resolution. Applications dealing with such images are common in fields such as computer graphics and computer vision. We derive algorithms that solve the simplest problem of obtaining an optimal orthogonal projection of a two-dimensional or three-dimensional point set. Due to the high cost of the proposed algorithms we also present algorithms that provide approximate solutions to the problems.

## 1. INTRODUCTION

An object in two-dimensional and three-dimensional space is often represented by a set of points and line segments that act as its features. An optimal projection of an object is one that gives an image in which the features of the object, relevant for some task, are visible without ambiguity so that the image is as simple and readable as possible: no two points are projected to the same point, the projection of a segment is not reduced to a point, the number of crossings of the projection of a set of segments is minimum, no two projected points have the same $x$-coordinate, no three projected points are collinear, no four projected points are cocircular, no two projected segments are parallel, etc. The terms *nice* and *non degenerate projections* refers to these requirements and to many others. The problem of determining nice and non-degenerate orthogonal and perspective projections of sets of points and disjoint line segments has been studied extensively in the literature for the theoretical case of infinite resolution images [3, 5, 6, 7].

In the finite resolution model a $d$-dimensional point projects onto a cell (pixel) of a regular grid (all cells are disjoint but identical $k$-dimensional hypercubes, for $k < d$). For clarity, when the dimensionality of the grid is important, we explicitly refer to the grid as a *graduated line* or *graduated plane*, for example.

A regular orthogonal image of a point set is an image obtained by an orthogonal projection such that no two points project to the same cell. Since a regular orthogonal image of a point set does not always exist, we are interested in finding a projection such that the maximum number of projected points onto any cell is minimal. When this maximum equals one, the projection is *regular*.

In this paper we consider orthogonal projections only. In the sequel, unless stated otherwise, a projection always means an orthogonal projection. Furthermore, without loss of generality, we assume unit size grid cells (i.e., hypercubes of side length equal to one). Other cell sizes can be reduced to unit cells by scaling the input set about the origin. We study the problem of finding orthogonal projections that minimize the maximum number of points that project to the same grid cell. Specifically, we present both exact and approximation algorithms for the following problems:

1. Given a set of points in $\mathbb{R}^2$, determine a graduated line $\ell$ in $\mathbb{R}^2$ such that the maximum number of input points that project to the same cell of $\ell$ is minimal (or within a small factor of the minimal).
2. Given a set of points in $\mathbb{R}^3$, determine a graduated line $\ell$ or graduated plane $\pi$ in $\mathbb{R}^3$ such that the maximum number of input points that project to the same cell of $\ell$ or $\pi$ is minimal (or within a small factor of the minimal).

DEFINITION 1.1. *Let $d$ and $k$ be positive integers such that $k < d$. An instance of the $\mathrm{P}(d,k)$ problem is a set of points $\mathcal{P}$ in $\mathbb{R}^d$. A solution to this instance is a $k$-dimensional graduated hyper-plane $\pi$ such that the maximum number $m$ of points of $\mathcal{P}$ projected to the same cell of $\pi$ is minimal. We refer to $m$ as the cost of the solution.*

In [2, 1] a similar, but different problem is studied. Given a set of points in the plane the problem is to find an optimal set of $b$ equally spaced parallel lines such that the maximum number of points in a region bounded by two consecutive lines is minimized. In [4] a regular grid is considered. Notice that for these problems, unlike ours, the distance between adjacent parallel lines is not fixed beforehand.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

## 2. TWO-DIMENSIONAL CASE

In this section we describe exact and approximate solutions to P(2,1). The graduated line $\ell = \ell(\theta, u, v)$ is the line obtained from the $x$-axis by composing a rotation by angle $\theta \in [0, \pi]$ around the origin $o$ with a translation by vector $(u, v)$. The point $(u, v)$ is called the origin of the graduated line. Since we are assuming unit size grid cells, each number $z \in Z$ defines a cell $c_z = [l_z, r_z)$ in $\ell$ that is obtained from cell $[z, z+1)$ of the $x$-axis by the composite transformation described above.

A region of the plane bounded by two lines orthogonal to $\ell$ through consecutive cell boundaries is called a *bucket* and the participating lines are the *bucket boundaries.* Equivalently, a bucket is the locus of points in the plane that project to the same cell.

REMARK 2.1. *Let $\ell$ be a graduated line with unit normal $\vec{n}$ and unit direction vector $\vec{d}$. Let $\ell'$ be the graduated line obtained by translating $\ell$ by $z\vec{d} + r\vec{n}$, for any $z \in Z$ and $r \in R$. Then lines $\ell$ and $\ell'$ determine the same bucket sets.*

A solution to an instance $\mathcal{P}$ of P(2,1) is a graduated line $\ell$ that minimizes the maximum number of points from $\mathcal{P}$ that project to the same cell of $\ell$. Equivalently, $\ell$ is a graduated line that minimizes the maximum number of points of $\mathcal{P}$ in a bucket determined by $\ell$.

DEFINITION 2.1. *Two graduated lines whose bucket sets produce exactly the same partition of a point set $\mathcal{P}$ are said to be equivalent with respect to $\mathcal{P}$.*

Clearly, if $\ell$ is an optimal solution to an instance of P(2,1) so is any graduated line equivalent to $\ell$.

LEMMA 2.1. *Let $\ell$ be a solution to an instance $\mathcal{P}$ of P(2,1). There exists a graduated line $\ell_0$ equivalent to $\ell$ such that a point of $\mathcal{P}$ lies on the boundary of a bucket of $\ell_0$.*

We are now ready to describe an algorithm to solve P(2,1). The algorithm is based on two key ideas that allow us to discretize the problem:
1. As a consequence of Lemma 2.1, we only need to consider graduated lines that have a bucket boundary passing through a point $p_i$ of $\mathcal{P}$. Furthermore, from Remark 2.1 and without loss of generality, we can assume that $p_i$ is the origin of the graduated line.
2. Let $\ell$ be a graduated line with origin $p_i$. Every rotation of $\ell$ by an angle $\theta$ about its origin produces a partition of $\mathcal{P}$ into buckets. Since $\ell = \ell(\theta, u, v)$ and $\ell = \ell(\pi + \theta, u, v)$ produce equivalent partitions, it suffices to consider angles in $[0, \pi)$. A radial sweep of $\ell$ about $p_i$ produces a finite number of different partitions of $\mathcal{P}$. As $\ell$ rotates from $\theta = 0$ to $\pi$, the current partition changes only at angles where the left boundary line of a bucket passes through a point $p_j$ of $\mathcal{P}$, $p_j \neq p_i$. At this instant, $p_j$ changes from one bucket to the adjacent one. Consequently, in searching for the optimal solution for a fixed origin, it suffices to examine the finite collection $\mathcal{A}$ of these angles.

Note that while sweeping around $p_i$, a point $p_j$ contributes $O(d_{ij})$ events, where $d_{ij}$ denotes the distance from $p_i$ to $p_j$. In fact, each such event corresponds to a value $\theta$ such that $p_j$ lies on the line $x = k$ rotated about $p_i$ by angle *theta.* Notice also that if $d(p_i, p_j) < k$ the rotated line cannot possibly contain $p_j$. Otherwise, $p_j$ contributes one or two solutions, depending on whether $p_j$ lies on the boundary or outside, respectively, of the circle of radius $k$ centered at $p_i$. In fact, the solutions correspond to lines through $p_j$ tangent to this circle.

The algorithm, which we call **MGL2**, proceeds in two steps:
1. For each point $p_i$ of $\mathcal{P}$ perform a radial sweep around $p_i$ to find the graduated line $\ell_i$ with origin $p_i$ that minimizes the maximum number of points of $\mathcal{P}$ that fall in the same bucket. Call this minimum number $m_i$.
2. After all points of $\mathcal{P}$ have been processed, report the line $\ell$ that corresponds to $\min\{m_1, \ldots, m_n\}$.

THEOREM 2.1. *An instance $\mathcal{P}$ of problem P(2,1) can be solved in $O(tn^2 \log tn)$ time and $O(tn)$ space, where $t$ is the diameter of $\mathcal{P}$.*

### 2.1 Approximate solutions to P(2,1)

The complexity of algorithm **MGL2** is high. In this section we describe more efficient approximation algorithms that compute answers that are guaranteed to be within a small constant factor of the optimal.

When the origin $(u, v)$ of the graduated line is fixed, and the only degree of freedom is the rotation angle $\theta$, we obtain a restricted instance AP(2,1) of problem P(2,1). When the angle $\theta$ of the projection line is fixed, but we are allowed to shift the projection line over itself, we obtain a restricted instance SP(2,1) of problem P(2,1).

It is natural to consider solutions to the restricted instances AP(2,1) and SP(2,1) as approximations to the solution of P(2,1). In the remainder of the section we investigate the quality of the solutions obtained by using these approximations.

The solution to problem AP(2,1) can be obtained by performing one iteration of **MGL2**, using $p = (u, v)$ (instead of $p_i$) as the designated origin. Let $t = \max_{1 \leq j \leq n}\{\text{dist}(p, p_j)\}$. We have

THEOREM 2.2. *An instance of problem AP(2,1) can be solved in $O(tn \log tn)$ time and $O(tn)$ space.*

Consider now problem SP(2,1). Since shifting by $s$ is equivalent to shifting by $s$ mod1, it suffices to consider shifts smaller than 1. This fact, combined with Lemma 2.1, allows us to establish the following.

THEOREM 2.3. *An instance of problem SP(2,1) can be solved in $O(n \log n)$ time.*

An interesting property of problem SP(2,1) is that the costs of any two candidate graduated lines cannot differ by more than a factor of 2.

LEMMA 2.2. *Let $\ell$ be a projection with cost $C$ and let $C'$ be the cost of a projection $\ell'$ obtained by shifting $\ell$ by an arbitrary amount. Then, $\frac{1}{2} \leq \frac{C'}{C} \leq 2$.*

One (informal) interpretation of this Lemma suggests that in solving P(2,1), it is more important to find a good angle than it is to find a good shift. Since each iteration of our algorithm **MGL2** for P(2,1) considers *all* possible angles, it seems plausible that a single iteration of Step 1 independently guarantees a good answer. This is, in fact, the case.

THEOREM 2.4. *Let $C$ be the cost of an optimal solution to P(2,1). There is an algorithm that runs in $O(tn \log tn)$ time that finds an answer with cost no bigger than $2C$*

Notice that, in a sense, most of the useful work of **MGL2** is already done during the first iteration! Later iterations can only yield small improvements, if any. Notice also that the center of rotation for the approximation need not be $p_1$ (or any other point of $\mathcal{P}$ for that matter). We could use any other convenient pivot, such as the origin.

# 3. THREE-DIMENSIONAL CASE

In this section we describe exact and approximate solutions to P(3,1) and P(3,2).

## 3.1 Graduated lines

The graduated line $\ell = \ell(\theta, \varphi, u, v, w)$ is the line obtained from the $x$-axis by composing a rotation by angle $\pi - \varphi$, $\varphi \in [-\pi/2, \pi/2]$ around the $y$-axis, a rotation by angle $\theta$, $\theta \in [0, 2\pi]$ around the $z$-axis and a translation by vector $(u, v, w)$. The point $(u, v, w)$ is called the origin of the graduated line $\ell$. Each number $z \in Z$ defines a cell $c_z = [l_z, r_z)$ in $\ell$ that is obtained from cell $[z, z+1)$ of the $x$-axis by the composite transformation described above.

A region of space consisting of all points that project to the same cell is called a *bucket*. A bucket is bounded by two planes orthogonal to $\ell$ through consecutive cell boundaries.

REMARK 3.1. *Let $\ell$ be a graduated line with unit normal $\vec{n}$ and unit direction vector $\vec{d}$. Let $\ell'$ be the graduated line obtained by translating $\ell$ by $z\vec{d} + r\vec{n}$, for any $z \in Z$ and $r \in R$. Then lines $\ell$ and $\ell'$ determine the same bucket sets.*

A solution to an instance $\mathcal{P}$ of P(3,1) is a graduated line that minimizes the maximum number of points from $\mathcal{P}$ that lie in the same bucket.

DEFINITION 3.1. *Two graduated lines whose bucket sets produce exactly the same partition of a point set $\mathcal{P}$ are said to be equivalent with respect to $\mathcal{P}$.*

LEMMA 3.1. *Let $\ell$ be a solution to an instance $\mathcal{P}$ of P(3,1). There exists a graduated line $\ell_0$ equivalent to $\ell$ with a point of $\mathcal{P}$ lying on a boundary plane of a bucket of $\ell_0$.*

We describe an algorithm, which we call **MGL3**, to solve P(3,1). The algorithm is based, as algorithm **MGL2**, on two key ideas that allow us to discretize the problem:
1. As a consequence of Lemma 3.1, we only need to consider graduated lines that have a bucket boundary passing through a point $p_i$ of $\mathcal{P}$. Furthermore, from Remark 3.1 and without loss of generality, we can assume that $p_i$ is the origin of the graduated line. By a simple translation of $\mathcal{P}$ we can also suppose that $p_i$ is the origin $o$.
2. Each graduated line with origin $p_i = o$ determines a partition of $\mathcal{P}$ into buckets and the set of all such graduated lines produces a finite number of different partitions of $\mathcal{P}$. The set of all graduated lines is obtained from the $x$-axis by composing a rotation by angle $\pi - \varphi$ around the $y$-axis and a rotation by angle $\theta$ around the $z$-axis. During such a motion of the $x$-axis, the current partition changes only at angles $\theta, \varphi$ where the left boundary plane of a bucket passes through a point $p_j$ of $\mathcal{P}$, $p_j \neq p_i$. At this instant, $p_j$ changes from one bucket to the adjacent one. Consequently, in searching for the optimal solution for a fixed origin, it suffices to consider the finite collection of curves $\mathcal{C}$ of the 2-dimensional space determined by the angles $\theta \in [0, 2\pi]$ and $\varphi \in [-\pi/2, \pi/2]$ for which such event occurs, since in any cell of the arrangement determined by $\mathcal{C}$ the partition of $\mathcal{P}$ remains constant.

The algorithm proceeds in two steps:
1. For each point $p_i$ of $\mathcal{P}$ find the graduated line $\ell_i$ that minimizes the maximum number of points of $\mathcal{P}$ that fall in the same bucket. Call this minimum number $m_i$.
2. After all points of $\mathcal{P}$ have been processed, report the line $\ell$ that corresponds to $\min\{m_1, \ldots, m_n\}$.

The complexity of the algorithm is dominated by the construction of the arrangement determined by $\mathcal{C}$. For $m$ curves, each pair of which intersects at most twice, this can be done in time $O(m\lambda_4(m))$, where $\lambda_4(m)$ is roughly linear in $m$ [8]. This results in the following.

THEOREM 3.1. *An instance $\mathcal{P}$ of problem P(3,1) can be solved in $O(tn^2\lambda_4(tn))$ time and $O(2t^2n^2)$ space, where $t$ is the diameter of $\mathcal{P}$.*

## 3.2 Approximate solutions to P(3,1)

When the origin $(u, v, w)$ of the graduated line is fixed, and the free variables are the rotation angles $\theta, \varphi$, we obtain a restricted instance AP(3,1) of problem P(3,1). When the angles $\theta, \varphi$ of the projection line are fixed, but we are allowed to shift the projection line over itself, we obtain a restricted instance SP(3,1) of problem P(3,1).

THEOREM 3.2. *An instance of problem AP(3,1) can be solved in $O(tn\lambda_4(tn)))$ time and $O(2t^2n^2)$ space.*

THEOREM 3.3. *An instance of problem SP(3,1) can be solved in $O(n \log n)$ time.*

LEMMA 3.2. *Let $\ell$ be a projection with cost $C$ and let $C'$ be the cost of a projection $\ell'$ obtained by shifting $\ell$ by an arbitrary amount. Then, $\frac{1}{2} \leq \frac{C'}{C} \leq 2$.*

THEOREM 3.4. *Let $C$ be the cost of an optimal solution to P(3,1). There is an algorithm that runs in $O(tn\lambda_4(tn))$ time that finds an answer with cost no bigger than $2C$*

## 3.3 Graduated planes

The graduated plane $\pi = \pi(\alpha, \theta, \varphi, u, v, w)$ is the plane obtained from the $xy$-coordinate plane by composing a rotation by angle $\alpha \in [0, 2\pi]$ around the $z$-axis, a rotation by angle $\pi - \varphi, \varphi \in [-\pi/2, \pi/2]$ around the $y$-axis, a rotation by angle $\theta, \theta \in [0, 2\pi]$ around the $z$-axis and a translation by vector $(u, v, w)$. The point $(u, v, w)$ is called the origin of the graduated plane $\pi$. The graduated lines obtained by transforming the $x$ and $y$-coordinate lines with the composite transformation described above are called the graduated axes of $\pi$. Each couple of numbers $z_1, z_2 \in \mathbb{Z}$ define a cell $c_{z_1,z_2}$ in $\pi$ obtained from the cell $[z_1, z_1 + 1) \times [z_2, z_2 + 1)$ of the $xy$-plane, by the the composite transformation described above.

A region of space consisting of all points that project to the same cell is called a *bucket*. A bucket is bounded by four planes orthogonal to $\pi$ that pass through the four boundaries of a cell.

REMARK 3.2. *Let $\pi$ be a graduated plane with unit normal $\vec{n}$ and let $\vec{d_x}$ and $\vec{d_y}$ be unit direction vectors of the graduated axes of $\pi$. Let $\pi'$ be the graduated plane obtained by translating $\pi$ by $z_x\vec{d_x} + z_y\vec{d_y} + r\vec{n}$, for any $z_x, z_y \in Z$ and $r \in R$. Then planes $\pi$ and $\pi'$ determine the same bucket sets.*

A solution to an instance $\mathcal{P}$ of P(3,2) is a graduated plane that minimizes the maximum number of points from $\mathcal{P}$ that lie in the same bucket.

DEFINITION 3.2. *Two graduated planes whose bucket sets produce exactly the same partition of a point set $\mathcal{P}$ are said to be equivalent with respect to $\mathcal{P}$.*

LEMMA 3.3. *Let $\pi$ be a solution to an instance $\mathcal{P}$ of P(3,2). There exist a graduated plane $\pi_0$ equivalent to $\pi$ such that a point of $\mathcal{P}$ lies on a bottom boundary plane of a bucket of $\pi_0$.*

Next we sketch an algorithm **MGPG** to solve P(3,2). The problem can be discretized according two key ideas:

1. As a consequence of Lemma 3.3 it suffices to determine graduated planes that have a bottom bucket boundary passing through a point $p_i$ of $\mathcal{P}$. Furthermore, from Remark 3.2, we can assume that $p_i$ belongs to the bottom side of the transformed cell $[0, 1) \times [0, 1)$ of the $xy$-plane. To fulfill this condition, we can translate $\mathcal{P}$ so that $p_i$ is the origin $o$ and consider only graduated planes with origin $(-u, 0, 0)$, $u \in [0, 1)$.

2. The set of graduated planes passing through $p_i = 0$ and whose origin is $(-u, 0, 0)$, $u \in [0, 1)$ is obtained from the $xy$-coordinate plane by composing a a rotation by angle $\alpha$ around the $z$-axis, a rotation by angle $\pi - \varphi$ around the $y$-axis, a rotation by angle $\theta$ around the $z$-axis and a translation by vector $(-u, 0, 0)$. Each of these graduated planes determines a partition of $\mathcal{P}$ into buckets and the set of all such graduated planes produces a finite number of different partitions of $\mathcal{P}$. During the motion of the $xy$-coordinate plane, the current partition changes only at angles $\alpha, \theta, \varphi$ and values $u$ where the left boundary plane passes through a point $p_j$ of $\mathcal{P}$ or the bottom boundary plane passes through a point $p_k$ of $\mathcal{P}$, $p_j, p_k \neq p_i$. At these instants, $p_j$ or $p_k$ change from one bucket to the adjacent one. Consequently, in searching for the optimal solution for a point $p_i$, it suffices to consider the finite collection of hyper-surfaces $\mathcal{H}$ of the 4-dimensional space determined by the angles $\alpha \in [0, \pi]$, $\theta \in [0, 2\pi]$, $\varphi \in [-\pi/2, \pi/2]$ and the value $u \in [0, 1)$ for which such events occur, since in any cell of the arrangement determined by $\mathcal{H}$ the partition of $\mathcal{P}$ remains constant.

Algorithm **MGPG**, similarly to the previous ones, proceeds in two steps:

1. For each $p_i \in \mathcal{P}$ find the graduated plane $\pi_i$ that minimizes the maximum number of points of $\mathcal{P}$ that fall in the same bucket. Call this minimum $m_i$.

2. Report the plane $\pi$ that corresponds to $\min\{m_i \mid 1 \leq i \leq n\}$.

THEOREM 3.5. *An instance $\mathcal{P}$ of problem P(3,2) can be solved in $O(n(tn)^{5+\epsilon})$ expected time and $O((tn)^4\lambda_q(tn))$ space, where $t$ is the diameter of $\mathcal{P}$ and $q$ is a constant depending on the maximum degree of the hyper-surfaces in $\mathcal{H}$.*

## 3.4 Approximate solutions to P(3,2)

Since algorithm **MGPG** is based on theoretical results of difficult implementation and moreover has high complexity it seems logical to ask for an approximate solution to P(3,2) problem. We have extended the results concerning approximate solutions to P(2,1) and to P(3,1) to similar results for approximate solutions to $P(3, 2)$.

THEOREM 3.6. *Let $C$ be the cost of an optimal solution to P(3,2). There is an algorithm that runs in expected time $O((tn)^{3+\epsilon})$ that finds an answer with cost no bigger than $4C$.*

# 4. OTHER OPTIMIZATION PROBLEMS

The algorithms described in this paper are based on identifying classes of equivalent graduated lines or planes that define bucket sets that give exactly the same partition of the input $\mathcal{P}$. Other optimization problems related to projections of a point set onto a graduated line or plane and depending on the number of points projected in each cell can be solved using basically the same approach. Examples include finding an orthogonal projection that maximizes the number of cells with at least one projected point of $\mathcal{P}$ or one that minimizes the total number of collisions, i.e., the number of pairs of points projected to the same cell.

## 5. ACKNOWLEDGMENTS

The authors would like to thank G. Toussaint and A. Mesa for organizing the workshop at which this research was initiated. The first author was supported in part by grant BFM2000-1052-C02-01. The second author was supported in part by grants MEC-DGES-SEUID PB98-0933, SGR1999-0356 and SGR2001-0224. The third author was supported in part by the National Science Foundation under grant DMS-0107628. The fourth author was supported in part by grants MEC-DGES-SEUID PB98-0933 and TIC2001-2392-C03-01.

## 6. REFERENCES

[1] P. Agarwal, B. Bhattacharya, and S. Sen. Improved Algorithms for Uniform Partititons of Points. *Algorithmica*, 32(4):521-539, 2002.

[2] T. Asano and T. Tokuyama. Algorithms for projecting points to give the most uniform distribution with applications to hashing. *Algorithmica*, 9:572-590, 1993.

[3] P. Bose, M. F. Gómez, P. Ramos, and G. Toussaint. Drawing nice projections of objects in space. *Journal of Visual Communication and Image Representation*, 10(2):155-172, 1999.

[4] P. Bose, A. Maheshwari, P. Morin, and J. Morrison. The grid placement problem. In *Proc. 17th European Workshop on Computational Geometry*, pages 74-77, 2001.

[5] F. Gómez, F. Hurtado, J. A. Sellarès, and G. Toussaint. Nice perspective projections. *Journal of Visual Communication and Image Representation*, 12(4):387-400, 2001.

[6] F. Gómez, F. Hurtado, J. A. Sellarès, and G. Toussaint. On degeneracies removable by perspective projection. *International Journal of Mathematical Algorithms*, 2:227-248, 2001.

[7] F. Gómez, S. Ramaswami, and G. Toussaint. On removing non-degeneracy assumptions in computational geometry. *Proc. of the 3rd Italian Conference on Algorithms and Complexity*, pages 74-77, 1997.

[8] M. Sharir, and P.K. Agarwal. *Davenport-Schinzel sequences and their geometric aplications.* Cambridge University Press, 1995.

# Efficient Construction of the Union of Geometric Objects*

Eti Ezra Dan Halperin Micha Sharir

School of Computer Science
Tel Aviv University
{estere,danha,michas}@post.tau.ac.il

## ABSTRACT

We present a new incremental algorithm for constructing the union of $n$ triangles in the plane. In our experiments, the new algorithm, which we call the Disjoint-Cover (DC) algorithm, performs significantly better than the standard randomized incremental construction (RIC) of the union. Our algorithm is rather hard to analyze rigorously, but we provide an initial such analysis, which yields an upper bound on its performance that is expressed in terms of the expected cost of the RIC algorithm. Our approach and analysis generalize verbatim to the construction of the union of other objects in the plane, and, with slight modifications, to three dimensions. We present experiments with a software implementation of our algorithm using the CGAL library of geometric algorithms.

## 1. INTRODUCTION

Computing the union of $n$ triangles in the plane is a fundamental problem in computational geometry with many applications. For example, this problem arises in the construction of the forbidden portions of the configuration space in certain robot motion planning problems, and in hidden surface removal for visibility problems in three dimensions.

Computing the union, by constructing the arrangement of the triangles, may result in a solution which is too slow in practice. This is because it is likely that most vertices of the arrangement lie in the interior of the union, so computing them is wasteful. Naturally, one would like to have an output-sensitive algorithm. However, such an algorithm is unlikely to exist: Even the problem of deciding whether the union of a given set of triangles in the plane covers another given triangle is a *3SUM-hard* problem [4]. The best known solutions for problems from this family require $\Theta(n^2)$ time in the worst case, even though the size of the output may be only linear or even constant.

*Work reported in this paper has been supported in part 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), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University. Micha Sharir has also been supported by NSF Grants CCR-97-32101 and CCR-00-98246, and by a grant from the U.S.-Israeli Binational Science Foundation.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

### 1.1 Randomized Incremental Construction

We compare our new algorithm to a randomized incremental algorithm (RIC) for constructing the union, which is *quasi output sensitive*, and which is a variant of a similar algorithm due to Mulmuley [9] (a similar algorithm is also presented by Agarwal and Har-peled [7]).

Given a set $T$ of $n$ triangles in the plane, the RIC algorithm computes their union as follows: We compute a random permutation $D := (\Delta_1, \ldots, \Delta_n)$ of $T$, and insert the triangles one at a time, in their order in $D$. In the $i$'th iteration, we compute the partial union $\bigcup_{j=1}^{i} \Delta_j$. This is accomplished by finding the intersection points of the boundary of the next triangle $\Delta_i$ with the boundary of the preceding union $\bigcup_{j<i} \Delta_j$, and by removing all features of the union that lie inside $\Delta_i$. For further details concerning possible implementations of these insertion steps, see [1, 9]. In our study we ignore these details because the measure we use for the cost of the algorithms is the number of vertices that they generate (some of which may not appear on the boundary of the union), and the set of these vertices depends only on the random permutation $D$. Note however that the actual expected cost of the algorithm (in the unit cost model) depends on this number in a more subtle way than we use here [1, 9]. The justification to this approach comes from our experimental observations, and is discussed in more details below.

The crucial parameter is thus the (expected) number of intersections between triangle boundaries created during this process. Define the *depth* $d(v)$ of a vertex $v$ to be the number of triangles of $T$ that contain $v$ in their interior. Vertices at depth 0 are the vertices of the union, and they have to be constructed by any algorithm that computes the union. We are thus only interested in the *residual cost* of the algorithm, defined as the (expected) number of positive-depth vertices that the algorithm constructs.

Let $\mathcal{A}(T)$ denote the arrangement of $T$, and let $L_i$ denote the number of vertices of $\mathcal{A}(T)$ that are intersections of

triangle boundaries and have depth $i$, for $1 \leq i \leq n-2$ (the vertices of the triangles themselves are ignored in the analysis). Then the expected number of vertices at positive depth constructed by the RIC algorithm is $\theta(\mathcal{A}(T)) = \sum_{i=1}^{n-2} \frac{2}{(i+1)(i+2)} \cdot L_i$; we refer to this sum as *Mulmuley's theta series*. The factor $\frac{2}{(i+1)(i+2)}$ expresses the probability that a vertex $v$ having depth $i$ will be constructed; indeed, $v$ is constructed if and only if the two triangles that create it appear in $D$ before the $i$ triangles that cover it.

## 1.2 Related Work

Agarwal and Har-Peled gave a randomized incremental algorithm for constructing the union of $n$ triangles in the plane based on Mulmuley's theta series [1]. If the given triangles are *fat* (every angle of each triangle is at least some constant positive angle), or arise in the union of Minkowski sums of a fixed convex polygon with a set of pairwise disjoint convex polygons, then their union has only linear or near-linear complexity [6, 7], and more efficient algorithms, based on either deterministic divide-and-conquer, or on randomized incremental construction, can be devised, and are presented in the above-cited papers.

## 1.3 Our Results

We present an incremental algorithm for constructing the boundary of the union. The algorithm, which we call the *Disjoint Cover* (DC) algorithm, inserts the triangles one by one in some order. Each insertion is performed exactly as in the RIC algorithm. The difference is in the order in which we process the triangles. The intuition behind our approach is that the random order used in the RIC construction makes sure that deep vertices of the arrangement are very unlikely to be constructed; however, shallow vertices have rather high probability of being created. A typical bad situation is when there exist triangles that cover many shallow vertices. If we could force these triangles to be inserted first, they would have eliminated many vertices that will be constructed under a random insertion order. This is exactly what the new algorithm is trying to achieve.

In Section 2 we present our algorithm and state a theoretical upper bound that expresses the residual cost of our algorithm in terms of the residual cost of the RIC. Section 3 describes experimental results that compare the performance of our algorithm and of the RIC, showing that our algorithm performs significantly better in practice.

# 2. CONSTRUCTING THE UNION: DISJOINT COVER ALGORITHM

Define the *weight* $w(v)$ of a vertex $v$ (at positive depth) to be $\frac{1}{d(v)}$. We denote by $V$ the set of vertices of the arrangement $\mathcal{A}(T)$ at positive depth (considering, as above, only intersection points of the triangle boundaries). Suppose that the insertion order of the DC algorithm (to be described shortly) is $(\Delta_1, \ldots, \Delta_n)$. Define $S_{\Delta_j} = V \cap \left(\Delta_j \setminus \bigcup_{i<j} \Delta_i\right)$, namely, the set of vertices in the interior of $\Delta_j$ that are not covered by the interior of any previously inserted triangle. The *weight* $W(\Delta_j)$, for $j = 1, \ldots, n$, is then defined to be the sum of the weights of the vertices in $S_{\Delta_j}$. Note that $\{S_\Delta\}_\Delta$ is a partition of $V$ into pairwise disjoint sets.

The DC algorithm chooses an insertion order that aims to maximize the sequence $(W(\Delta_1), \ldots, W(\Delta_n))$ in lexicographical order. In an ideal setting (which is too expensive to implement, and which will therefore be modified later), we proceed as follows. Suppose we have already chosen $(\Delta_1, \ldots, \Delta_j)$ to be inserted. For each remaining triangle $\Delta$, we set (temporarily) $S_\Delta$ to be the set of all vertices of $V$ in the interior of $\Delta$ that are not covered by $\bigcup_{i \leq j} \Delta_i$. We compute the corresponding weights $W(\Delta)$ of all the remaining $\Delta$'s, and set $\Delta_{j+1}$ to be the triangle with the maximum weight. We proceed in this manner until all triangles have been chosen.

The problem with this approach is that it requires knowledge of all the vertices of $\mathcal{A}(T)$, which is too expensive to compute. Instead, we consider a smaller subset $R$. We fix some parameter $r$, select $r$ random pairs of triangles from $T$, construct and collect the intersection points, if any, of the boundaries of each pair. We now estimate each set $S_\Delta$ by the corresponding set $S_\Delta \cap R$, which is computed using only the vertices in $R$, and consequently estimate $W(\Delta)$ by the sum of weights of vertices in $S_\Delta \cap R$. At present, this simplification should be viewed as purely heuristic—the theory of random sampling and $\varepsilon$-approximations (see, e.g., [10]) is not directly applicable to argue that $S_\Delta \cap R$ is a good approximation of $S_\Delta$, because the portion of the plane over which $S_\Delta$ is estimated at the $j$-th step, namely, $\Delta \setminus \bigcup_{i<j} \Delta_i$, may not have constant complexity, which is a (sufficient) condition that is usually needed to be assumed in order to facilitate the application of the random sampling theory. We hope and plan to set this heuristic on solid theoretical footing. Nevertheless, our experimental results indicate that this heuristic performs very well in practice—see Section 3.

The above discussion is summarized in the following lemma. (Proofs are omitted in this abstract.)

LEMMA 1. *Given a set $T$ of $n$ triangles and a vertex set $R$ as above, the construction of the insertion order by the DC algorithm takes $O(n|R| \log n)$ time.*

The following theorem relates the residual cost of the DC algorithm (in its ideal setting) to that of the RIC algorithm:

THEOREM 1. *Let $T$ be a collection of $n$ triangles with $\kappa$ intersection points at positive depth. Then the number of positive-depth vertices generated by the ideal DC algorithm is at most $O(n^{2/3}\kappa^{1/3}M^{1/3})$, where $M$ is the expected number of positive-depth vertices generated by the RIC algorithm.*

Note that when $M$ and $\kappa$ are $\Theta(n^2)$, both algorithms generate the same number of vertices asymptotically. If either of these two parameters is strictly subquadratic, then the DC algorithm will produce a strictly subquadratic number of vertices at positive depth. However, this bound seems rather pessimistic, and we believe that it can be improved (as is strongly suggested by our experimental results).

We remark though that there exist (rather pathological) examples in which $\kappa \ll n^2$ and the DC algorithm performs

| input name | description | figure |
|---|---|---|
| regular | arbitrary triangles randomly placed inside a square | Figure 1 |
| fat | equilateral triangles randomly placed inside a square | Figure 1 |
| fat_with_grid | a grid-like pattern partially covered by many random fat triangles; half of the triangles form the grid and the other half are the fat triangles | Figure 2 |

**Table 1: The different data sets**

considerably worse than the RIC algorithm. No such examples are known for $\kappa = \Theta(n^2)$, and we conjecture that in this case the residual cost of the DC algorithm is at worst comparable with that of the RIC.

# 3. EXPERIMENTAL RESULTS

In this section we present experimental results comparing the RIC and the DC algorithms. We start by describing the input sets and the implementation and then display the results and comment on them.

The motivation to devise the DC algorithm is practical. We wish to precede the incremental construction of the union with a simple and fast procedure that will speed up the more heavy-duty incremental stage. The incremental stage uses rather involved data structures for representing the topology of the partially constructed union and for searching in it. In comparison, the preprocessing stage of the DC algorithm, where we compute the order of insertion, uses very simple operations. The most expensive operation, since we use exact arithmetic (see below), is the construction of a vertex, which we do $|R|$ time. Another operation is testing whether a vertex lies inside a triangle, which we do $n|R|$ times. What is the best size of $R$ in practice is the subject of on-going investigation, which has to determine the optimal trade-off between the preprocessing cost and the degree of approximation of the true triangle weights (which may affect the insertion order). In our experiments the preprocessing time is negligible compared with the time of the incremental construction, even when $|R|$ is linear in the number of input triangles.

## 3.1 Input Sets

The input data that we used is described in Table 1 and in Figures 1 and 2. For each type of input, we have experimented with a varying number of triangles, up to 800 per run. The complexity of the union of *fat* is almost linear in the number of triangles (see Figure 1), while the union of *fat_with_grid* can have a superlinear number of holes. In addition to the three types of input presented here, we performed other tests on different data sets which are not reported in this abstract. In all our experiments the DC algorithm performs better than the RIC.

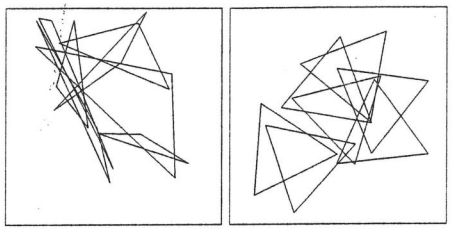


**Figure 1: Regular input (left) and fat input**

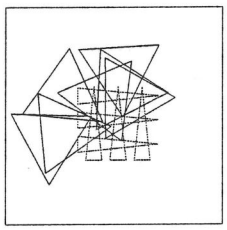


**Figure 2: Fat_with_grid input**

## 3.2 Implementation Details

Our implementation of the union algorithms is based on the CGAL (version 2.3) and LEDA (version 4.3) libraries. The implementation employs the CGAL maps and arrangements packages [3, 5], and uses *exact arithmetic*. We use LEDA's rational kernel which employs a floating point filter in computing predicates [8]. Note however that the construction of new vertices does not benefit from filters and the coordinates of intersection points are computed to unlimited precision (namely with as much precision as required).

More specifically, the DC and RIC algorithms use the CGAL Planar_map_with_intersections class. The union constructed by each of the two algorithms is stored in a Doubly Connected Edge List (DCEL for short) [2]. Every insertion of a triangle to the partial union constructed is performed by the insertion of the three edges defining the triangle, one at a time. Each insertion of an edge $e$ to the DCEL is done in the following manner: first we locate one of the endpoints of $e$ in the DCEL. The point-location operation is performed by "walking" backwards along the zone of a vertical ray emanating from the query point. The walk starts at the unbounded face and progresses towards the query point as described in [3]. Next, we traverse the zone of $e$. Each time we discover an intersection along $e$ we create a new vertex in the DCEL and insert into the structure the portions of $e$ that do not lie in the present union (each such portion is delimited by a pair of consecutive intersection points).

In the RIC algorithm, we first randomly permute all the triangles in the input set and then construct the union boundary incrementally by adding one triangle at a time, and removing all features that lie inside the union, and have not yet been removed. The removal stage is performed by traversing all edges in the current structure, and checking whether each such edge lies inside the union (in constant time per edge, due to extra information that should be maintained in the edges). In practice, the time consumed by traversing all edges is negligible.

The preprocessing stage of the DC algorithm that produces the ordering of the triangles uses a random subset of the vertices of the underlying arrangement of the triangles. For the experiments reported below we implemented this stage slightly differently from what is described in Section 2. We fix a number $r$ and we choose pairs of triangles at random till we get $r$ vertices. We make sure to stop after $x$ trials if the arrangement has only $x$ intersection vertices. After obtaining the insertion order, the algorithm is implemented in the same way as the RIC algorithm.

**Remark.** As mentioned above, our decision to focus on the number of positive-depth vertices created during the algorithm as the measure of algorithm performance comes from experimental observations. We split the operations in our implementation into two types: (i) *search* and (ii) *construction*. Type (i) operations include the point location in the current map (describing the union constructed so far) of the endpoint of a new triangle edge $e$ that we wish to insert, as well as traversal of the zone of $e$ as we insert $e$ into the map. Type (ii) operations consist of computation of the intersection points of triangle boundaries (i.e., new vertices) and updating the DCEL. Our experiments show that type (ii) operations consume the larger portion of the running time by far. The explanation of this phenomenon is involved. One obvious reason for the dominance of type (ii) operations is the computing of intersection points which we carry out to arbitrary precision. This phenomenon can also be attributed to the specific way in which we implement the DCEL. Our implementation is not theoretically optimal (as compared, e.g., with [1, 9]) but has various practical advantages. For example, it is known that during the construction of a planar map it is more effective to use fairly naive point location algorithms (such as the walk along the zone algorithm mentioned above) over more intricate logarithmic search-time structures; see [3].

## 3.3 Results

We present experimental results of applying both algorithms to each of the data sets described in Section 3.1. We present the number of positive-depth vertices created by each of these algorithms, which, as discussed above, is the yardstick we use for measuring and comparing the performance of the algorithms.

As mentioned above, determining the right size of $R$ in practice is a subject of on-going investigation. For each input type we show five graphs. Besides the graph for the RIC algorithm we present graphs for the DC algorithm where the size of $R$ varies between a constant, a logarithmic factor in the number of input triangles, a linear factor, and $R$ being the full set $V$. The results are presented in Figures 3 through 5.

In all the graphs we see that if we take the whole set $V$ into account in computing the insertion order then the savings in the union construction stage are big. In general, in all our experiments, the DC algorithm performs better than the RIC,[1] and the performance improves as the size of $R$ increases. In some cases, e.g., in the case of the *fat_with_grid* input, even if we use much smaller samples $R$, for example, samples of size linear in the input size, then we save the construction of over 1500 vertices (out of about 2000) during the incremental stage when the input consists of 190 triangles (Figure 5). The saving hardly changes when we use $R = V$.

[1]Recall that this may fail to hold in some pathological examples, when $|V| \ll n^2$.

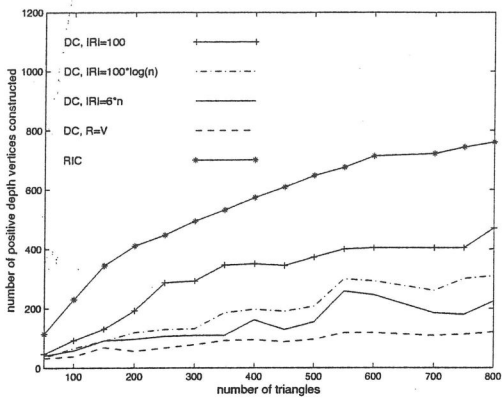



**Figure 3: Number of positive-depth vertices created for the *regular* input**

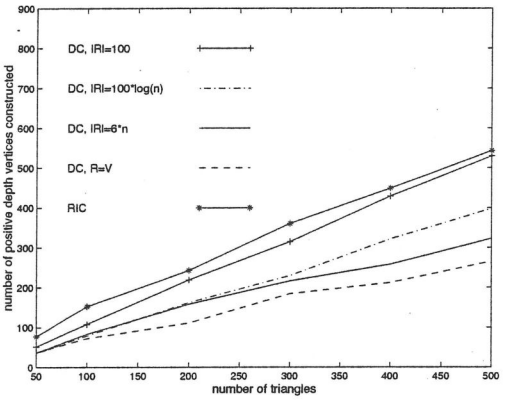



**Figure 4: Number of positive-depth vertices created for the *fat* input**

For the *fat* input (see Figure 4), the improvement is less significant than the improvement obtained for the other input sets. Notice that the amount of work the RIC algorithm performs for fat triangles is always close to linear.[2] Hence improving such small amount of work is more difficult than improving that amount for regular triangles.[3]

For the *fat_with_grid* input (Figure 5) the RIC algorithm performs poorly since the intersection points of the grid are shallow on the average.

We remark that the number of vertices $|V|$ in some of our examples is huge (reaching roughly half a million for the

[2]The proof is given in the full paper.

[3]Note that we do not know the effect of the randomization in the choice of the regular input on the number of positive-depth vertices constructed by one algorithm or by the other.

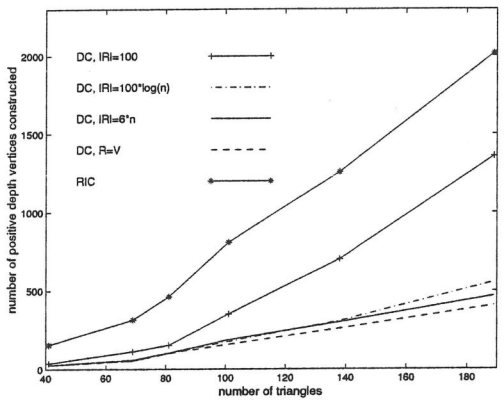



**Figure 5: Number of positive-depth vertices constructed for the *fat_with_grid* input**

*regular* input with 800 triangles), rendering the construction of the union by first computing the underlying arrangement unacceptable (when using exact arithmetic).

## 3.4 Conclusions

The experiments reported above demonstrate the practical advantages of the DC algorithm relative to the RIC algorithm. Our results show ~~that~~ the number of positive-depth vertices constructed by the RIC algorithm is larger (and, in many cases, significantly larger) than the number of such vertices constructed by the DC algorithm, for all three kinds of input. We also performed similar tests on different data sets, and got similar results.

We note that the DC algorithm in its ideal setting (with $R = V$) is *deterministic*. In practice we use randomness, but only to estimate the weights of triangles. After doing so, the insertion order is still computed deterministically.

In another batch of tests, we defined the *weight* of each vertex simply as 1, instead of $\frac{1}{d(v)}$ (see Section 2). In that case, although the number of positive-depth vertices constructed by the DC algorithm is still smaller than that constructed by the RIC algorithm, there are some data sets for which this ratio is less significant than the ratio obtained in our experiments presented in Section 3.3. For instance, when taking $R = V$ and the *regular* input sets with 700 triangles, the ratio is roughly 2 : 7 for this alternative weighing scheme, compared with 2 : 13 when employing the weight scheme defined in Section 2. On the other hand, for the *fat_with_grid* input sets, we observed no significant difference between the two approaches. This is because the DC algorithm that uses the original weight scheme gives high priority to shallow vertices, and hence tends to cover them first (with high probability). The deep vertices are likely not to cause any problem, in either scheme, since they are likely to be covered by some triangle before they need to be constructed. This explains the different behavior of the two weight schemes on the *regular* input sets. However, when most of the vertices are shallow, as in the *fat_with_grid* input sets, giving higher priority to shallow vertices has less significant effect, and hence there is no substantial difference between the two weighting schemes.

# 4. REFERENCES

[1] P. K. Agarwal and S. Har-Peled. Two randomized incremental algorithms for planar arrangements, with a twist. Manuscript, 2001.

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

[3] E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *The ACM Journal of Experimental Algorithmics*, 5, 2000. Also in LNCS Vol. 1668 (WAE '99), Springer, pp. 154–168.

[4] 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.

[5] I. Hanniel and D. Halperin. Two-dimensional arrangements in CGAL and adaptive point location for parametric curves. In *Proc. of the 4th Workshop of Algorithm Engineering*, volume 1982 of *Lecture Notes Comput. Sci.*, pages 171–182. Springer-Verlag, 2000.

[6] 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.

[7] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 49–58, 1991.

[8] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[9] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey 07632, U.S.A., 1994.

[10] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. John Wiley & Sons, New York, NY, 1995.

# Exact $L_\infty$ Nearest Neighbor Search in High Dimensions

Laura Heinrich-Litan *
Institut für Informatik
Freie Universität Berlin
D-14195 Berlin, Germany
litan@inf.fu-berlin.de

## 1. INTRODUCTION

The *nearest-neighbor problem* is the problem of designing an efficient data structure storing a set $P$ of $n$ points in $\mathbb{R}^d$ for answering *nearest neighbor queries*. In a nearest neighbor query some point $q \in \mathbb{R}^d$ is specified and the (some) point in $P$ closest to $q$ has to be determined. This apparently very natural geometric problem has numerous applications in statistics and data analysis, information retrieval, data compression, pattern recognition, and multimedia databases.

Various data structures for nearest neighbors have been developed in computational geometry, early ones by Dobkin and Lipton [4] and by Clarkson [3] based on higher-dimensional Voronoi-diagrams. Although these data structures have query times logarithmic in $n$, it was assumed that the dimension $d$ is a (low) constant and, in fact, the preprocessing time, the storage, or the query time are exponential in $d$. Only a structure by Meiser [9], more generally for point location in an arrangement of hyperplanes, has query time $O(d^5 \log n)$ but storage $O(n^{d+\varepsilon})$.

In many applications, however, the dimension $d$ of the search space is quite high and can reach several hundreds or even several thousands. Therefore, running times and storage requirements exponential in $d$ are prohibitive in these cases. All query algorithms are in fact competing with the *brute-force* method of just determining the distance of $q$ to each point in $P$ and selecting the minimum. This method obviously requires no preprocessing, only the set $P$ itself needs to be stored, and the query time is $O(nd)$ for all $L_p$-distances, $1 \leq p \leq \infty$.

*Partially supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg "Algorithmische Diskrete Mathematik", grant GRK 219/3 and by IST Programme of EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

In order to overcome the "curse of dimensionality" researchers started to look for *approximate* solutions for the nearest-neighbor-problem. This was investigated intensely in the last years and a real break-through was achieved by working with *randomized* techniques [7], [8], [6], [5].

In practice even for dimensions in the range of some tens the brute-force method is used rather than the known, sophisticated data structures. In this paper we present an algorithm for solving the nearest-neighbor problem with respect to the $L_\infty$-distance. It has the advantage that it solves the problem *exactly* and, furthermore, is very simple and easy to implement. Its average runtime assuming that the set $P$ is drawn randomly from the unit cube $[0,1]^d$ under uniform distribution is essentially $\Theta\left(n \ln(\frac{d}{\ln n}) + n\right)$, thereby improving the brute-force method by a factor of $\Theta(\frac{\ln(d/\ln n)}{d})$. The constant in the $\Theta$-term is close to one, and for many applications the dimension $d$ is in the range of some hundreds so the speedup factor comes close to 50, which is a considerable improvement in practice.

## 2. THE ALGORITHM

Throughout this paper we will assume that $P$ is a fixed set of $n$ points $p^i = (p_1^i, \ldots, p_d^i) \in \mathbb{R}^d$, $i = 1, \ldots, n$. Since we can translate and scale $P$ without changing the problem we will assume without loss of generality that $P \subset [0,1]^d$. For our probabilistic considerations we assume that $P$ is drawn at random from $[0,1]^d$ under uniform distribution. We concentrate on the $L_\infty$-distance.

The idea of the query algorithm is to consider for a query $q = (q_1, \ldots, q_d) \in [0,1]^d$ the cube $C_{q,\alpha} = [q_1 - \frac{\alpha}{2}, q_1 + \frac{\alpha}{2}] \times \cdots \times [q_d - \frac{\alpha}{2}, q_d + \frac{\alpha}{2}]$ around $q$ of side length $\alpha$ such that the expected number of points in $C_{q,\alpha}$ is a low number $\varphi$. The cube is searched for the points contained in it. Let $P_\alpha$ be the set containing the points of cube $C_{q,\alpha}$. If $P_\alpha \neq \emptyset$ we determine the nearest neighbor of $q$ by the brute-force method for the points of $P_\alpha$. There is a nonzero probability that the cube $C_{q,\alpha}$ contains no points of $P$ at all. In this case, the *brute-force* method will be called for all points of $P$. The parameter $\varphi$ should be determined such that the probability of this event is so small that it does not effect the total asymptotic runtime. This query algorithm is called the CUBE-METHOD. The following gives a schematic description of this method.

**CUBE_METHOD(q, P)**

$\alpha$ = DETERMINE_LENGTH($\varphi, q$);
$P_\alpha$ = SEARCH_CUBE($\alpha, q, P$);
if ( $P_\alpha \neq \emptyset$ ) $\hat{p}$ = BRUTE_FORCE($P_\alpha$);
else $\hat{p}$ = BRUTE_FORCE($P$);

Procedure DETERMINE_LENGTH determines the side length $\alpha$ of a cube $C_{q,\alpha}$ with center $q$ such that the expected number of points in $P \cap C_{q,\alpha}$ is $\varphi$ in time $O(d \log(d\varphi))$ as shown in [1]. Procedure SEARCH_CUBE determines the set $P_\alpha = P \cap C_{q,\alpha}$.

The CUBE_METHOD can be improved as follows : if $C_{q,\alpha}$ is empty the alternative to determining the nearest neighbor by brute-force is to consider a bigger cube with center $q$ and search it for points. The size of the current cube should be increased as long as it contains no points of $P$. The expected number $\varphi$ of the points in the first cube $C_{q,\alpha}$ can be chosen to be a small constant. Nevertheless, this variant called the *growing-cube method* has the same expected asymptotic complexity as the CUBE_METHOD.

The runtime of the CUBE-METHOD is essentially determined by the runtime of the SEARCH_CUBE procedure. In the following we present three searching strategies to determine the points of $P$ contained in the $C_{q,\alpha}$.

## 2.1 Searching the cube by scanning

In [1] the following searching procedure is investigated : for each point $p^i$ the algorithm scans its coordinates $p^i_j$ whether they lye in the appropriate interval $[q_j - \frac{\alpha}{2}, q_j + \frac{\alpha}{2}]$. As soon as a point $p^i \in P$ turns out not to lie in $C_{q,\alpha}$, the algorithm eliminates it from further consideration. The following gives a schematic description of this procedure called SCAN_CUBE. The set $P_\alpha$ containing the points of the cube is returned as result. $\mathcal{D}$ is a list of the dimensions $\{1, \ldots, d\}$.

**SCAN_CUBE($\alpha$, q, P, $\mathcal{D}$)**

$P_\alpha = \emptyset$;
forall $p^i \in P$
  for $j \in \mathcal{D}$ (COORD)
    if $|p^i_j - q_j| > \frac{\alpha}{2}$ then break for-loop; (TESTS)
  if ( no coordinate did fail ) then $P_\alpha = P_\alpha \cup \{p^i\}$;
return $P_\alpha$ ;

In [1] is shown that the expected number of tests of the SCAN_CUBE procedure is bounded by $\max\left\{\frac{2nd}{\ln(n/\varphi)}, 2n\right\}$ if the order in which the coordinates of the points are checked in (COORD) corresponds to the increasing order of the side lengths of the box $C_{q,\alpha} \cap [0,1]^d$. This order can be computed once at the beginning in time $O(d \log d)$. For suitable $\varphi$ the CUBE-METHOD with the SCAN_CUBE search-procedure has an expected asymptotic runtime of $O(\frac{nd}{\ln n} + n + d \ln d)$ and it requires $\Theta(nd)$ storage and no preprocessing.

The drawback of the SCAN_CUBE procedure is that the side lengths of the box $C_{q,\alpha} \cap [0,1]^d$ are very large (larger than 0.9) for the case of high dimension $d$ and reasonable values $n$, and this implies a large number of points which *do not* fail the test in step labeled by (TESTS) for some dimension $j \in \{1, \ldots, d\}$. This means that in each of the dimensions the expected number of points whose coordinates are *outside* of the appropriate interval is a small number. This leads to the idea to look at the points which are not in the cube with respect to each dimension independently and reject them in order to find out the points which are in the cube[1].

## 2.2 Searching the cube by rejecting

During the *preprocessing* the coordinates of the points are sorted in each dimension $j = 1, \ldots, d$. This takes $O(nd \ln n)$ time. In dimension $j \in \{1, \ldots, d\}$ an array $T_j$ with $n$ items is stored where $T_j[i]$ contains the $i$-th largest coordinate $T_j[i].coord$ in dimension $j$ and the index $T_j[i].index = l$ of the point $p^l$ corresponding to that coordinate.

The *query algorithm* is the CUBE-METHOD where the procedure to search the cube for points works as follows: for each dimension $j \in \{1, \ldots, d\}$ those points are marked as "rejected" in a bit array whose coordinates in dimension $j$ are outside the interval $[q_j - \frac{\alpha}{2}, q_j + \frac{\alpha}{2}]$. At the end the bit array is scanned and all non-rejected points are returned. The following gives a schematic description of this search-procedure called REJECT.

**REJECT($\alpha$, q, T, P, $\mathcal{D}$)**

forall $j \in \mathcal{D}$
  $i = 1$;
  while ( $i \leq n$ and $T_j[i].coord < q_j - \frac{\alpha}{2}$ ) (TESTS)
  do mark point with index $T_j[i].index$ as rejected;
    $i := i + 1$;

  $i = n$;
  while ( $i \geq 1$ and $T_j[i].coord > q_j + \frac{\alpha}{2}$ ) (TESTS)
  do mark point with index $T_j[i].index$ as rejected;
    $i := i - 1$;

$P_\alpha = \emptyset$;
for $i = 1$ to $n$ do
  if ( $p^i$ not rejected ) then $P_\alpha = P_\alpha \cup \{p^i\}$; (BITARRAY)
return $P_\alpha$ ;

*Analysis of the expected runtime*

Let $s_j$ be the side length of $W_{q,\alpha} = C_{q,\alpha} \cap [0,1]^d$ in dimension $j$. The volume of $W_{q,\alpha}$ is $V(W_{q,\alpha}) = \prod_{j=1}^d s_j(\alpha)$ which equals $\frac{\varphi}{n}$ since $\varphi$ is the expected number of points in $W_{q,\alpha}$.

The expected total number of points which are outside of some interval $[q_j - \frac{\alpha}{2}, q_j + \frac{\alpha}{2}]$, for some $j \in \{1, \ldots, d\}$ is $n \sum_{j=1}^d (1 - s_j)$. It can be shown that $\sum_{j=1}^d (1 - s_j) \leq \ln(n/\varphi)$, thus the expected number of tests performed in steps (TESTS) can be bounded by $2d + n \ln(n/\varphi)$. Clearly, the number of tests performed in step (BITARRAY) is $n$. Thus, the total expected runtime of REJECT is $E[T_{reject}] \leq 2d + n \ln(n/\varphi) + n$.

The probability that no point of $P$ is in $C_{q,\alpha}$ is $(1 - \frac{\varphi}{n})^n$; in this case, the CUBE-METHOD calls the brute-force method which has a runtime of $\Theta(nd)$. With probability $1 - (1 - \frac{\varphi}{n})^n$ there is at least a point in $C_{q,\alpha}$ and in this case brute-force method will be called with the set $P_\alpha$ of the points in $C_{q,\alpha}$ having the expected runtime $\varphi d$. Thus, the expected

[1]The author would like to thank Piotr Indyk for this hint.

number $T_{cube}$ of comparisons of the CUBE-METHOD using the REJECT searching procedure is given by :

$$\begin{aligned} E[T_{cube}] &= E[R] + (1-\frac{\varphi}{n})^n nd + \left(1-(1-\frac{\varphi}{n})^n\right)\cdot \varphi d \\ &\quad + c\cdot d\ln(d\varphi) \\ &\leq 2d + n\ln(n/\varphi) + n + e^{-\varphi}nd + \varphi d \\ &\quad + c\cdot d\ln(d\varphi) \end{aligned}$$

where $c$ is a constant and $c\cdot d\ln(d\varphi)$ is the runtime to determine the side length $\alpha$ of the cube $C_{q,\alpha}$ and the increasing order of the side lengths of the box $C_{q,\alpha}\cap[0,1]^d$. A detailed analysis shows that a suitable $\varphi$ is $\max\{\frac{n}{d}, \ln d\}$ and in this case we get $E[T_{cube}] = O(n\ln d + d\ln n)$.

THEOREM 1. *Let $P$ be a set of $n$ points from $[0,1]^d$. The* CUBE_METHOD *with the* REJECT *search-procedure finds the nearest neighbor from $P$ to the query point $q\in[0,1]^d$ with an expected asymptotic runtime of $O(n\ln d + d\ln n)$ if the points of $P$ are independently drawn at random from $[0,1]^d$ under uniform distribution.*

## 2.3 Searching by rejecting and scanning

The same *preprocessing* is done as discussed in Section 2.2. For the *query algorithm* the idea is to use both methods presented above, each restricted to a different subset of the set $\mathcal{D}=\{1,\ldots,d\}$ of dimensions. Consider a partition $\mathcal{D}_1\cup\mathcal{D}_2$ of the set of dimensions $\mathcal{D}$. Use the REJECT-procedure with respect to the dimensions of $\mathcal{D}_1$ to find out the set $P_\alpha^{\mathcal{D}_1}$ of points which are in $C_{q,\alpha}$ with respect to $\mathcal{D}_1$. In the second phase SCAN_CUBE procedure is called with respect to the dimensions of $\mathcal{D}_2$ taking into account only the points from $P_\alpha^{\mathcal{D}_1}$. The following gives a schematic description of this search-procedure called REJECT_SCAN.

**REJECT_SCAN($\alpha$, q, P, T)**

$\{1,\ldots,d\} = \mathcal{D}_1\cup\mathcal{D}_2$; (PARTITION)
$P_\alpha^{\mathcal{D}_1}$ = REJECT($\alpha, q, T, P, \mathcal{D}_1$); (REJECT)
$P_\alpha$ = SCAN_CUBE($\alpha, q, P_\alpha^{\mathcal{D}_1}, \mathcal{D}_2$); (SCAN)
return $P_\alpha$ ;

It can be shown that :

A) if $d \geq \ln n$ then there is a suitable partition $\mathcal{D} = \mathcal{D}_1\cup\mathcal{D}_2$ which can be determined in $O(d)$ time, such that the expected number of comparisons of procedure REJECT_SCAN is

$$E[T_{rej\_scan}] \leq n\ln\left(\frac{4d}{\ln(n/\varphi)}\right) + 3n + 2d$$

and the expected runtime of CUBE_METHOD with the REJECT_SCAN is $O\left(n\ln(\frac{d}{\ln n}) + n + d\ln d\right)$,

B) if $d < \ln n$ then $\mathcal{D}_1 = \emptyset$ and $\mathcal{D}_2 = \mathcal{D}$, i.e. only the SCAN_CUBE is called. In this case, the expected number of comparisons of REJECT_SCAN is

$$E[T_{rej\_scan}] \leq 3n$$

and the expected runtime of CUBE_METHOD with the REJECT_SCAN is $O(n)$,

where a suitable $\varphi$ is $\max\{\ln\ln n, \ln d\}$.

THEOREM 2. *Let $P$ be a set of $n$ points from $[0,1]^d$. The* CUBE_METHOD *with the* REJECT_SCAN *procedure finds the nearest neighbor from $P$ to the query point $q\in[0,1]^d$*

*A) if $d \geq \ln n$ with an expected asymptotic runtime of $O\left(n\ln(\frac{d}{\ln n}) + n + d\ln d\right)$*

*B) if $d < \ln n$ with an expected asymptotic runtime of $O(n)$*

*if the points of $P$ are independently drawn at random from $[0,1]^d$ under uniform distribution. The method requires $O(nd)$ storage and $O(nd\ln n)$ preprocessing time.*

# 3. EXPERIMENTS

For the experiments the data set $P$ and also the query point $q$ were generated in $[0,1]^d$ according to the uniform distribution.

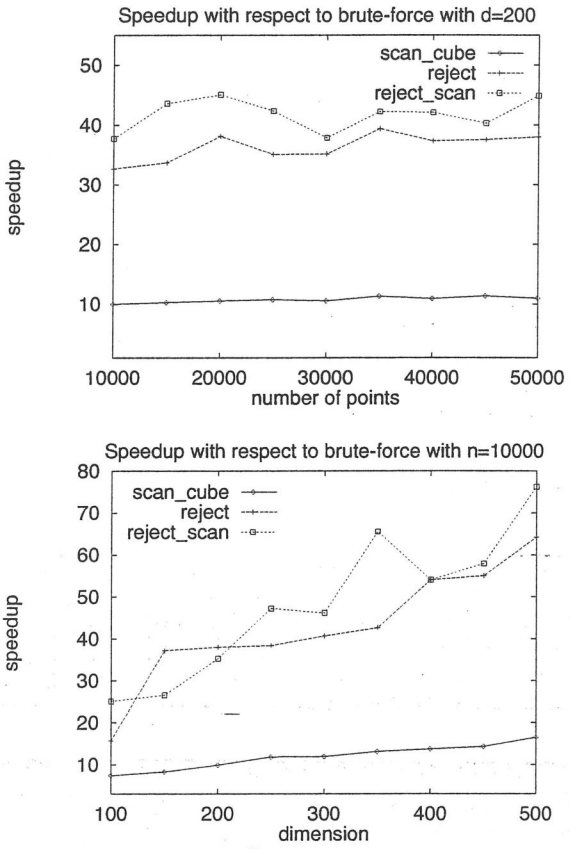



**Figure 1: Comparison of the algorithms with the brute-force method**

The first figure shows an experimental comparison of the implementations of the variants of the algorithm for dimension $d = 200$ and sizes of $P$ between 10000 and 50000. The second figure shows the experimental comparison of the methods for $n = 10000$ and dimension $d$ between 100 and 500. The value on the vertical scale is the factor by which each algorithm is faster then the brute-force method. A speedup factor in the range between 30 to 80 is a considerable improvement in practice.

## 4. ACKNOWLEDGMENTS

The author would like to thank Helmut Alt and Piotr Indyk for hints and helpful discussions.

## 5. REFERENCES

[1] H. Alt and L. Heinrich-Litan. Exact $L_\infty$ Nearest Neighbor Search in High Dimensions. In *Proc. 17th ACM Symposium on Computational Geometry*, pages 157–163, 2001.

[2] J. L. Bentley, B. W. Weide and A. C. Yao. Optimal expected time algorithms for closest point problems In ACM Trans. on Math. Software 6, pages 563–580, 1980.

[3] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17, pages 830–847, 1988.

[4] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5, pages 181–186, 1976.

[5] S. Har-Peled. *A Replacement for Voronoi Diagrams of Near Linear Size.* To appear in *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, 2001. Available from `http://www.uiuc.edu/~sariel/papers`.

[6] P. Indyk. Dimensionality reduction techniques for proximity problems. In *Proceedings, ACM Symposium on Data Structures and Algorithms, SODA 2000*, pages 371–378, 2000.

[7] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimension. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 599–608, 1997.

[8] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 614–623, 1998.

[9] S. Meiser. Point location in arrangements of hyperplanes. *Inform. Comput.*, 106, pages 286–303, 1993.

# Euclidean position in 2-orbifolds*

C. Cortés
University of Sevilla, Spain
ccortes@us.es

A. Márquez
University of Sevilla, Spain
almar@us.es

J. Valenzuela
University of Extremadura, Spain
jesusv@unex.es

## ABSTRACT

Intuitively, a set of sites on a surface is in Euclidean position, if points are so close each other that planar algorithms can be easily adapted in order to solve classical problems in Computational Geometry. In this work we formalize a definition of the term "Euclidean position" for a relevant class of metric spaces, the Euclidean 2-orbifolds, and present methods to compute when a set of sites has this property. We also show the relation between the convex hull of a point set in Euclidean position on a 2-orbifold and the planar convex hull of the counter-image (via the quotient map) of the set.

## Keywords

Euclidean position, metrically convex hull, orbifold.

## 1. INTRODUCTION

There exist many applications of Computational Geometry in which the input and/or the output data are given on a surface other than the plane. It is generally assumed in those applications that if a given set is contained on a small portion of the surface, then simple adaptations of planar algorithms (in order to obtain, for instance, the convex hull, the Voronoi diagram or a triangulation with nice properties) can be given. But we are not aware of a general framework for approaching the problem of deciding for which data planar methods are still valid. The only steps in that direction are those given in [1] defining and working with a new concept, Euclidean position, but limited to very specific surfaces as the cylinder, the torus, the cone, or the sphere. It is the aim of this work to generalize that concept to a very broad class of spaces such as that of 2-orbifolds. Intuitively, if a set is in Euclidean position, all planar methods of Computational Geometry are valid for that set.

As it is known, any 2-orbifold is obtained as a quotient space $R^2/\Gamma$, being $\Gamma$ a group of planar motions. The four locally Euclidean surfaces (the cylinder, the twisted cylinder, the torus and the Klein bottle) and others well-known surfaces as the Möebius strip or the projective plane are 2-orbifolds. A more complete study of them can be found in [2].

*Partially supported by MCyT project BFM2001-2474

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

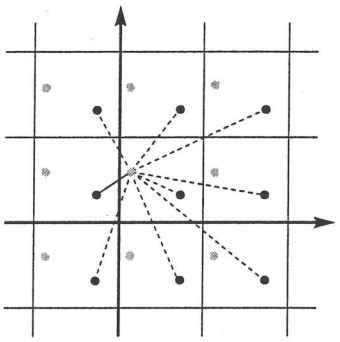


**Figure 1: Planar representation of the geodesic arcs between two points in the flat torus. The segment between them is shown in dark.**

Geodesics joining two points $P$ and $Q$ in the quotient metric correspond to straight-line segments matching one element of the orbit of $P$ with all the elements of the orbit of $Q$. The shortest of these line segments will be called the *segment* between $P$ and $Q$ (Figure 1).

Given a set of sites $P$ in the cylinder, the torus, the cone, or the sphere, we say that the set $P$ is in *Euclidean position* if

1. it is contained between opposite generatrices of the cylinder or the cone;
2. it is contained in a quadrant (the region between two opposite parallels and two opposite meridians) of the torus;
3. it is contained in a hemisphere of the sphere.

In this work we give a generalization of the concept "Euclidean position" to 2-orbifolds that will agree with the definition given for the cases of the surfaces mentioned above. In addition, that definition will be consistent with the main objective sought; and, in this way, if a set of points is in Euclidean position on a 2-orbifold, then its convex hull has exactly the same shape as the planar convex hull of the

translation of that set into the plane. Observe that the convex hull provides a good test in order to know if a given set presents a planar behavior.

## 2. EUCLIDEAN POSITION

Let $S$ be a 2-manifold given by the quotient map

$$\varphi : \mathbb{R}^2 \longrightarrow \mathbb{R}^2/\Gamma \simeq S.$$

It is known that in order to get a simple representation of these surfaces, a very useful tool is the *fundamental domain*: a closed region in the plane containing one element of each orbit, that is unique except for some points of the boundary (double points). If we delete all double points of a fundamental domain and consider its imagen by $\varphi$, we obtain what we call a *fundamental region*. In this way, we say that a set $P$ on a 2-orbifold is in *Euclidean position* if there exists a fundamental region in such a way that all segments joining points of $P$ are contained in that fundamental region.

The definition of Euclidean position is just a generalization of the same concept considered previously for the cylinder or the torus as we can see in the following theorem

THEOREM 1. *1. Let $P$ be a set in Euclidean position on the cylinder, then it is contained between opposite generatrices of the cylinder.*

*2. Let $P$ be a set in Euclidean position on the torus, then it is contained in a quadrant of the torus.*

Of course, in this context, to know whether a point set is in Euclidean position or not will be very important . For this purpose, we will make use of the *Dirichlet domains*, that is, a fundamental domain equals to the Voronoi region of a point $Q$ with respect to its orbit:

$$\{R \in \mathbb{R}^2 : d(R,Q) \leq d(R,gQ) \quad \forall g \in \Gamma\}.$$

THEOREM 2. *Given a set $P$ on a 2-orbifold $S$, $P$ is in Euclidean position if and only if there exists a plane copy of $P$ contained in the intersection of the Dirichlet domains of their elements.*

According to Theorem 2, the different methods to establish if a set of points is in Euclidean position are going to be determined by the particular shapes of the Dirichlet domains associated to each class of motions that appear in the group of the surface, and we can obtain

THEOREM 3. *Given a set $P$ of $N$ points on a 2-orbifold $S$, it is possible to determine if $P$ is in Euclidean position in*

1. *$O(N)$ time if there is not a glide reflection in the group of $S$.*
2. *$O(N \log N)$ otherwise.*

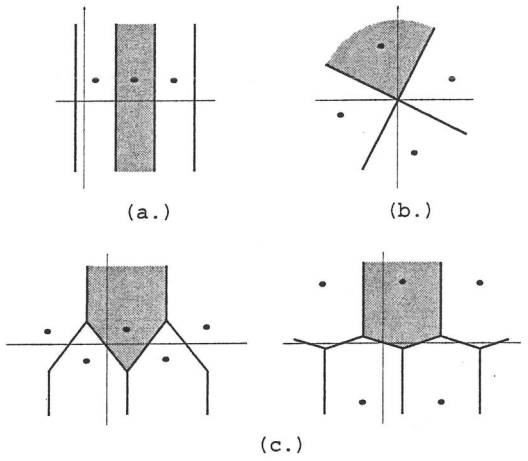



**Figure 2: The shape of the Dirichlet domains for groups generated by (a.) a traslation or (b.) a rotation, does not depend on the point configuration; however, the height of the points becomes relevant for those generated by a reflection glide, as shown in (c.).**

The different behaviour of the surfaces whose groups contain glide reflections is due to, unlike the rest of the motions, the shape of the Dirichlet domains. They strongly depend on the position of the generating points (see Figure 2).

## 3. EUCLIDEAN POSITION AND CONVEX HULL

The concept of planar convexity can be generalized to a surface $S$ by defining a set $C$ to be *metrically convex* if the segment joining any two points of $C$ lies inside $C$. Then, by extension, the *metrically convex hull* (*convex hull*, for short) of a set of sites $P$ on $S$ can be defined as the smallest metrically convex set containing $P$, denoted $CH_S(P)$. It can be easily proved that, as in the plane, the convex hull $CH_S(P)$ can be obtained by intersecting all the convex sets containing $P$.

Next results will lead to the main theorem of this section, which will establish the relation between the convex hull of a point set $P$ in Euclidean position on $S$ and the planar convex hull of the counter-image of $P$.

From now on, if $P$ is in Euclidean position, we will denote $df(P)$ a fundamental domain containing every segment joining two points of $P$.

LEMMA 1. *If a set of sites $P$ on $S$ is in Euclidean position for a certain fundamental region $df(P)$, then so $CH_S(P)$ is for the same fundamental region.*

PROPOSITION 1. *Let $C$ be a connected set in the plane. Then, the following assertions hold,*

1. *$\varphi(C)$ is in Euclidean position if and only if $\varphi$ on $C$, $\varphi|_C$, is an isometry.*

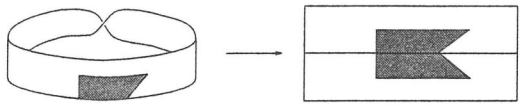


**Figure 3: The connected components in the plane of a convex set in the Mebius strip may not be convex.**

2. *If $\varphi(C)$ is in Euclidean position then, $\varphi(C)$ is convex if and only if $C$ is convex.*

Notice that it is not possible to rewrite part 2.) of the proposition above by simply considering a connected set $C$ in Euclidean position on $S$ and setting that $C$ is convex if and only if all the connected components of $\varphi^{-1}(C)$ are connected, because counterexamples to this result can be easily found, as it is shown in Figure 3.

Results above lead to the main theorem of this section.

THEOREM 4. *Let $P$ be a set of sites on $S$ in Euclidean position, $df(P)$ a fundamental region containing $P$ and $D$ a fundamental domain in the plane such that $df(P)$ can be obtained from it. Then, $CH_S(P) = \varphi(CH_{R^2}(\varphi^{-1}(P) \bigcap D))$.*

## 4. REFERENCES


[1] A. Márquez and C. I. Grima. *Computational Geometry on Surfaces.* Kluwer Academic Publisher, 2001.

[2] V. V. Nikulin and I. R. Shafarevich. *Geometries and Groups.* Springer Series in Soviet Mathematics. Springer, Berlin, 1987.

# Optimal Polygonal Schema on an Orientable Surface

Éric Colin de Verdière
Dépt. d'informatique
École normale supérieure (Paris)
France
Eric.Colin.de.Verdiere@ens.fr

Francis Lazarus
CNRS
University of Poitiers
France
lazarus@sic.sp2mi.univ-poitiers.fr

## 1. INTRODUCTION

Let $M$ be a compact, connected, orientable, boundaryless 2-manifold of genus $g$. Let $v_0$ be a point of $M$. A *reduced polygonal schema* of $M$, or *schema* for short, with basepoint $v_0$, is set $S$ of $2g$ simple loops $S_1, \ldots, S_{2g}$, meeting at $v_0$, pairwise disjoint except at $v_0$, such that the complement of the union of these loops is a topological disk. Such a schema is *canonical* if, in addition, the orientation of the loops around basepoint has the form $S_1, \hat{S}_2, \hat{S}_1, S_2, S_3, \ldots$ For a general reference on this subject, see for example [7].

Computing a (possibly canonical) schema on a surface is known to be useful in several problems where a correspondence between the surface and a topological disk needs to be established, like in surface parametrization [2] and texture mapping [5, 6]. For the construction of homeomorphisms between two oriented surfaces of same genus, a natural method is to compute a canonical schema for each surface and to find a correspondence between both disks. Schemata also provide a way to determine whether two curves are homotopic.

Lazarus, Pocchiola, Vegter, and Verroust [4] gave two different methods to compute a combinatorial canonical schema of the triangulated manifold $M$ with basepoint $v_0$ drawn on the vertex-edge graph of $M$ (it is a canonical schema in the ordinary setting if we spread all paths going along an edge with a thin space between them). However, their algorithms yield schemata with a larger complexity than actually required, and do not take into account the geometry of the surface.

We investigate a way to remedy this problem. We assume that each edge of the surface $M$ has a (fixed, positive) length, and start with any reduced schema $S$ of $M$ with basepoint $v_0$; we consider the set $\mathcal{S}$ of all schemata homotopic to $S$ with the same basepoint drawn on the vertex-edge graph of $M$. The result of this paper is twofold. First, we show that a schema $T$ of minimal length in $\mathcal{S}$ has the following property: for each $i \in \{1, \ldots, 2g\}$, $T_i$ is a shortest simple loop with basepoint $v_0$ homotopic to $S_i$. Second, we give an algorithm to compute such a $T$, with running time $O(g^3 n^3 \log n)$, if $n$ is the complexity of the surface, and assuming the ratio between the longest and shortest edge is bounded. The algorithm is quite simple, and we have implemented a variant of it. The cornerstone of our approach is a careful analysis of the possible ways of crossings between a schema $S$ and a loop homotopic to some $S_i$.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002)*
*April 10-12, 2002, Warsaw, Poland*

## 2. COMBINATORIAL SCHEMATA

Let $G = (V, E)$ be the vertex-edge graph of the surface $M$. We want to store a set of paths on $G$, with the additional data that the ordering of the paths along the edges is known. To this purpose, to each edge of $G$, oriented arbitrarily, we attach an ordered list representing the parts of the paths going along this edge, from left to right (Figure 1, left). Each element of the list is a *Path Data Structure* (PDS), consisting basically of two pointers to the previous (resp. next) PDS in this path. Since the paths end at the basepoint $v_0$, the structure near this vertex is exceptional: we put a "ring" around $v_0$ (one PDS list at each "corner", i.e., between two successive neighbors of $v_0$). We also put a *terminal PDS list*, designed to contain all points of the paths ending at $v_0$ (Figure 1, right). The whole structure is referred to as *Edge-Ordering Data Structure* (EODS).

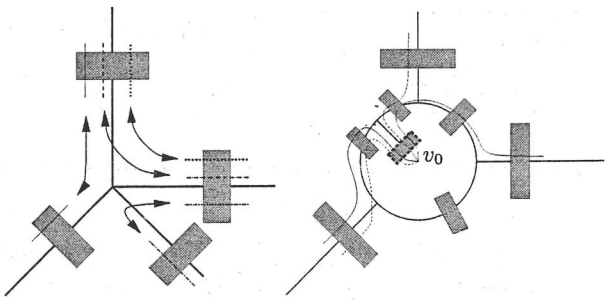



**Figure 1: Each rectangular box is a list of PDSs. On the left, an ordinary vertex with four incident edges. On the right, the basepoint $v_0$ has degree three, and four paths arrive at its terminal PDS list (with dashed, bold border).**

If $a$ and $b$ are PDSs arriving at a vertex $v$ and $a'$ (resp. $b'$) is the PDS next to $a$ (resp. $b$), then we say that $(a, a')$ *crosses* $(b, b')$ at $v$ if and only if, in the cyclic ordering around $v$, $a$ and $a'$ separate $b$ and $b'$. For example, no intersection occurs

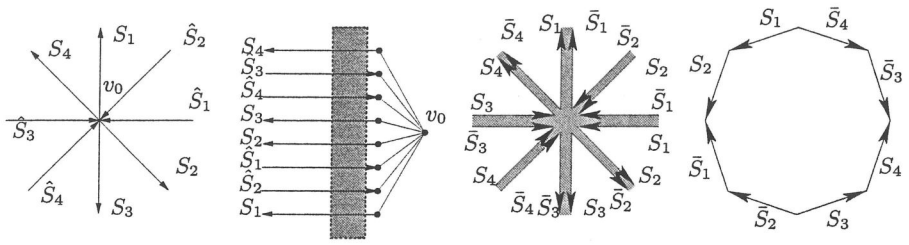



**Figure 2: How the ordering of the curves around basepoint and the simple curves are related (example for $g = 2$). From left to right: the loops meeting at the basepoint; the corresponding terminal PDS list; the cutting along the loops near the basepoint; and the topological disk $M(S)$ after this cutting.**

in Figure 1. A *simple path* in the EODS is a path without self-intersections. To each set of paths $S_1, \ldots, S_n$ drawn in the EODS of our triangulated manifold, we can associate a set of piecewise linear continuous paths $s_1, \ldots, s_n$ on the manifold which preserves the crossings. For this, we consider all curves going along an edge and spread them by a thin space. We assume that the endpoints of $s_i$ are really at $v_0$ (Figure 1, right).

DEFINITION 1. *A* (reduced) polygonal schema *on $M$ is a set of $2g$ oriented paths $S_1, \ldots, S_{2g}$ on the EODS of $M$, with endpoints on the terminal PDS list of the basepoint $v_0$, such that: (i) they do not self-intersect; (ii) any two paths do not cross; and (iii) cutting $M$ along the corresponding continuous paths $s_1, \ldots, s_{2g}$ yields a topological disk.*

In the figures of this paper, schemata are *canonical* (the ordering of the curves around the basepoint is $S_1$, $\hat{S}_2$, $\hat{S}_1$, $S_2$, $S_3$, $\hat{S}_4$, $\hat{S}_3$, $S_4$,..., where hat is used for paths coming to the basepoint), but it is not required in the algorithm.

The expression "cutting the surface $M$ along the loops $S_i$" has a precise meaning in the EODS framework, and we define $M(S)$ to be the surface with boundary obtained after this cutting (see Figure 2). We now define $M(S, i)$ to be the surface with boundary resulting from gluing infinitely many copies of $M(S)$ along $S_i$ (Figure 3). $M(S, i)$ can be viewed as a part of the universal covering space of $M$.

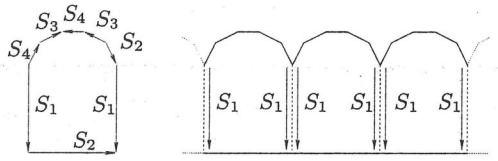



**Figure 3: On the left, the topological disk $M(S)$ (case $g = 2$). On the right, the space $M(S, 1)$.**

## 3. THE ALGORITHM

PROPOSITION 2. *Let $i \in \{1, \ldots, 2g\}$. Consider the set $\mathcal{P}$ of all paths, in the EODS of $M$, which do not cross any $S_j$ with $j \neq i$ and are homotopic to $S_i$ with the same endpoints. Then, there exists an element $S_i'$ in $\mathcal{P}$, which is not longer than any other element in $\mathcal{P}$, such that $S_i'$ is simple.*

PROOF (SKETCH). The proof is constructive, and we will use this construction in the algorithm. The idea is to use Dijkstra's algorithm to find a shortest path $p$, in the vertex-edge graph of $M(S, i)$, between the endpoints of a lift of $S_i$. Then, using the Jordan Curve Theorem, we can show that the projection of $p$, drawn on the vertex-edge graph of $M$, represents a path drawn in the EODS of $M$ which is simple. The implementation of Dijkstra's algorithm must take care of degenerate cases (several shortest paths between two points can exist in $M(S, i)$) to ensure this property. □

If $S$ is a polygonal schema and $i \in \{1, \ldots, 2g\}$, we define $S' = f_i(S)$ to be the schema $S$ where $S_i$ is replaced by $S_i'$ (the loop obtained by Proposition 2). Given a schema $S^0$ of $M$, we want to compute the shortest schema homotopic to $S^0$. Let $f = f_{2g} \circ f_{2g-1} \circ \cdots \circ f_1$. The whole algorithm is simply to iterate the function $f$ on $S^0$: let $S^{n+1} = f(S^n)$. Define a preorder (a reflexive and transitive relation) on homotopic schemata on $M$ by $S \preceq T$ if and only if, for each $i \in \{1, \ldots, 2g\}$, $S_i$ is not longer than $T_i$. Define $S \sim T$ if and only if $S \preceq T$ and $T \preceq S$. Here is our main result:

THEOREM 3. *$(S^n)_{n \geq 0}$ is a decreasing family of schemata homotopic to $S^0$ (with respect to "$\preceq$"); for some $k \in \mathbb{N}$, $S^k \sim S^{k+1}$, and in this situation, for each $j \in \{1, \ldots, 2g\}$, $S_j^k$ is a shortest loop among all simple loops homotopic to $S_j^0$. In particular, $S^k$ is a smallest element for "$\preceq$".*

## 4. CORRECTNESS OF THE ALGORITHM

LEMMA 4. *Let $M'$ be an oriented surface with boundary, and $c_1, \ldots, c_n$ be simple, pairwise disjoint, oriented curves, each of which separates $M'$ into two connected components. Let $\ell$ be an oriented loop such that every intersection of $\ell$ with a curve $c_i$ is transverse. Let $L$ be the (cyclic) list of crossings between $\ell$ and the $c_i$'s, in this order on $\ell$, taking into account the orientation of the crossings (with or without a bar). Then $L$ is a parenthesized expression.*

Henceforth, $D$ is a small topological open disk around $v_0$, and $M'$ is the universal covering space of $M \setminus D$. For a path $U$ drawn on the EODS of $M$, define $\dot{u}$ to be the part of the continuous version $u$ which is outside $D$.

LEMMA 5. *Let $U$ and $U'$ be homotopic simple paths drawn in the EODS of $M$, with endpoints $v_0$. There are two paths $\ell_D$ and $\ell_D'$ drawn on the boundary of $D$ so that $\dot{u}^{-1} \ell_D \dot{u}' \ell_D'$ is homotopic to a point in $M \setminus D$.*

The proof uses [1, Theorem 4.1] which states that, on an orientable surface, two piecewise linear, simple, homotopic loops with basepoint which are not null-homotopic are piecewise linearly isotopic, keeping the basepoint fixed.

Let $j \in \{1, \ldots, 2g\}$. From now on, we consider a schema $S$, and a simple loop $T_j$ homotopic to $S_j$. We represent $S$ and $T_j$ in the EODS of $M$: "erasing" the PDSs of $S$ (resp. $T_j$) yields $T_j$ (resp. $S$). (Of course, there can be several ways to do this.)

For each $i \in \{1, \ldots, 2g\}$, let $(\dot{s}_i^\alpha)_{\alpha \in \mathbb{N}}$ be the set of all lifts of $\dot{s}_i$ in $M'$, with an (arbitrary) orientation. Let $A$ be the set of *symbols* of the form $i^\alpha$ or $\bar{i}^\alpha$, where $i \in \{1, \ldots, 2g\}$ and $\alpha \in \mathbb{N}$. The set $A^\star$ of *words* on $A$ is the set of finite sequences of elements in $A$. We consider a fixed lift of $\dot{t}_j$ in $M'$ (also denoted by $\dot{t}_j$ in the following). Define $[\dot{s}/\dot{t}_j] \in A^\star$ to be the ordered list of the lifts $\dot{s}_i^\alpha$ (for all $i$ and $\alpha$) crossed by the lift of $\dot{t}_j$, the orientation of the crossing being indicated (with or without a bar). Let $i \in \{1, \ldots, 2g\}$. We define two types of *i-reductions* on an expression of the form $[\dot{s}/\dot{t}_j]$:

- a *parenthesized i-reduction* consists in removing an expression of the form $i^\alpha \bar{i}^\alpha$ or $\bar{i}^\alpha i^\alpha$ appearing in $[\dot{s}/\dot{t}_j]$;
- a *terminal i-reduction* consists in the removal of the first (resp. last) element of $[\dot{s}/\dot{t}_j]$, if it is of the form $i^\alpha$ or $\bar{i}^\alpha$ and one of the endpoints of $\dot{s}_i^\alpha$ can be linked to the source (resp. target) of $\dot{t}_j$ by a curve on the boundary of $M'$.

Any expression $e \in A^\star$ derived from $[\dot{s}/\dot{t}_j]$ by successive $i$-reductions is *i-irreducible* if no new $i$-reduction is possible. It is straightforward to check that, for any $[\dot{s}/\dot{t}_j]$, there is exactly one $i$-irreducible expression $e \in A^\star$ for $[\dot{s}/\dot{t}_j]$, and we define $g_i([\dot{s}/\dot{t}_j])$ to be this expression. Lemmas 4 and 5 enable us to show:

PROPOSITION 6. *$[\dot{s}/\dot{t}_j]$ can be reduced to the empty word.*

Intuitively, the strategy of the proof of Theorem 3 is to show that, if $T_j$ is a shortest simple loop homotopic to $S_j$, then applying $f_i$ to $S$ unties the intersections between $S_i$ and $T_j$:

PROPOSITION 7. *Let $i \in \{1, \ldots, 2g\}$. There exists a simple loop $T'_j$, homotopic to $S_j$ and $T_j$, not longer than $T_j$, such that $[f_i(\dot{S})/\dot{t}'_j] = g_i([\dot{s}/\dot{t}_j])$.*

PROOF (SKETCH). Let $R = f_i(S)$. It is easy to see that $g_i([\dot{s}/\dot{t}_j]) = g_i([\dot{r}/\dot{t}_j])$. We show that, if an $i$-reduction is possible on $[\dot{r}/\dot{t}_j]$, then there exists a closed curve in $M'$, made of a part of a lift of $\dot{r}_i$, a part of a lift of $\dot{t}_j$, and possibly a part of the boundary of $M'$, which is crossed by no lift of $\dot{r}$ and $\dot{t}_j$. This enables to build, in the EODS of $M$, a simple loop $T'_j$ homotopic to $T_j$, not longer than $T_j$, such that $[\dot{r}/\dot{t}'_j]$ is deduced from $[\dot{r}/\dot{t}_j]$ by an $i$-reduction. The proof is finished by induction. □

The preceding proposition enables to prove Theorem 3, because, if $[\dot{s}/\dot{t}_j]$ is the empty word, then a lift of $T_j$ is in fact in $M(S, j)$, hence $f_j(S)_j$ is not longer than $T_j$.

## 5. COMPLEXITY ANALYSIS

We now give an upper bound on the complexity of our algorithm. Let $n$ be the number of edges of $M$, $j \in \{1, \ldots, 2g\}$, and $S$ be a schema homotopic to $S^0$.

LEMMA 8. *Suppose there exists a shortest simple loop $T_j$ homotopic to $S_j$ such that $S_j^0$ and $f_j(S)_j$ cross $T_j$ at most $C_j$ times. Then, computing $f_j(S)$ is possible in time $O(k \log k)$, where $k = (n + |S| + |S^0|)C_j$. ($|S|$ is the number of edges of schema $S$.).*

PROOF (SKETCH). Consider a part $\bar{S}_j^0$ of $S_j^0$ which goes from one boundary of $M(S, j)$ to the other one. A lift of $f_j(S)_j$ in $M(S, j)$ crosses at most $2C_j + 1$ lifts of $\bar{S}_j^0$, hence a lift of $f_j(S)_j$ is confined to the space made of $2C_j + 2$ patches delimited by two consecutive lifts of $\bar{S}_j^0$ in $M(S, i)$. This space has size $O(k)$; the lemma follows. (It is not required to know $C_j$ in advance: we can prove that searching through an exponentially increasing number of patches leads to the announced complexity.) □

Note that the values of $C_j$, for all schemata $S$ considered in the algorithm, are bounded from above by the maximum number of crossings between $S_j^0$ and any shortest simple loop $T_j$ homotopic to $S_j^0$.

Henceforth, we assume that the algorithm starts with a schema $S^0$ obtained with one of the methods described in [4]. It can be shown that an oriented edge of $M$ may appear at most twice in each of the $2g$ loops of $S^0$ (whence each loop has linear size). The next lemma follows easily:

LEMMA 9. *The number of crossings between $S_j^0$ and a loop $\ell'$ is bounded by twice the number of edges of $\ell'$.*

Let $c_j$ denote the minimum number of crossings between schema $S^0$ and a shortest simple loop homotopic to $S_j^0$. The preceding section actually shows that no more than $\max_{1 \leq j \leq 2g} c_j + 1$ main steps are needed to complete the algorithm.

Let $L$ be the maximum over $j$ of the maximal number of edges of any shortest simple loop homotopic to $S_j^0$. From Lemmas 8 and 9 we get that the total time spent by the algorithm is $(g^2 L^2 m \log(mL))$, where $m$ is the maximum of $(n + |S^k| + |S^0|)$ over the shortening iterations. To get a more practical bound we assume that the ratio of the longest edge length to the shortest one in $M$ is bounded by $\alpha$:

THEOREM 10. *Assume we are given a schema of an orientable triangulated surface, $M$, such that the number of times a loop can go along a given edge is bounded from above by a constant. Then, there is an algorithm that computes a homotopic schema with minimal length in $O(\alpha^3 g^3 n^3 \log \alpha n)$ time.*

Remark: According to [3], the logarithmic term in the theorem could actually be removed.

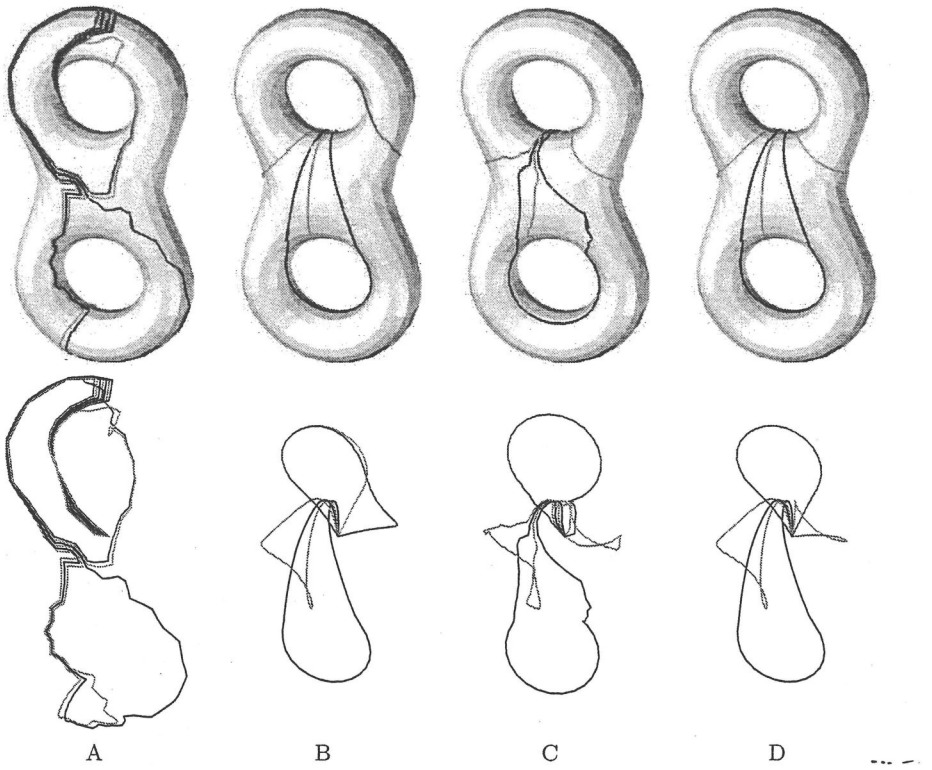



**Figure 4: A: A canonical schema, $S$, obtained after [4]. The basepoint is on the back side of the double torus. B: A geodesic schema, homotopic to $S$, computed with our local optimizer. 30,000 star optimizations were performed. C: The result of our algorithm on $S$. Only 4 iterations of each loop were necessary to obtain an optimal schema (Euclidean distances were used for the edges). Note the change of homotopy class. D: The local optimization was applied to this optimal schema to get a geodesic schema on the surface.**

## 6. IMPLEMENTATION

We have implemented a slightly different version of the algorithm using the C++ based CGAL library. The presented algorithm shortens each loop $S_i$ in turn by computing a shortest path homotopic to $S_i$ in $M(S,i)$. In the implemented version, we replace $S_i$ by a shortest among all paths whose endpoints are lifts of the basepoint and going from one boundary of $M(S,i)$ to the other one. Hence, loops may change of homotopy class, but the algebraic relation satisfied by the loops, and the fact that the schema is canonical or not, is preserved.

In order to make comparisons, we also implemented a simple local optimization that produces geodesic loops *on the surface* of $M$: we visit each vertex star of $M$ and replace pieces of loops in the star by shortest paths in the star. We repeat this operation until the shortening gain is below a given threshold. The resulting loops are geodesics (not necessarily the shortest ones) and keep their homotopy class.

Figure 4 shows a simple example run on a genus 2 torus with 1536 facets.

## 7. ACKNOWLEDGEMENTS

We wish to thank Michel Pocchiola for interesting discussions and comments.

## 8. REFERENCES

[1] D. Epstein. Curves on 2-manifolds and isotopies. *Acta Mathematica*, 115:83–107, 1966.

[2] M. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[3] M. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1, part 1):3–23, 1997.

[4] F. Lazarus, M. Pocchiola, G. Vegter, and A. Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proceedings of the 17th Annual Symposium on Computational Geometry*, pages 80–89, 2001.

[5] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *SIGGRAPH 93*, pages 27–34, 1993.

[6] D. Piponi and G. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *SIGGRAPH 2000*, pages 471–478, 2000.

[7] J Stillwell. *Classical topology and combinatorial group theory*. Springer-Verlag, New York, 1993.

# Grid representations of graphs on surfaces*

N. de Castro
University of Seville, Spain
natalia@us.es

## ABSTRACT

We present a complete characterization of the class of graphs which admit an orthogonal tessellation representation on the cylinder and on the torus, where each constituent (vertex, edge and face) of an embedded graph on these surfaces is represented by a rectangle and incidencies between constituents correspond to geometric adjacencies between rectangles. We also characterize the graphs that admit a segment grid contact representation on those surfaces, where each vertex is represented as a horizontal or vertical segment and two vertices are adjacent if their correspondent segments have a contact.

## 1. INTRODUCTION

Recently automatic drawing of graphs has created intense interest due to their broad range of applications and, as a consequence, a number of drawing styles and corresponding drawing algorithms have emerged [1]. Among different drawing styles, an "orthogonal drawing" has attracted much attention due to its numerous applications in circuit layouts, database diagrams, entity-relationship diagrams, etc. [2], [5], [13].

A *tessellation representation* on a surface $S$ (the cylinder or the torus) for a map $M$ of a graph $G$ is a partition of $S$ into disjoint rectangles (possibly degenerate), each associated with a vertex, edge or face of $M$, such that topological incidences in $M$ correspond to geometric adjacencies between rectangles. This problem was firstly studied by Tamassia and Tollis [11] who gave a linear time algorithm to construct tessellation representations on the plane and on the sphere.

Mohar and Rosenstiehl [10] study tessellation representation for maps on the *skew* torus (i.e., a torus whose horizontal and vertical grid directions are not parallel with the sides of the base rectangle). Those authors point out the interest of characterize the graphs that admits a orthogonal tessellation representation on the torus. Our purpose is to characterizing the graphs which admit an orthogonal tessellation representation on the torus and also on the cylinder.

Our Theorems provide a natural and intrinsic characterization of the graphs that admit a segment contact grid representation on these surfaces. In this representation, each vertex of the graph is represented as a horizontal or vertical segment, and two vertices are adjacent if their segments have a contact.

Contact graphs of segments and other geometrical objects have been widely studied in the past. An interesting result is due to de Fraysseix, Osona de Mendez and Pach [6]. Those authors prove that every planar bipartite graph is the intersection graph of a set of horizontal and vertical segments. Moreover, every triangle-free planar graph can be represented as a segment contact graph using just three prescribed distinct directions of segments [3]. On the other hand, it is known that the recognition of such graphs is an NP-complete problem (see [7], [8] and recently [4]).

## 2. TESSELLATION REPRESENTATIONS

Throughout this work we assume that a *graph* $G$ is a so-called multigraph which may have *multiple edges*, i.e., edges sharing both ends. If $G$ has no multiple edges, then $G$ is called a *simple* graph.

A *map* $M$ on a surface $S$ is a connected graph $G$ together with 2-cell embedding of $G$ in $S$.

### 2.1 Tessellation representation on the cylinder

In the most usual way of representing graphs (topological representation) vertices map to points in the plane or other surfaces, edges are represented by Jordan curves joining those points. Recently, some other ways to represent graphs have been proposed (intersection graphs or, in particular, contact graphs, grid graphs, etc.) where vertices are represented by segments and edges are represented by relations between the segments (visibility, intersection, contact, etc).

An interval of the cylinder $C$ is a connected subset of a line of $C$, and an arc is a connected subset of a circumference of $C$. In a tessellation representation for a cylindric embedding of a graph, each vertex $v$ is associated with an interval $\mathcal{T}(v)$ (or horizontal segment) of $C$ and each face $f$ is associated with

*Partially supported by MCyT project BFM2001-2474

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

an arc $\mathcal{T}(f)$ (or vertical segment) of $C$, in such way that a vertex $v$ belongs to $f$ if and only if $\mathcal{T}(v)$ is adjacent to $\mathcal{T}(f)$. Thus, we obtain a family of rectangles on the cylinder, and since each rectangle is delimited by two horizontal segment $\mathcal{T}(u)$ and $\mathcal{T}(v)$, we can associate it with the edge $(u, v)$.

The tessellation representation for a map is closely related with the visibility representation of maps, where vertices are associated with horizontal intervals and there is an edge between two vertices if their intervals can see each other in the vertical direction [12].

Unless otherwise stated, we assume that the arcs associated with the faces can not complete a circumference of $C$. Thus, we can prove the following Theorem.

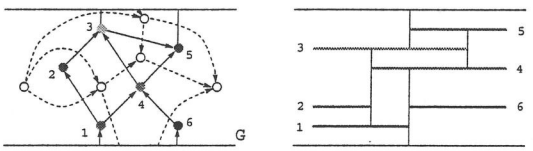



**Figure 1: Tessellation representation on the cylinder.**

THEOREM 1. *A graph $G$ with $n$ vertices admits a tessellation representation on the cylinder if and only if it is 2-connected. Also, a tessellation representation for a map of $G$ on the cylinder can be constructed in $O(n)$ time.*

## 2.2 Tessellation representation on the torus

In this section we consider maps on the torus which can be represented by a rectangle $Q$ in the plane whose opposite sides are pairwise identified. This representation is known as the *flat torus* and allows us to define two orthogonal families of parallel lines. Let $M$ be a map of a graph on the torus. If every facial walk $f$ of $M$ is a simple cycle, we say that $M$ is a *closed 2-cell embedding.*

In a tessellation representation for a toroidal map, each vertex $v$ is associated with a horizontal segment $\mathcal{T}(v)$ of $C$ and each face $f$ is associated with a vertical segment $\mathcal{T}(f)$ of $C$, in such a way that a vertex $v$ belongs to $f$ if and only if $\mathcal{T}(v)$ is adjacent to $\mathcal{T}(f)$. According to the above remark, the edges of the graphs are associated with the rectangles delimited by these families of segments.

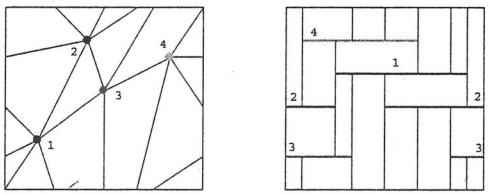


**Figure 2: Tessellation representation on the torus.**

THEOREM 2. *A toroidal map admits a tessellation on the torus if and only if it is a closed 2-cell embedding.*

# 3. SEGMENT REPRESENTATION

Two families of horizontal and vertical segments of the plane or other surface, each one being disjoint from the others except for some contact between two segments of different families, define a bipartite graph, called the *contact graph of the segments families.*

Theorems 1 and 2 now give a characterization of the graphs which can be represented as cylindrical or toroidal contact graph on these surfaces.

THEOREM 3. *Let $M$ be a map of a graph $G$ on $S$ (the cylinder or the torus). $M$ can be represented as the contact graph of horizontal and vertical segments on $S$ if and only if $G$ is a bipartite simple graph.*

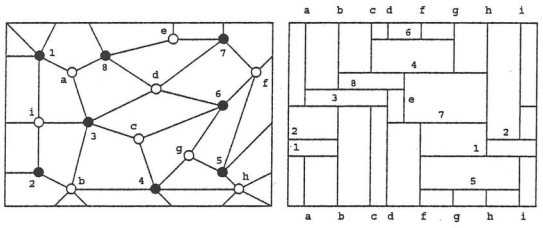


**Figure 3: Segment representation of a toroidal map.**

# 4. CONCLUDING REMARKS

In section 2 we have characterized the graphs which admit a tessellation representation on the cylinder and on the torus with the hypothesis that the segments which represents vertices and faces can not complete a circumference. This hypothesis can be relaxed. We give in this section a new version of Theorems 1 and 2.

A *block* of a graph $G$ is a maximal 2-connected subgraph of $G$. A *bridge* of $G$ is an edge whose removal disconnects $G$. The *block-cutvertex tree* of $G$ is the tree whose vertices are the blocks and the cutvertices of $G$, and whose edges connect every block $B$ to the cutvertices contained in $B$. A *chain* is a tree that has at most two leaves and a *centipede* is a tree whose non-leaf vertices form a chain.

Let $M$ be a toroidal map that is not a closed 2-cell embedding, then there is at least one face $f$ whose facial walk is not a simple cycle. We call *essential vertex* the one which appears twice on the facial walk of $f$. If there are two adjacent vertices $(u, v)$ that appear twice on the facial walk of $f$, we call this edge an *essential edge.*

THEOREM 4. *A graph admits a tessellation representation on the cylinder if and only if it has no bridges and its block-cutvertex tree is a centipede. Also, a tessellation representation for a map of the graph on the cylinder can be constructed in linear time.*

THEOREM 5. *A map on the torus admits a tessellation representation if and only if it has no essential edges.*

However, these new versions of the Theorems 1 and 2 could not be used to characterize the contact segment grid.

## 5. ADDITIONAL AUTHORS

Additional authors: F.J. Cobos (University of Seville, Spain, email: cobos@us.es), J.C. Dana (University of Seville, Spain, email: dana@us.es) and A. Márquez (University of Seville, Spain, email: almar@us.es) .

## 6. REFERENCES

[1] G. Di Battista, P. Eades, R. Tamassia and I.G. Tollis, "Algorithms for drawing graphs: an annotated bibliography", *Computational Geometry: Theory and Applications*, Vol. 4(5), pp. 104–110, 1994.

[2] T.C. Biedl, "Optimal orthogonal drawings of triconnected plane graphs", *Proc. SWAT 96. Lecture Notes in Computer Science. Springer-Verlag, Berlin/ NY*, Vol. 1097, pp. 333-344, 1996.

[3] N. de Castro, F.J. Cobos, J.C. Dana, A. Márquez and M. Noy, "Triangle-free planar graphs as segment intersection graphs", *JGAA*, Vol. 5 no. 6, 2002.

[4] P. Hlineny, "Contact graphs of line segments are NP-complete", *Discrete Mathematics*, Vol. 235, pp. 95–106, 2001.

[5] G. Kant, "Drawing planar graphs using the canonical ordering",*Algorithmica*, Vol. 16, pp. 4-32, 1996. 1991.

[6] H. de Fraysseix and P.O. de Mendez and J. Pach, "Representation of planar graphs by segments", *Colloquia Mathematica Societatis János Bolyai, Intuitive Geometry*, 1991.

[7] J. Kratochvíl, "A special planar satisfiability problem and a consequence of its NP-completeness". *Discrete Applied Math.*, Vol 52, pp. 233–252, 1994.

[8] J. Kratochvíl and J. Matoušek, "Intersection graphs of segments". *J. of Comb. Theory, Serie B*, Vol 62, pp. 289–315, 1994.

[9] B. Mohar and N. Robertson, "Disjoint essential circuits in toroidal maps", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 9, pp. 109-130, 1993.

[10] B. Mohar and P. Rosenstiehl, "Tessellation and visibility representations of maps on the torus", *Discrete Computational Geometry*, Vol. 19, pp. 249–263, 1998.

[11] R. Tamassia and I.G. Tollis, "Tessellation representations of planar graphs", *Proc. 27th Annual Allerton Conf.*, pp. 48-57, 1989.

[12] R. Tamassia and I.G. Tollis, "A unified approach to visibility representations of planar graphs", *Discrete and Computational Geometry*, Vol. 1 pp. 321-341, 1986.

[13] R. Tamassia, I.G. Tollis and J.S. Vitter, "Lower bounds for planar orthogonal drawings of graphs", *Inform. Process Letter*, Vol. 39 pp. 35-40, 1991.

# Flat-face embeddings of certain 2-complexes

[extended abstract]

Colm Ó Dúnlaing *
Mathematics, Trinity College, Dublin 2, Ireland

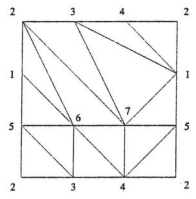



**Figure 1: Möbius torus, a 7-vertex triangulation of the torus.**

## ABSTRACT

A graph $G$ is a finite simplicial 1-complex. If it is planar, that is, there exists a topological embedding of $G$ in the plane, then there exists a straight-edge embedding. Analogously: if a finite simplicial 2-complex can be embedded topologically in 3 dimensions, can it be embedded with flat faces? Certain surprising positive examples exist, such as the Möbius torus [1] (see Figure 1), where the answer is yes, but it is thought that triangulations of surfaces of higher genus might not be flat-embeddable.

We show a simple positive result: if the 2-complex is realised as a tetrahedral subdivision of the unit ball in 3 dimensions, then it admits a flat-face embedding.

We assume that the given subdivision is sufficiently smooth, and basing our ideas on Morse Theory, we alter the embedding to a suitable form, related to the canonical representation considered in [4]. Then we produce a flat-face embedding.

*e-mail: odunlain@maths.tcd.ie. Mathematics department website: http://www.maths.tcd.ie. A full version of this paper will be in the TCDMATH report series at this website.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

This is a nonconstructive existence proof, and leaves the way open for an efficient algorithm. We mention two efficient planar layout methods, barycentric maps [9] and Read's vertex-deletion method [8], showing that they cannot be adapted to our embedding problem.

Nevertheless, the area looks promising for further research.

## 1. FINITE SIMPLICIAL 2-COMPLEXES AND SPATIAL REALISATIONS

A finite simplicial 2-complex consists of a finite set $V$ of *vertices*, together with a set $E$ of *edges*, unordered pairs $\{u, v\}$ of distinct vertices, and a set $F$ of *faces*, unordered triples $\{u, v, w\}$ of pairwise distinct vertices. Thus every edge has two endpoints and every face has three corners and three bounding edges. It is assumed that if $f \in F$ is a face then its three bounding edges are in $E$.

In this paper '2-complex' will mean finite simplicial 2-complex.

A 2-complex can be used to define a topological space as follows. The *moment curve* in $\mathbf{R}^5$ is

$$\{(t, t^2, t^3, t^4, t^5) : t \in \mathbf{R}\}.$$

Given a finite 2-complex $K$, let its vertices be mapped to distinct points on the moment curve. For each edge $\{u, v\}$ there is a line-segment joining the images of $u$ and $v$ on the moment curve, and for each face $\{u, v, w\}$ there is a triangular convex region whose corners are the images of $u$, $v$, and $w$. It is known that these line-segments intersect only at endpoints, and the triangles intersect only at endpoints or along bounding edges. The union of these points, edges, and faces is a compact topological space which, by abuse of notation, we shall also denote by $K$.

A *spatial realisation* of $K$ is a topological embedding of $K$ in 3-dimensional space $\mathbf{R}^3$. This is analogous to a planar embedding of a graph. Some 2-complexes admit spatial realisations, some do not (for example, non-orientable triangulated surfaces).

The related property of planar graph embedding is well understood. Graph planarity is concerned with the existence of topological embeddings, but straight-edge embeddings are the simplest, and all planar graphs admit straight-edge embeddings [5]. However, it is not known (though believed false) whether all spatially realisable 2-complexes admit flat-face embeddings. This abstract introduces a class of 2-complexes for which flat-face embeddings exist.

## 2. MONOTONE EMBEDDINGS OF CERTAIN 2-COMPLEXES

**The restricted kind of complex studied here.** For the rest of this paper, $B$ will denote a closed 3-ball in $\mathbf{R}^3$, i.e., a subset of $\mathbf{R}^3$ homeomorphic to the set $\{p\colon |p| \leq 1\}$.

It is assumed that we are given a 2-complex $K$ which has already been embedded in 3 dimensions, as a subdivision of a closed ball $B$. Thus the external faces together form a topological 2-sphere. Since our most important construction involves Morse-like functions and critical points, we assume that the subdivision is smooth, with nondegenerate critical points, etcetera [6].

The following definition has features in common with the canonical representation of planar graphs described in [4].

**(2.1) Definition** *Given $K$ as a subdivision of $B$, it is* monotone embedded *if every cell in the subdivision contains a unique maximum (in the $z$-direction), located at one of the four vertices on the cell, and every cell except one contains a unique minimum, at another vertex; and all vertices are at different heights, except for the lowest three vertices, which bound an external face, which is flat and horizontal.*

*The cell incident to the bottom face does not, of course, have a unique minimum.*

**(2.2) Proposition** *For any external face of $K$, $K$ admits a monotone embedding in which the given face is the bottom face*

Idea: choose co-ordinates so the three bottom vertices are co-horizontal. Use an isotopy to flatten the bottom face, pushing some of $K$ above the face. After this, alter the cell boundaries to remove saddle-points. By this time, all cells except the bottom one will have exactly one maximum and one minimum. Finally, alter the cells to ensure that these local maxima and minima are all at vertices.

**(2.3) Definition** *Given an interior vertex $v$ of $K$ (satisfying Proposition 2.2), its* link *consists of all faces $\{w,x,y\}$ such that $\{v,w,x,y\}$ is a cell of $K$ — i.e., all faces opposite $v$. Its* lower link *consists of all faces $\{w,x,y\}$ in the link such that $x, y$, and $z$ are below $v$.*

**(2.4) Lemma** *If $K$ is monotone embedded, then the lower link of any vertex $v$ is simply connected.*

**(2.5) Definition** *Let $R$ be a region in $\mathbf{R}^n$ (we only consider $n = 2$). A point $q \in R$ is* visible *from a point $p \in R$ if the line-segment $pq$ is entirely within $R$. The set $V$ of points in $R$ from which* all *points are visible is called the* visibility set *of $R$. $V$ can be empty, but if $V \neq \emptyset$ then $R$ is called* star-shaped *and if $V$ has nonempty interior then $R$ is* strictly star-shaped.

## 3. FLAT-FACE EMBEDDING OF A MONOTONE EMBEDDED COMPLEX

We assume that the complex $K$ is a finite 2-complex which has been embedded as a partition of a closed 2-ball, and the embedding is is monotone. A vertex $q$ in $K$ is *at or below* another vertex $p$ if the $z$-coordinate of $q$ is $\leq$ that of $p$.

**(3.1) Notation** *For any vertex $p$ of $K$, $K(p)$ is the full subcomplex spanned by the set $U$ of all vertices at or below $p$.*

*In other words, the vertices of $K(p)$ are those in $U$, and the cells, faces, and edges of $K(p)$ are those in $K$ all of whose incident vertices are in $U$.*

**(3.2) Definition 'Mountainous configurations.'** *We shall say that $F(p)$ is a* mountainous configuration *when $F(p)$ is a flat-face embedding of $K(p)$ with two further properties:*

*(i) all faces except the bottom are upward-facing (i.e., their outward normals have positive $z$-component), and*

*(ii) the vertices are in general position, with no four coplanar.*

**(3.3) Theorem** *For all vertices $p$ there exists an embedding $F(p)$ which is a mountainous configuration.*

*Therefore there exists a flat-face embedding of $K$.*

**Sketch of proof.** We construct $F(p)$ iteratively along the lines of the de Fraysseix-Pach-Pollack layout of a planar graph [4, Proposition 1]. We begin by placing the bottom three vertices on the $xy$-plane. This defines the bottom face. After this, the other vertices $p$ of $K$ are taken in ascending order of $z$-coordinate (as given in the original monotone embedding), and $F(p)$ is constructed.

When adding a vertex $p$ its lower link is a simply-connected union of faces (Lemma 2.4).

Let $\Pi$ be the projection of the lower link onto the $xy$-plane. Since the faces face upwards, projection is one-to-one on the lower link so $\Pi$ is a simply-connected union of triangles, i.e., a simple polygon plus its interior.

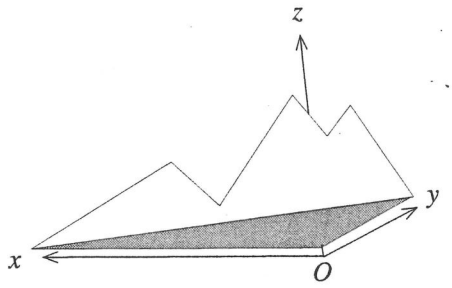



**Figure 2: a mountainous configuration.**

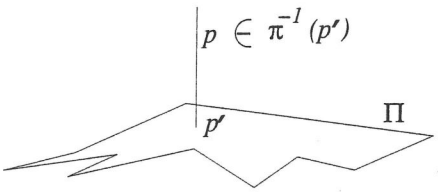



**Figure 3: $p$ is on the vertical line directly above $p'$, a visibility point in $\Pi$.**

If $\Pi$ is strictly star-shaped, then one can correctly position $p$, by taking a point $p'$ interior to its visibility set, and positioning $p$ directly above $p'$ and sufficiently high so that all faces in the lower link are visible. See Figure 3.

We cannot assume that $\Pi$ is star-shaped, and need the following result. In it, $q$ is the vertex just below $p$ in the monotone embedding $K$, $F(q)$ has been constructed, $P$ is the lower link of $p$, $\Pi = \pi(P)$, and we are attempting to construct $F(p)$.

**(3.4) Lemma** *Given $F(q)$, let $P$ be a simply-connected union of faces on the upper boundary of $F(q)$. Then there exists a similar embedding $F'$ of $K(q)$ as a mountainous configuration in which the vertical projection of $P$ is strictly star-shaped.*

**Sketch proof.** By induction on the number $t$ of faces in $P$: certainly true if $t = 1$ or 2. Suppose that we have a flat-face embedding $F(q)$ of $K(q)$ in which $P$ projects onto a strictly star-shaped region, and $P'$ is a simply-connected set obtained by attaching one more face along the boundary of $P$. If the new face has two edges in common with $P$ then the projection of $P'$ has the same or a larger visibility set, so suppose that it has one edge in common with $P$, and the projection of $P'$ is not star-shaped. Let $u, v, w$ be the vertices bounding this face, supposing that $u$ and $v$ are on $P$ but $w$ is not.

**(3.5) Definition** *Given $F(q)$ and $P$, let*

$$Q = P \cup \{\textit{bottom vertices}\},$$

*and let $h$ be some number greater than the height of $q$ in $F(q)$. Then*

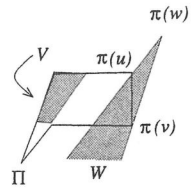



**Figure 4: Region $W$ from which additional triangle is visible, and visibility region $V$.**

*$C(F(q), P, h)$ is the set of all flat-face embeddings $F'$ of $K(q)$ with the following three properties: (a) $F'$ is a mountainous configuration, (b) $F$ and $F'$ agree on $Q$, so all vertices in $Q$ are fixed, and (c) $F'(q)$ has height at most $h$.*

These conditions ensure $C(F(q), P, h)$, viewed as a subspace of $\mathbf{R}^N$ for some $N$, is bounded, and that $P$ and $\Pi$ are the same for all configurations in this set.

Given $F' \in C(F(q), P, h)$, let $V$ be the visibility set of $\Pi$ It is the same for all such $F'$. Let $W$ be the wedge-shaped region bounded by two infinite rays from $\pi(w)$ through $\pi(u)$ and $\pi(v)$. The intersection $V \cap W$ is the visibility set of $\pi(P')$, so we need to ensure this intersection has nonempty interior. See Figure 4.

The 'Minkowski separation' of $W$ from $V$is

$$\inf\{|p_1 - p_2| : p_1 \in V, p_2 \in W\}.$$

Since $W$ is closed and $V$ is compact, the separation is zero if and only if $V \cap W \neq \emptyset$.

The closure of $C(F(q), P, h)$, being bounded, is compact, and the minimum separation is attained for some $M$ in this closure. We want to show that in $M$, the separation is zero. $M$ is a limit of valid configurations, but may not be valid: e.g., some cells may be degenerate.

If the separation of $W$ from $V$ in $M$ is nonzero, there must be some restrictions on the directions in which $w$, or rather $\pi(w)$, could be moved without producing an invalid configuration. For example, the projection $\Pi'$ of $P'$ may fail to be a simple polygon, meaning some outer face incident to $w$ may becomes vertical, or some cell incident to $w$ may be degenerate. See Figure 5.

One of the sides $wu$ or $wv$ must touch $\pi(z)$ for some other vertex $z$. If that vertex is not in $Q$ then it can be displaced allowing the separation to be reduced. Hence $z \in Q$.

Suppose that $\pi(z)$ touches the interior of $\pi(wu)$. We may suppose that the length of $\pi(wu)$, call it $\ell$, is minimal in the sense that we cannot bring $\pi(w)$ closer to $\pi(u)$ on this line. Since $\ell$ is minimal, $\pi(w)$ must be involved in some other degeneracy, but that degeneracy can be removed in its turn. This implies that $w$ and $z$ actually coincide, $w$ can be displaced away from $q$ and again the separation

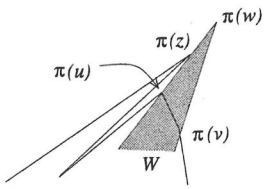



**Figure 5: $\pi(wu)$ touches $\pi(z)$.**

from $V$ be reduced.

By such arguments it can be shown that a possibly degenerate configuration exists in which the Minkowski separation is zero. Further perturbation arguments lead to a (nondegenerate) configuration in $C(F(q), P, h)$ in which $\Pi'$ is strictly star-shaped, allowing Lemma 3.4 to be proved by induction, and hence Theorem 3.3 can be proved by induction.

## 4. ADDITIONAL REMARKS

**(4.1)** This work attempts to extend some ideas from planar graph-embedding to the embedding of 2-complexes in 3 dimensions (a difficult problem: see [7]).

We believe this to be a novel line of research. The first problem is to replace the pseudo-constructive existence proof by a reasonably efficient algorithm.

**(4.2)** Note that for any $K$ the corresponding flat-embedded complex can be described by a suitable formula of Tarski geometry [2], hence is effectively constructible, but there must be more efficient methods.

**(4.3)** Rather than requiring an explicit embedding of $K$, one should be able to recognise suitable $K$ based on combinatorial properties, in the same way that the faces of a planar graph can be identified with peripheral polygons [9].

**(4.4)** We consider only subdivisions of a closed ball. It would be of great interest to study subdivisions of solid tori and solids of higher genus.

**(4.5)** We conclude with two efficient planar layout methods and show that they cannot be adapted to the present problem.

**(4.6) Barycentric mappings.** Tutte showed the following remarkable fact [9]: let $G$ be a nodally triconnected planar graph, and suppose that vertices on one of its faces are mapped onto the corners of a convex polygon. There is a unique barycentric extension of this map to the other vertices. 'Barycentric' means that the image of every non-outer vertex is the average of the images of its neighbours. Then this barycentric map defines a straight-line embedding of $G$.

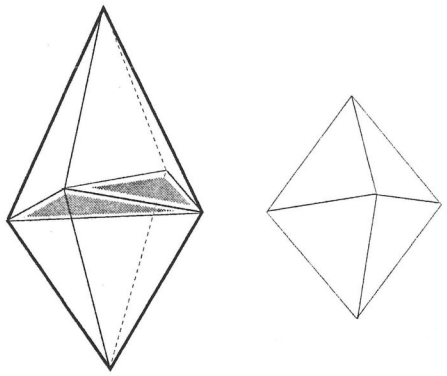


**Figure 6: Barycentric mapping can be degenerate**

Figure 6 shows that this idea doesn't work for 2-complexes. The complex is a subdivision of the closed ball (viewed as an irregular octahedron plus its interior) into five cells. Mapping its boundary to a regular octahedron flattens the middle cell. This difficulty cannot be resolved by placing the six vertices in general position: while some arrangements will produce an embedding, others will evert the middle cell so that the mapping is not one-to-one.

**(4.7) Read's algorithm [8].** This simple algorithm lays out a triangulated planar graph recursively as follows: choose a low-degree vertex $v$, delete it, re-triangulate the new face containing $v$, lay out the modified graph, and restore $v$ and its incident edges. The triangulation chosen must not introduce edges inside the face which already exist outside the face. The algorithm uses a simple *'fan-triangulation'* where all triangles radiate from a single vertex $w$. A suitable vertex $w$ exists by interlacing arguments.

The same algorithm would work for our 2-complexes if we could establish the existence of a triangulation: we would have an efficient algorithm. But this is not always possible.

As a counterexample, we use a 2-complex $K$ which has 6 vertices and is a simplicial subdivision of a tetrahedron in 3 dimensions. Its 1-skeleton is $K_6$, i.e., for every two vertices $u$ and $v$, there is an edge $\{u, v\}$. To construct it, take six distinct points on the moment curve in 4- (not 5-) dimensional space. Their convex hull is a neighbourly 4-polytope [3] whose boundary is equivalent to a 3-complex $L$, a simplicial subdivision of the 3-sphere. A suitable perspective projection takes $L$ minus one 3-cell to the desired flat-face simplicial subdivision of a tetrahedron.

Delete one of the two interior vertices $v$, and you are left with a 2-complex whose edges form the complete graph $K_5$ and in which the cell containing $v$ cannot be triangulated, for the simple reason that all pairs of points on its boundary are already connected by edges.

## 5. REFERENCES


1. J. Bokowski and A. Eggert (1986). All realizations of Möbius' torus with 7 vertices. Preprint 1009, FB Mathematik, Technische Hochschule Darmstadt.

2. J. Bokowski and B. Sturmfels (1989). *Computational Synthetic Geometry.* Springer Lecture notes in Mathematics 1355.

3. A. Brøndstred (1983). *An Introduction to Convex Polytopes.* Springer Graduate Texts in Mathematics **90**.

4. H. de Frayssex, J. Pach, and R. Pollack (1990). How to draw a graph on a grid. *Combinatorica* **10**, 41–51.

5. Goos Kant (1993). *Algorithms for drawing planar graphs.* Ph. D. dissertation, Computer Science, University of Utrecht.

6. John Milnor (1963). *Morse Theory.* Annals of Mathematics studies no. 51, Princeton University Press.

7. C. Ó Dúnlaing, C. Watt, D. Wilkins, and C-K Yap (1995). Miscellaneous topological algorithms. Report ALCOM-II-430 (1995) and TCDMATH 97-01 (1997).

8. R.C. Read (1987). A new method for drawing a graph given the cyclic order of edges at each vertex. *Congressus Numerantium* **56**, 31–44.

9. W. T. Tutte (1963). How to draw a graph. *Proc. London Math. Soc.* **3:13**, 743–768.

# On Flips in Polyhedral Surfaces: a new development

## [Extended Abstract]

Lyuba Alboul *
University of Twente
P.O. Box 217
7500 AE Enschede
the Netherlands
alboul@math.utwente.nl

Ruud van Damme
University of Twente
P.O. Box 217
7500 AE Enschede
the Netherlands
vandamme@math.utwente.nl

## ABSTRACT

Let $V$ be a finite point set in $3D$ and let $ST(V)$ be the set of closed triangulated polyhedral surfaces with a vertex set $V$. Those surfaces can be defined as $2.5D$ (closed) triangulations of the given discrete data set $V$. We generalise the operation of diagonal flip for $2.5D$ triangulations by omitting the usual restriction that the flip operation should not produce a self–intersecting triangulation. We denote this flip operation by *EDF* (*extended diagonal flip*). Among all possible $2.5D$ triangulations with the vertex set $V$ we first single out those that are topologically equivalent to the $2D$ sphere. We show that any two such $2.5D$ triangulations (if $V$ is situated in general position), are equivalent under EDF, *i.e.*, they can be transformed into each other via a finite sequence of EDF.

## Keywords

Triangulations, diagonal flips, polyhedral surfaces

## 1. INTRODUCTION

This paper contributes to the study on diagonal flips in triangulations. We study diagonal flips in non–planar triangulations of discrete data.

Much research on this subject is carried out in the field of computational geometry as well as in topological graph theory. Whereas in topological graph theory an extended study has been done on diagonals flips in triangulations on surfaces of various genera, in computational geometry investigations are limited mostly to diagonal flips in triangulations of point sets or of polygons in the plane.

A *triangulation* $T$ is a partition of a geometric domain into simplices that meet only at shared faces.

*The research of the author is partially supported by the NWO (STW) (Dutch Organization for Scientific Research), project No. TWI4816

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

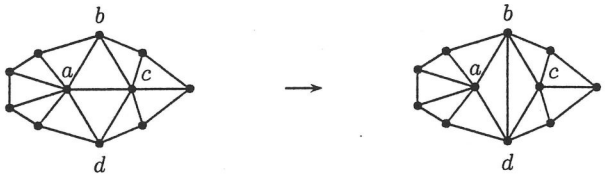



**Figure 1: A diagonal flip in a triangulation**

In topological graph theory therefore under a *triangulation* on a surface one understands a simple graph $G$ embedded on the surface so that each face is triangular and any two faces share at most one edge [8]. A *diagonal flip* is a local transformation (deformation) of a triangulation $G$ which replaces a diagonal edge $ac$ with the other $bd$ in a quadrilateral region obtained from two triangular faces $abc$ and $acd$ and which preserves the simplicity of the graph (as in Fig. 1).

Let us note that positions of vertices can be changed freely and edges can be bent. One of the research directions concerns the study of *equivalence* (connectedness) of one or another class of triangulations under diagonal flips, *i.e.*, if any two triangulations that belong to the same class can be transformed to each other via sequence of diagonal flips.

The starting point in this research direction belongs to Wagner and is known as Wagner's theorem [9]:

THEOREM 1. *Any two triangulations on the sphere with the same number of vertices are equivalent to each other under diagonal flips, up to homeomorphism.*

The concept of homeomorphism between two graphs can be found in ([8]):

There is a simple algorithm to transform any triangulation of $n$ vertices to the standard form $\Delta_n$, which is depicted in Fig. 2

All standard forms are homeomorphic to each other.

In the field of computational geometry a geometric domain can be a point set, a polygon or a polyhedron. In the pla-

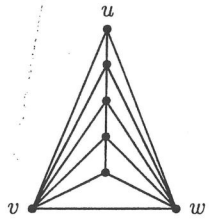



**Figure 2: The standard form of spherical triangulations $\Delta_n$**

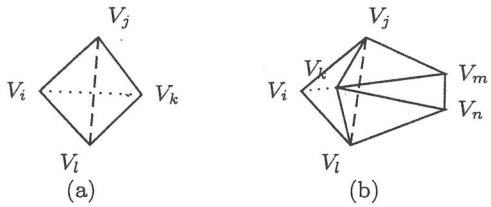



**Figure 3: Swapping a diagonal in a quadrilateral: (a) Convex quadrilateral.(b) Non-convex quadrilateral. In both cases the dotted edge swapped to the dashed one.**

nar case a triangulation represents a collection of triangles. The difference with respect to topological graph theory is that the points (vertices) have fixed positions and the edges are straight line segments. Therefore, we must distinguish between *convex* and *concave* quadrilaterals. Consequently, not all diagonal flips, which are possible topologically, might be allowed. We call diagonal flips which occur in topological graph theory *abstract diagonal flips*. In the field of computational geometry the diagonal flip (called then an exchange) was first introduced by Lawson [6]. The operation consists of swapping (flipping) a diagonal of the *convex* quadrilateral formed by two adjacent triangles, to the other diagonal, thus replacing one edge by a new one and obtaining another triangulation of the given data. He showed also that any two triangulation s $T_1$ and $T_2$ of point set $V$ in domain $Q$ (with the convex hull of $V$as the boundary) are connected via a sequence of diagonal flips.

A diagonal flip is an example of a *local transformation*, *i.e.*, an operation that allows us to produce a new object in a class of given objects from previously generated objects by means of some small change [5]. In Surface Reconstruction the diagonal flip, as a local transformation, is often used to obtain an optimal triangulation [3],[2].

Let us note that if we have a given discrete point set in $3D$, we can, in general, construct triangulated surfaces of different genera with the given points as the vertices. In computational geometry such triangulated surfaces are called $2.5D$ triangulations. All $2.5D$ triangulations $ST$ of the data set $V$ can be separated into mutually disjoint classes of triangulations $ST_1, ...ST_l$ in such a way that all triangulations of the same class have the same genus and orientability (*i.e.*, topologically equivalent to the $2D$ sphere, the torus, and so on). It seems logical to assume that these characteristics should be preserved under the edge flip. Let us also note that every triangulation represents in general a new polyhedral surface (if the data are situated in general position). Every flip will produce a new triangulation and therefore a new polyhedral surface.

Traditionally the flip operation is only considered if it does not yield a self–intersection. Indeed, this is a logical restriction on this operation for plane triangulations, since a self–intersection results in producing no triangulation. If the quadrilateral is not convex, the second 'diagonal' will lie outside the 'body' of the quadrilateral and will intersect some other triangles or coincide with the already existing edge, and edge swapping will not produce a triangulation (see Fig. 3)

Unfortunately, the above–mentioned restriction in the case of $2.5D$ triangulations excludes the use of a traditional flip operation as a local transformation procedure. It has been shown that given two traingulations $ST_1$ and $ST_2$ with the same data set $V$, it is possible that there is no sequence of (traditional) flips transforming $ST_1$ into $ST_2$ (*i.e.*, at a certain step a triangulation with self–intersections is produced [1]).

## 2. MAIN RESULTS

The main contribution of the paper is a generalisation of the diagonal flip operation. We call this new operation the *extended diagonal flip*, or simply the *EDF*. A $2.5D$–triangulation (a triangulated surface) is a collection of triangles, which are, in contrast to the $2D$ case, situated not in plane but in space. The EDF can produce self–intersections, but a valid triangulation can still be determined.

The operation of *EDF* preserves the given genus and orientability of a triangulated surface. In this paper we restict ourselves to triangulations of genus 0.

Let us make the concept of a triangulation more accurate. The following definition is standard:

*Definition 1.* A triangulation $T$ is a collection of triangles, that satisfies the following properties:

1. Two triangles are either disjoint, or have one vertex in common, or have two vertices and consequently the entire edge joining them in common.
2. $T$ is connected.

In the case of compact surfaces a triangulation $T$ consists of a finite number of triangles, and we can conclude that the two following conditions are valid [7]:

1. Each edge is an edge of exactly two triangles.
2. For every vertex $V$ of a triangulation $T$, we may arrange the set of all triangles with $V$ as a vertex in cyclic order, $T_0, T_1, ..., T_{n-1}, T_n = T_0$, such that $T_i$ and $T_{i+1}$ have an edge in common for $0 \leq i \leq n-1$.

The last condition means, that our triangulated surface is a manifold, *i.e.*, the neighbourhood of every point, as well

as a vertex, is topologically the same as the open unit ball in $\mathbf{R^2}$. The first condition of Definition 1 excludes any intersection. Actually, the above–mentioned definition is the definition of an *abstract triangulation* of an abstract surface [4]. 'Real' surfaces can have self–intersections and/or have singularities, in which a surface is not a manifold. Therefore, we should distinguish between the abstract topological representation of a surface $S$ (as a 2-dimensional *manifold*) and its realisation in 3-space (as an image of this 'canonical' manifold).

**Quadrilaterals in the space.** In space, if the data are situated in general position, any four vertices form a tetrahedron. We would like to stick to a surface ('plane') terminology, so we prefer to consider two adjacent triangles, formed by edges connecting the vertices, instead of a tetrahedron. The figure, built of two adjacent triangles, will be called a *spatial quadrilateral*. The edges of two triangles except the common edge form a closed polygonal line. We call it the *boundary* of the quadrilateral. If we keep the boundary edges fixed, then there are only two possible spatial quadrilaterals for every four vertices. If a triangulation is given then the boundary of a quadrilateral is always fixed. The common side of two adjacent triangles in a spatial quadrilateral will be called a *(spatial) diagonal*. The orientation of the triangles forming the spatial quadrilateral should be coherent. As we deal with oriented surfaces, we can determine two directions of normals to a surface (and therefore, to each triangle of our oriented surface). Usually, the direction of outwards pointing normals is said to be positive. We can say that an edge (diagonal) is *reflex*, if two lines, determined by the unit normals to the adjacent triangles, sharing this edge as the common side, intersect each other in the positive direction, otherwise, the edge is called *convex*. The edge is *flat* if the normals are parallel.

Then flipping in space means the exchange of a convex edge to a reflex edge and vice versa.

At each step we work only with two triangles, which together form a spatial quadrilateral. As the data are situated in general position and if we connect two vertices of the quadrilateral that are not the end–points of the common edge, the new edge will lie outside the surface of the quadrilateral and we can determine two new triangles. We will call a spatial quadrilateral *convex*, if its development on the plane is a convex quadrilateral; otherwise, we call it *reflex*. Therefore, we have four cases: convex quadrilateral with a convex diagonal, convex quadrilateral with a reflex diagonal, reflex quadrilateral with a convex diagonal, reflex quadrilateral with a reflex diagonal. We can denote the above–mentioned quadrilaterals by **CC, CR, RC, RR.** See Fig. 4, where we present the swapping of the concave (reflex) diagonal in CR and RR quadrilaterals.

We can encounter a degenerated situation when a quadrilateral collapses into two glued together triangles, as any three vertices lie always in a plane. The EDF is also valid for such a degenerated situation, but we need to introduce the definition of a generalised 2.5D triangulation.

*Definition 2.* (EXTENDED DEFINITION OF A TRIANGULATION) A triangulation T (triangulated surface,

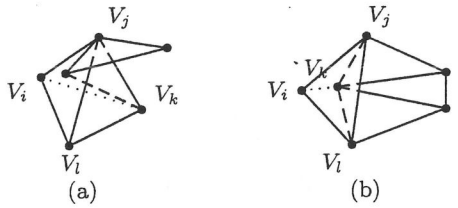



**Figure 4: Swapping diagonal $V_iV_k$ in a spatial quadrilateral: (a) CR quadrilateral (b) RR quadrilateral. Vertex $V_j$ on the left picture is a so-called pinch point.**

2.5$D$ triangulation) either corresponds to the properties of Definition 1 or, if not, then the following exceptions may occur:

1. For each triangle we can still single out three adjacent triangles, with which it shares a common edge. Two of three adjacent triangles may coincide.
2. Two triangles can have some points in common, besides vertices or edges. If these triangles are not adjacent then they can intersect each other along a line segment. Two adjacent triangles may have a triangular domain in common.

In other words we consider now not only 'pure' simplicial complexes and we allow some singularities. We call such triangulations generalised 2.5D triangulations, or shortly **2.5GD**. Let us note that the above definition is valid also if the data are situated not in general position.

THEOREM 2. *All generalised 2.5D triangulations of the data, which are in general position, and that are topologically equivalent to the 2D sphere are connected under the EDF.*

The proof is based on the following procedure: we put our 2.5D triangulation in one–to–one correspondence to some abstract triangulation (topological graph) on the 2D sphere and then by adapting in a proper way Wagner's theorem, we show that our generalised 2.5D triangulations are connected under the EDF.

COROLLARY 1. *Among 2.5GD (closed) triangulations of the given data there are always some with self–intersections.*

We refer the interested reader to the full version of the paper.

## 3. REFERENCES


[1] O. Aichholzer, L. Alboul, and F. Hurtado. On flips in polyhedral surfaces. *Report MA2-IR-00-00036, Universitat polytècnica de Catalunya. (Accepted (conditionally) to Int. J. of Found. of Comp. Sc.)*, 2000.

[2] L. Alboul, G. Kloosterman, C. Traas, and R. van Damme. Best data–dependent triangulations. *J. of Comp. and Applied Math.*, 119:1–12, 2000.

[3] N. Dyn, D. Levin, and A. Rippa. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1990.

[4] P. Giblin. *Graphs, Surfaces and Homology.* Chapman and Hall, London, 1977.

[5] C. Hernando, F. Hurtado, and M. Houle. On local transformations of simple polygons. *Lecture Notes in Computer Science, Vol. 1858 (Proc. 6th International Computing and Combinatorics Conference (COCOON'00) Sydney)*, pages 54–63, 2000.

[6] C. Lawson. Transforming triangulations. *Discrete mathematics*, 3:365–372, 1972.

[7] W. Massey. *A Basic Course in Algebraic Topology.* Springer-Verlag, Berlin Heidelberg New York, 1991.

[8] S. Negami. Diagonal flips of triangulations on surfaces. *Yokohama Math.Journal (special issue)*, 47:1–40, 1999.

[9] K. Wagner. Bemekungen zum vierfarbenproblem. *J. der Deut. Math.*, 46(1):26–32, 1936.

# Computing the symmetries of non-convex polyhedral objects in 3-space

(Extended Abstract)

Peter Braß* Christian Knauer

Institut für Informatik, Freie Universität Berlin
Takustraße 9, D–14195 Berlin, Germany

{brass,knauer}@inf.fu-berlin.de

## ABSTRACT

This paper presents an algorithm that computes the symmetries of a polyhedral object of size $n$ in 3-dimensional space in $O(n \log n)$ time. This improves the currently fastest algorithms for that problem.

## Keywords

Computational geometry, Polyhedral object, Symmetry detection.

## 1. INTRODUCTION

The task of identifying symmetries of three-dimensional objects arises naturally in many applications, e.g., in computer graphics, computer vision, computer aided manufacturing (CAM), and computer aided design (CAD), see [3] for examples. Often such objects are composed from 'simple' geometric entities. From an application point of view, convexity is usually not an acceptable recstriction. Therefore we use what we call *polyhedral objects* as an abstraction.

DEFINITION. *A* polyhedral object $P$ *(in 3-space) consists of a set of vertices* $V \subseteq \mathbb{R}^3$, *a set of edges* $E \subseteq \binom{V}{2}$, *and a set* $F \subseteq 2^V$ *of polygonal faces, where each face* $f \in F$ *is given as the counterclockwise ordered set of vertices from* $V$ *on* $f$. *Incidence between vertices, edges, and faces is defined in the obvious way. Moreover it is required that*

- *there are no isolated vertices,*
- *each edge is incident to* $O(1)$ *faces, and*
- *the graph of edge-face incidences is connected.*

*This research was supported by the Deutsche Forschungsgemeinschaft under grant no. BR 1465/5-2.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

The *size* of a polyhedral object $P$, denoted $|P|$ is the total number of vertices on all edges and faces (counting multiplicities), i.e.,

$$|P| = |V| + |E| + \sum_{f \in F} |f|.$$

Note that $|P|$ can be superlinear in $|V|$, the number of vertices. In the following $P$ will always be a polyhedral object with $|P| = n$.

We consider the *symmetry detection problem* for polyhedral objects in 3-space:

PROBLEM (SYMMETRY DETECTION).

*Given* *a polyhedral object* $P$ *of size* $n$,
*decide,* *whether* $P$ *has a symmetry.*

Here by a (non-trivial) symmetry of $P$ we mean a rigid motion $\sigma : \mathbb{R}^3 \to \mathbb{R}^3$ (different from the identity map) such that $P = \sigma(P)$, that induces an automorphism on the vertex-edge-face-incidence graph of $P$. Usually we also would like to give a suitable representation of all the symmetries of $P$. Note that these symmetries form a group which will be denoted by $\Sigma(P)$.

The symmetry detection problem in $\mathbb{R}^3$ is well understood when $P$ is a *convex* polyhedron. Alt et al. [1] and Atallah [2] independently developed optimal $O(n \log n)$ algorithms for that case. For the more general case of polyhedral objects, Jiang et al. [4] gave a $O(n^2 \log n)$ time solution which we improve to $O(n \log n)$ below. The main result of this paper is summarized in the following

THEOREM. *The symmetry group of a polyhedral object in* $\mathbb{R}^3$ *of size* $n$ *can be computed in* $O(n \log n)$ *deterministic time.*

## 2. THE ALGORITHM

The basic idea of the algorithm is simple: We first compute a supergroup of $\Sigma(P)$ with the algorithm of Alt et al., and then determine the correct subgroup. Since the finite symmetry groups of subsets of $\mathbb{R}^3$ are fully classified (see Hessel's Theorem below) and since they are essentially determined

by a rotation of high order, this can be done by inspecting only few cyclic subgroups.

## 2.1 Preliminaries

We make use of the following Theorem which states that there are only finitely many types of symmetry groups of subsets of $\mathbb{R}^3$:

THEOREM (HESSEL'S THEOREM, [6, 5]). *Any finite symmetry group of a subset of $\mathbb{R}^3$ is one of the following groups:*

(a) *The rotation groups $T$, $O$, $I$ of the tetrahedron, octahedron, and icosahedron, respectively.*

(b) *The cyclic groups $C_n$ $(n \geq 1)$, and the dihedral groups $D_n$ $(n \geq 2)$.*

(c) *The groups $T'$, $O'$, $I'$, $C'_n$ $(n \geq 1)$, $D'_n$ $(n \geq 2)$, where $G'$ is the group generated by $G$ and a reflection at some point (inversion).*

(d) *The groups $OT$, $C_{2n}C_n$, $D_{2n}D_n$, $D_nC_n$ $(n \geq 2)$ where $GH$ means $H \cup (G \setminus H) \circ i$, where $i$ is an inversion.*

We will also exploit the following two facts which easily follow from the previous characterization:

PROPOSITION 1.

1. *None of the groups $T$, $O$, $I$, $T'$, $O'$, $I'$, $OT$ occurs as a subgroup of any group with more than 120 elements.*
2. *Each of the groups $G$ with more than 120 elements contains a unique cyclic subgroup of maximal order, denoted $\Gamma(G)$.*

It is clear that a symmetry of the underlying point set $V$ is not necessarily a symmetry of $P$. However, since a symmetry of $P$ maps vertices to vertices, the symmetry group of $P$ is a subgroup of the symmetry group of $V$. Moreover the symmetry group of $V$ is also a subgroup of the symmetry group of $\mathrm{ch}(V)$, the convex hull of $V$. This yields

PROPOSITION 2.

$$\Sigma(P) \leq \Sigma(\mathrm{ch}(V)).$$

## 2.2 Overall description

The algorithm consists of two steps.

In a first step we compute $\mathrm{ch}(V)$ in $O(n \log n)$ time and then run the algorithm of Alt et al. [1] to determine $\Sigma(\mathrm{ch}(V))$, the symmetry group of $\mathrm{ch}(V)$. If $\Sigma(\mathrm{ch}(V))$ is not one of the groups derived from the platonic solids the algorithm also gives $k+1$ rotation axes (for some $k \geq 0$), $k$ of which lie in a plane and the $(k+1)$–st is perpendicular to that plane. This last rotation axis is also the rotation axis of $\Gamma(\mathrm{ch}(V))$, the unique cyclic subgroup of maximal order in $\Sigma(\mathrm{ch}(V))$.

In a next step we try to determine the correct subgroup of $\Sigma(\mathrm{ch}(V))$. This can be done by testing for $O(\log n)$ candidate mappings if they constitute symmetries of $P$:

LEMMA 1. *In order to determine $\Sigma(P)$ given $\Sigma(\mathrm{ch}(V))$ it is sufficient to verify for $\log n + O(1)$ mappings $\sigma : \mathbb{R}^3 \to \mathbb{R}^3$ whether $\sigma$ is a symmetry of $P$.*

PROOF. If $\Sigma(\mathrm{ch}(V))$ has at most 120 elements we can determine the correct subgroup by brute force by performing $O(1)$ tests.

Otherwise we proceed as follows: First we determine $\Gamma(\mathrm{ch}(V))$, the unique cyclic subgroup of maximal order in $\Sigma(\mathrm{ch}(V))$ and its corresponding rotation axis $\ell$ (actually we get that for free from the algorithm of Alt et al.). Let $m$ denote the order of that group. As easily follows from Hessel's Theorem we have that $m \leq |\mathrm{ch}(V)| \leq n$.

Next we determine $\Gamma(P)$, which, by Proposition 2 is a subgroup of $\Gamma(\mathrm{ch}(V))$ and is therefore cyclic, too. Of course its corresponding rotation axis is $\ell$. Its order is one of the divisors of the group order $m$. Instead of testing all divisors (of which there can be up to $2^{\log m/\log\log m}$ many) we proceed as follows: for each maximal prime power $p^\alpha$ that divides $m$ we check the $\alpha$ rotations around $\ell$ corresponding to $p, p^2, p^3, \ldots, p^\alpha$ and determine the largest exponent $\alpha' \leq \alpha$ such that the rotation of order $p^{\alpha'}$ constitutes a symmetry. Finally we multiply all these maximal prime powers $p^{\alpha'}$ to get the right order of $\Gamma(P)$. If $m = p_1^{\alpha_1} \cdot \cdots \cdot p_k^{\alpha_k}$ with $\alpha_i \geq 1$ is the prime decomposition of $m$, then we have to test $\alpha_1 + \cdots + \alpha_k$ rotations with this approach. Since

$$n \geq m = p_1^{\alpha_1} \cdot \cdots \cdot p_k^{\alpha_k} > 2^{\alpha_1} \cdot \cdots \cdot 2^{\alpha_k} = 2^{\alpha_1 + \cdots + \alpha_k},$$

it follows that $\alpha_1 + \cdots + \alpha_k \leq \log n$.

Once we have computed $\Gamma(P)$ we need to determine the 'real' symmetry group $\Sigma(P)$. But according to Hessel's Theorem there is only a finite number of possibilities which we can identify by testing $O(1)$ additional candidate symmetries. □

To finish the proof of our main result, we have to specify how a candidate symmetry is verified. With some amount of preprocessing, this can actually be achieved in linear time per candidate:

LEMMA 2. *After $O(n \log n)$ preprocessing we can check for a mapping $\sigma : \mathbb{R}^3 \to \mathbb{R}^3$ in $O(n)$ time whether $\sigma$ is a symmetry of $P$.*

PROOF. We store all vertices, edges and faces of $P$ in a suitable search tree. Then we map each vertex, edge, or face $x$ of $P$ to $\sigma(x)$ and check whether $\sigma(x)$ does exist. There is a minor problem with this approach in that it is not clear how to store faces with many vertices. We can remedy this situation as follows: store an edge $\{a, b\}$ as the two pairs $(a, b)$ and $(b, a)$, and a face $\{a_1, \ldots, a_l\}$ as the $l$ triples $(a_1, a_2, a_3)$, $(a_2, a_3, a_4)$, $(a_3, a_4, a_5), \ldots, (a_{l-1}, a_l, a_1)$, $(a_l, a_1, a_2)$. The number of objects stored in the data structure is still $O(n)$.

In $O(n \log n)$ time we can build the corresponding data structure (which is essentially a balanced binary search tree) and the check whether $\sigma(x)$ does exist requires $O(\log n)$

time. So we can verify a candidate symmetry in $O(n \log n)$ time which yields a total runtime of $O(n \log^2 n)$.

We can improve upon this approach by exploiting the adjacency information associated with $P$. To this end we transfer this information into the search structure in the following way: each object in the search tree gets $O(1)$ neighbors:

- an edge is linked to the triples corresponding to the adjacent faces, and
- a triple of a face is linked to the adjacent edges.

Note that vertices are not explicitly handled, for if each vertex is adjacent to an edge (as it is required for a polyhedral object) this is not necessary.

Obviously a symmetry $\sigma$ of $P$ has to preserve the adjacency information. So the problem is to decide, given two graphs $G_1$ and $G_2$ of bounded degree, whether they are isomorphic (and to construct an isomorphism in case it exists). Observe that $G_1$ is given explicitly, whereas $G_2$ is actually the 'image' of $G_1$ under $\sigma$. We proceed as follows: We pick an arbitrary vertex $v_1 \in G_1$ (that corresponds to an edge of $P$), compute the image $\sigma(v_1)$, and locate it in the search tree. We can extend this partial isomorphism (as long as it is possible) by conducting a depth first search from $v_1$ in both graphs simultaneously. Since the graphs have bounded degree there are only $O(1)$ possibilities for the next image (i.e., the next vertex in $G_2$), and we can check for each of them in $O(1)$ if they are induced by $\sigma$.

So since each edge has only $O(1)$ incident faces, the edge-face incidence graph is connected, and each vertex is adjacent to at least one edge, we can check whether $\sigma$ constitutes a symmetry in $O(n)$ time after $O(n \log n)$ preprocessing which yields a total runtime of $O(n \log n)$. □

This concludes the proof of our main result.

## 3. REFERENCES


[1] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3:237–256, 1988.

[2] M. J. Atallah. On symmetry detection. *IEEE Trans. Comput.*, C-34:663–666, 1985.

[3] X. Jiang and H. Bunke. Symmetries of polyhedra: Detection and applications. Technischer bericht, Institut für Informatik, Universität Bern, Schweiz, 1994.

[4] X. Jiang, K. Yu, and H. Bunke. Detection of rotational and involutional symmetries and congruity of polyhedra. *Visual Comput.*, 12(4):193–201, 1996.

[5] G. E. Martin. *Transformation Geometry: An Introduction to Symmetry.* Springer-Verlag, 1982.

[6] P. B. Yale. *Geometry And Symmetry.* Dover Publications, Inc., 1968.

# Chromatic Variants of the Erdös-Szekeres Theorem on Points in Convex Position *

[Extended Abstract]


Olivier Devillers
INRIA
BP93
06902 Sophia-Antipolis
France
Olivier.Devillers@sophia.inria.fr

Ferran Hurtado†
Dept. Matemàtica Aplicada II,
Univ. Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain
hurtado@ma2.upc.es

Carlos Seara‡
Dept. Matemàtica Aplicada II,
Univ. Politècnica de Catalunya
Pau Gargallo 5
08028 Barcelona, Spain
seara@ma2.upc.es



## ABSTRACT

Let $S$ be a point set in the plane in general position, such that its elements are partitioned into $k$ classes or *colors*. In this paper we study several variants on problems related to the Erdős-Szekeres Theorem about subsets of $S$ in convex position, when additional chromatic constraints are considered.


## 1. INTRODUCTION AND PRELIMINARY RESULTS

The following result is commonly called *the Erdős-Szekeres Theorem*:

THEOREM 1.1. *[9] For every natural number $m$ there exists a number $n(m)$ such that any $n$-point set $S$ in the plane in general position with $n \geq n(m)$ contains an $m$-subset of points in convex position.*

This problem has been attracting the attention of many researchers, both because its beauty and elementary statement, and because finding the exact value of $n(m)$ turns out to be a very challenging problem. The reader is referred to the survey paper [18] for a history of the problem, a description of many variants, and a wide list of references. The best currently known bounds are


*Most of this work was made possible by the *Picasso French-Spanish collaboration program* and the *Acción Integrada Francia-España HF99-112.*

†Partially supported by *Proyecto DGES-MEC PB98-0933* and *Gen.Cat. SGR1999-0356.*

‡Partially supported by *Proyecto DGES-MEC PB98-0933* and *Gen.Cat. SGR1999-0356.*



*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

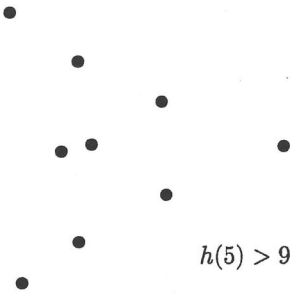




**Figure 1: A set of 9 points with no 5-hole.**

$$2^{m-2} \leq n(m) \leq \binom{2m-5}{m-2} + 2,$$

where the lower bound was essentially proved by Erdős and Szekeres in their first papers and the upper bound is due to Tóth and Valtr [26].

Let $A$ be a point set in the plane in general position. An $m$-point subset $B \subset A$ in convex position is called an *$m$-hole* in $A$ if the convex hull $conv(B)$ is a polygon whose interior does not contain any point of $A$.

In 1978 Erdős [8] raised the following problem: is there a number $h(m)$, for every natural $m$, such that any $n$-point set $S$ in the plane in general position with $n \geq h(m)$ contains an $m$-hole?

Obviously $h(3) = 3$ and it is easy to see that $h(4) = 5$ and that $h(5) \geq 10$ (see Figure 1). Harborth [10] proved in 1978 that $h(5) = 10$, and in 1983 Horton [11] showed that $h(m)$ does not exist for $m \geq 7$ by constructing arbitrarily large sets without a 7-hole. The existence of $h(6)$ is a problem that still remains open.

The seminal paper by Erdős and Szekeres already mentioned the generalization of their original problem to higher dimensions, but even in the plane many variants have been considered, we mention next some examples. Bistriczky and Fejes Tóth [5] proved a generalization replacing points with convex bodies. Bialostocki *et al* [4], Caro [6] and Károlyi *et al* [15] gave results on the conjecture from [4] that for any given integers $m$ and $q$ any set large enough contains

a $n$-set for which the number of interior points is divisible by $q$. Several authors have studied the number of subsets in convex position of a given size that a sufficiently large point set can have [3, 25, 22, 23, 17, 2, 28, 7]. Let us finally mention the papers by Ambarcumjan [1], Károlyi [13], Hosono and Urabe [12] and Urabe [27], where several issues on partitioning a point set into subsets in convex position are considered.

Let $S = S_1 \dot{\cup} \cdots \dot{\cup} S_k$ be a partition of a planar point set $S$ in general position in the plane; we refer to $S_i$ as the set of points of *color* $i$. A subset $T \subset S$ is called *monochromatic* if all its points have the same color, and *polychromatic* otherwise. The term *heterochromatic* is used for the special case in which every element in $T$ has a different color.

In this paper we consider the following collection of problems. Given an integer $m$ and a set $S$ as above, possibly with additional requirements for $n = |S|$ to be large enough, can we find an $m$-hole of $S$ falling into one of the three described chromatic classes? Or an $m$-subset in convex position?

The original motivation for us to study these problems came from a different area. A finite set $\Gamma$ of curves in the plane is a *separator* for the sets $S_1, \ldots, S_k$ if every connected component in $\mathbb{R}^2 - \Gamma$ contains objects only from some $S_i$. We also say that each connected component is *monochromatic*. A thorough study of the subject is developed in [24].

When we have two sets, say the *red* points and the *blue* points, another way to approach their separability is to look for triangulations in which as many edges as possible (or as many triangles as possible) are monochromatic, which somehow contributes to isolate the two populations.

As a consequence of the above motivation, in this paper we study the conditions for the existence of some configuration, and also consider how many *compatible* such configurations can we guarantee, where compatibility stands for having disjoint relative interiors. For example, if the configuration is a monochromatic edge, we also try to find how many monochromatic edges can we guarantee without producing any crossing; the compatibility allows the edges to be completed to a triangulation.

In the sequel we will study the numbers $n_M(m,k)$, the minimal number of points colored with $k$ colors to ensure the existence of one monochromatic $m$-subset in convex position, and $MC(n,m,k)$, the minimal number of compatible monochromatic $m$-holes in a set of $n$ points colored with $k$ colors. $n_H(m,k)$, $n_P(m,k)$ $HC(n,m,k)$ and $PC(n,m,k)$ are similarly defined in the heterochromatic and polychromatic cases. A related (yet quite different) problem is considered in [19]. More in the spirit of our problems, several results are described in [20, 21, 14, 16] but looking for configurations like cycles or paths when *edges* are colored.

## 2. SUBSETS IN CONVEX POSITION

It is natural to start by considering subsets in convex position without the additional constraint of requiring empty interiors. While the former case is quite direct, the latter one, studied in the following section, is much less simple.

THEOREM 2.1. $n_M(m,k) = k \cdot (n(m) - 1) + 1$.

THEOREM 2.2.

*If* $k \geq n(m)$ *then* $n_H(m,k) = n(m)$.

*If* $k < n(m)$ *then* $n_H(m,k) = \infty$.

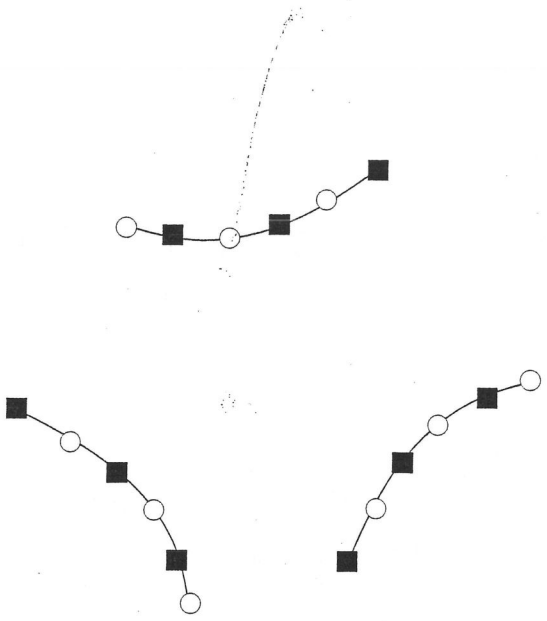


**Figure 2: A set of 18 points with no monochromatic 4-hole.**

The more interesting situation arises in the polychromatic case

THEOREM 2.3.

*If* $k \geq n(m)$ *then* $n_P(m,k) = n(m)$.

*If* $k < n(m-1)$ *then* $n_P(m,k) = \infty$.

There is a gap remaining for $n(m-1) \leq k < n(m)$. We have examples showing that $n_P(5, n(4)) = n_P(5,5) = \infty$ and that $n_P(6, n(5)) = n_P(6,9) = \infty$.

## 3. EMPTY CONVEX SUBSETS

### 3.1 Monochromatic holes

THEOREM 3.1. *The minimum numbers of compatible monochromatic $m$-holes, $MC(n,m,k)$ are as follows:*

| $m \setminus k$ | 2 | 3 | |
|---|---|---|---|
| 2 | $\longleftarrow$ | $2\lceil \frac{n}{k} \rceil - 3$ | $\longrightarrow$ |
| 3 | $\lceil \frac{n-8}{4} \rceil$ | 0 | $\longrightarrow$ |
| 4 | ? | 0 | $\longrightarrow$ |
| 5 | 0 | 0 | $\longrightarrow$ |
| 6 | 0 | 0 | $\longrightarrow$ |
| 7 | 0 | 0 | $\longrightarrow$ |

The problem of finding $MC(n,4,2)$ remains open, however we can remark that a bichromatic set with no monochromatic 4-hole cannot contain any 7-hole, otherwise such heptagon would have at least 4 points of the same color and they would form a monochromatic 4-hole. As a consequence, any example showing $MC(n,4,2) = 0$ must be a Horton set or an equivalent construction.

A construction with 18 points which contains no monochromatic 4-hole is shown on Figure 2 proving that $MC(n,4,2) = 0$ for $n \leq 18$.

CONJECTURE 3.1. *For $n$ large enough, $MC(n,4,2) > 0$. In other words, every large bichromatic point set contains some monochromatic 4-hole.*

## 3.2 Heterochromatic holes

THEOREM 3.2. *The minimum numbers of compatible heterochromatic m-holes, $MH(n,m,k)$ are the following:*

| $m \setminus k$ | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|
| 2 | $\longleftarrow$ | $n+k-3$ | $\longrightarrow$ | | | | |
| 3 | | $\longleftarrow$ | $k-2$ | $\longrightarrow$ | | | |
| 4 | | | 0 | $(\forall n \geq 2k-3)$ | | | |
| 5 | | | | 0 | $(\forall n \geq 2k-4)$ | | |
| 6 | | | | | 0 | $(\forall n \geq 2k-5)$ | |
| 7 | | | | | | 0 | $\longrightarrow$ |

For $\lfloor \frac{h(m)(k-1)+3}{h(m)-2} \rfloor \leq n < 2k-m+1$ (with $m \geq 4$) there is a gap in our results in which we do not know whether $HC(n,m,k)$ is non-zero. $HC(4,4,4)$ and $HC(6,4,5)$ are shown to be zero by example.

## 3.3 Polychromatic holes

THEOREM 3.3. *The minimum numbers of compatible polychromatic m-holes, $MP(n,m,k)$, are the following:*

| $m \setminus k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | $\longleftarrow$ | $n+k-3$ | $\longrightarrow$ | | | | | | | | |
| 3 | $\longleftarrow$ | $n-2$ | $\longrightarrow$ | | | | | | | | |
| 4 | 0 | 1 | 2 | $\geq \lfloor \frac{k-2}{3} \rfloor$ | | | | | | | |
| 5 | 0 | 0 | 0 | 0 | ? | ? | ? | ? | ? | ? | |
| 6 | | | | | | | | $\longleftarrow$ | 0 | ? | |
| 7 | 0 | $\longrightarrow$ | | | | | | | | | |

Notice that for $m \geq 7$ we can not expect polychromatic holes since there are sets with no 7-holes at all, and that even without colors the existence of 6-holes is yet an open problem.

# 4. CONCLUSION

Several results on a generalization of the Erdős-Szekeres Theorem to colored sets of points have been presented in this paper. The chromatic version with $k$ colors differs significantly from the non-chromatic version since for fixed $m$ and $k$ and $n$ large at will, it is possible to construct point sets with no heterochromatic or polychromatic subset of size $m$ in convex position.

The more interesting results arise for the problem of the existence of $m$-holes for $3 \leq m \leq 6$. We have succeeded in proving some results on the existence or non-existence of $m$-holes, but some intriguing problems remain open. Among them, our conjecture on the existence of a monochromatic 4-hole in any large enough bichromatic point set is maybe the most challenging one.

# 5. REFERENCES


[1] R. V. Ambarcumjan. Convex subclusters of point clusters in the plane. *Dokl. Acad. Nauk Armjan. SSR*, 43:12–14, 1966.

[2] I. Bárány and Z. Furedi. Empty simplices in Euclidean space. *Can. Math. Bull.*, 30:436–445, 1987.

[3] I. Bárány and P. Valtr. A positive fraction Erdős-Szekeres theorem. *Discrete Comput. Geom.*, 19:335–342, 1998.

[4] P. D. A. Bialostocki and B. Voxman. Some notes on the Erdős-Szekeres theorem. *Discr. Math.*, 91:231–238, 1991.

[5] T. Bisztricky and G. F. Tóth. A generalization of the Erdős-Szekeres theorem. *J, reine Angew. Math.*, 395:167–170, 1989.

[6] Y. Caro. On the generalized Erdős-Szekeres conjecture. *Discr. Math.*, 160:229–233, 1996.

[7] A. Dumitrescu. Planar point sets with few empty convex polygons. *Studia Sci. Math. Hungar*, 36, 2000.

[8] P. Erdős. Some more problems on elementary geometry. *Austral. Math. Soc. Gaz.*, 5:52–54, 1978.

[9] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

[10] H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elem. Math.*, 33:116–118, 1979.

[11] J. D. Horton. Sets with no empty convex 7-gons. *Canad. Math. Bull.*, 26:482–484, 1983.

[12] K. Hosono and M. Urabe. On the number of disjoint convex quadrilaterals for a given point set in the plane. *Comp. Geom. Theory Appl.*, 20:97–104, 2001.

[13] G. Károlyi. Ramsey-remainder for convex sets and the Erdős-Szekeres theorem. *Discr. Appl. Math*, 109:163–175, 2001.

[14] G. Károlyi, J. Pach, and G. Tóth. Ramsey-type results for geometric graphs, i. *Discrete Comput. Geom.*, 18:247–255, 1997.

[15] G. Károlyi, J. Pach, and G. Tóth. A modular version of the Erdős-Szekeres theorem. *Studia Mathematica Hungarica*, to appear.

[16] G. Károlyi, J. Pach, G. Tóth, and P. Valtr. Ramsey-type results for geometric graphs, ii. *Discrete Comput. Geom.*, 20:375–388, 1998.

[17] M. Katachalski and A. Meir. On empty triangles determined by points in the plane. *Acta Math. Hungar.*, 51:323–328, 1988.

[18] W. Morris and V. Soltan. The Erdős-Szekeres problem on points in convex position - a survey. *Bull. Amer. Math. Soc.*, 37:437–458, 2000.

[19] J. Nešetřil and P. Valtr. A Ramsey-type theorem in the plane. *Combinatorics, Probability and Computing*, 3:127–135, 1994.

[20] J. Nešetřil and P. Valtr. A Ramsey property of order types. *J. Combin. Theory Ser. A*, 81:88–107, 1998.

[21] J. Nešetřil, J. Solymosi and P. Valtr. Induced monochromatic configurations. In *Contemporary trends in Discrete Math.*, DIMACS Series 49:219–228, 1999.

[22] M. J. Nielsen. Transverse matchings of a finite planar set, 1995. University of Idaho, Moscow.

[23] G. B. Purdy. The minimum number of empty triangles. *AMS Abstracts*, 3:318, 1982.

[24] C. Seara. *On the separability of point sets.* Ph. D. thesis, UPC, Barcelona, in preparation.

[25] J. Solymosi. *Combinatorial problems in finite Ramsey theory.* Master's thesis, Eötvös Univ., Budapest, 1988.

[26] G. Tóth and P. Valtr. Note on the Erdős-Szekeres theorem. *Discrete Comput. Geom.*, 19:457–459, 1998.

[27] M. Urabe. On a partition into convex polygons. *Discrete Appl. Math.*, 64:179–191, 1996.

[28] P. Valtr. On the minimum number of empty polygons in planar point sets. *Studia Sci. Math. Hungar.*, 30:155–163, 1995.

# On the Crossing Number of Complete Graphs (Extended Abstract)

Oswin Aichholzer* Franz Aurenhammer Hannes Krasser†
Institute for Theoretical Computer Science
Graz University of Technology
Graz, Austria
{oaich,auren,hkrasser}@igi.tu-graz.ac.at

## ABSTRACT

We determine $\overline{cr}(K_{11}) = 102$ and $\overline{cr}(K_{12}) = 153$. Moreover, we give improved upper and lower bounds on the asymptotic saving of crossings when drawing $K_n$ optimally.

## Categories and Subject Descriptors

G.2.1 [**Mathematics of Computing**]: Discrete Mathematics—*Combinatorics, Counting problems*; F.2.2 [**Nonnumerical Algorithms**]: Geometric problems and computations—*computations on discrete structures*

## Keywords

Rectilinear crossing number, complete graph, order type

## 1. INTRODUCTION

The *crossing number* of a graph $G$ is the least number of edge crossings that can be attained when drawing $G$ in the plane. *Drawing* may be interpreted in several ways leading to different concepts of crossing number; Pach and Tóth [13] discuss this issue in detail. Crossing number problems have a quite long history in the graph theory community (see, e.g., Tutte [19] and Erdös and Guy [6]) and have more recently been of interest to computer scientists (see Leighton [12] and Garey and Johnson [7]). For an overview paper, see Pach and Tóth [14].

In the present work, we are concerned with the *geometric* version of the crossing number problem, where the edges of the underlying graph are required to be straight line segments in the plane. Following Harary and Hill [10], we will use the notation *rectilinear crossing number*, $\overline{cr}(G)$, of a graph $G$. The vertices of $G$ are assumed to be in general position, meaning that no three vertices are allowed to be collinear in the drawing. Our specific interest is that of finding $\overline{cr}(K_n)$, the rectilinear crossing number of the complete graph on $n$ vertices.

Determining $\overline{cr}(K_n)$ is commonly agreed to be a difficult task. In fact, the asymptotic value, as $n$ tends to infinity, is unknown for *any* interpretation of the crossing number of $K_n$ considered in the literature; see Richter and Thomassen [15]. From an algorithmic point of view, deciding whether $\overline{cr}(G) \leq k$ for a given graph $G$ and parameter $k$ is NP-hard, as has been proved in Bienstock [3]. Only for very small $n$ the exact values of $\overline{cr}(K_n)$ are known. Whereas the instances $n \leq 9$ have been settled quite a time ago in Erdös and Guy [6], no progress has been made until in 2001 two groups of researchers (Brodsky et al. [4] and the authors [1], respectively) independently found $\overline{cr}(K_{10}) = 62$. In [4] the goal was reached by a purely combinatorial argument, while in [1] the result came as a byproduct of the exhaustive enumeration of all combinatorially inequivalent point sets (so-called *order types*) of size 10.

In fact, the order type data base in [1] gives some more information about $K_n$. For instance, beside the drawing constructed in [4] there is one (and only one) inequivalent drawing of $K_{10}$ which also achieves $\overline{cr}(K_{10}) = 62$. (The interested reader may consult Aichholzer and Krasser [2] for a collection of new results on the crossing properties of small geometric graphs.) For completeness, we include the following table from there.

| $n$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|
| $\overline{cr}(K_n)$ | 0 | 1 | 3 | 9 | 19 | 36 | 62 |
| $I_n$ | 1 | 1 | 1 | 3 | 2 | 10 | 2 |

**Table 1: $\overline{cr}(K_n)$ is attained by exactly $I_n$ drawings**

## 2. NEW RESULTS

Our results mentioned above came from a simple observation: point sets of the same order type yield equivalent drawings of the complete graph. To go further, we tailored the exhaustive search approach in [1] to generate inequivalent drawings of $K_n$ for larger $n$, and, luckily, were surprisingly successful.

In a first step, we generated all combinatorially inequivalent point sets for $n = 11$. In fact, 'only' a list of candidates

*Research supported by the Austrian Programme for Advanced Research and Technology
†Research supported by the FWF [Austrian Fonds zur Förderung der Wissenschaftlichen Forschung]

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

| $n$ | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| upper bound | 102 | 153 | 229 | 324 | 447 | 603 | 798 | 1030 | 1318 | 1658 |
| drawings | 374 | $\geq 1$ | $\geq 4272$ | $\geq 22$ | $\geq 2360$ | $\geq 17$ | $\geq 17532$ | $\geq 62$ | $\geq 3069$ | $\geq 54$ |
| lower bound | 102 | 153 | 221 | 310 | 423 | 564 | 738 | 949 | 1204 | 1505 |

**Table 2: New bounds on $\overline{cr}(K_n)$**

was generated, namely all the *abstract order types* that correspond to wiring diagrams; see Goodman and Pollack [8]. For each abstract order type (and there are 2 343 203 071 in the list) we computed the number of (abstract) crossings. As an important fact, both steps can be carried out in a purely combinatorial manner. This allows for tremendous savings in runtime, as well as for reliable computations. Only for those (few) abstract order types which gave the *smallest* number of crossings, we had to perform the more challenging geometrical task of finding point coordinates – if they exist at all.

They do exist, and we could restore them, which gave our first result; it completely resolves the situation for $n = 11$.

THEOREM 1. *The rectilinear crossing number of $K_{11}$ is 102, and this value is attained by exactly 374 inequivalent drawings. Each drawing shows only 3 extreme vertices.*

By carefully constructing larger drawings with low numbers of crossings, we obtained upper bounds superior (except for $n = 13$) to all previous ones. For $n \leq 20$ these bounds are listed in line 2 of Table 2. Line 3 comments on the number of inequivalent drawings that exist for the respective number of crossings. Some drawings for $n \leq 81$ are selected in Table 3. The interested reader is encouraged to visit our web site [20] where integer representations for the realizing point sets are provided.

| drawing | $D_{27}$ | $D_{36}$ | $D_{39}$ | $D_{40}$ | $D_{81}$ |
|---|---|---|---|---|---|
| crossings | 6186 | 21191 | 29737 | 33093 | 618616 |

**Table 3: Selected drawings**

Earlier experiments, by Thorpe and Harris [18], were based on randomized search and achieved drawings of $K_{12}$ and $K_{13}$ with 155 and 229 crossings, respectively. Their latter result thus competes with our best examples for $n = 13$.

All drawings reported in Tables 2 and 3 possess exactly three extreme vertices. This adds evidence to the commonly believed conjecture that the convex hull of every minimal rectilinear drawing of $K_n$ is a triangle.

Concerning lower bounds, Theorem 1 directly leads to an improvement by the following well-known argument, first used in Guy [9]. Each of the $n$ subgraphs $K_{n-1}$ of $K_n$ contains at least $\overline{cr}(K_{n-1})$ crossings, each being counted $n-4$ times because the 4 vertices determining the crossing appear in all but 4 such subgraphs. This implies the recurrence relation

$$\overline{cr}(K_n) \geq \lceil \frac{n}{n-4} \cdot \overline{cr}(K_{n-1}) \rceil. \qquad (1)$$

We further exploit that $\overline{cr}(K_n)$ and $\binom{n}{4}$, the maximum number of crossings in a $K_n$, are known to have the same parity for $n$ odd; see Erdös and Guy [6].

Plugging in the value $\overline{cr}(K_{11}) = 102$ yields the specific lower bounds listed in line 4 of Table 2 and – when driven to larger $n$ – the general lower bound in Theorem 3. Lower and upper bounds happen to match for $n = 12$, which allows us to break new ground again.

THEOREM 2. *The rectilinear crossing number of $K_{12}$ is 153. The only known drawing that attains this value is depicted in Figure 1.*

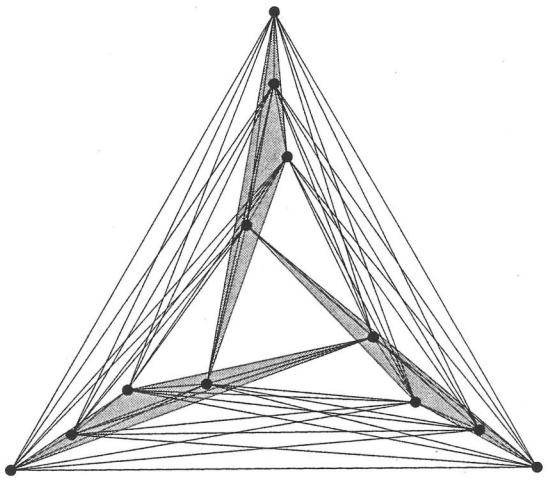


**Figure 1: Minimal drawing of $K_{12}$**

Actually, this result is not incidental. The drawing in Figure 1 is highly symmetric such that each subgraph $K_{11}$ is drawn optimally, which is sufficient for global optimality. This situation cannot be achieved in general, not even for smaller $n$ where it is met only for $K_4$ and $K_6$.

To describe the asymptotic bound, let $\overline{\nu}(n) = \overline{cr}(K_n)/\binom{n}{4}$. Inequality (1) implies that $\overline{\nu}(n)$ is a strictly increasing function and thus exists in the limit. Adopting the notation in [16], we define $\overline{\nu}^* = \lim_{n\to\infty} \overline{\nu}(n)$. By performing calculations through large $n$ we obtained the following value.

THEOREM 3. $\overline{\nu}^* > 0.311507$.

The lower bound for the limit in Theorem 3 has been improved several times. The previously best value has been 0.3001, in Brodsky et al. [5].

The same paper also offers a recursive construction that

yields an asymptotic *upper* bound of 0.3838, which is superior to previous results in Singer [17] and in Jensen [11]. Arbitrarily large drawings with few crossings are typically constructed by taking the best drawing $D$ known so far and recursively replacing each of its vertices by $D$. We could show that better results are achieved by a different, somewhat counter-intuitive construction. Replacement of vertices is done only *once*, with a fixed configuration of points in *convex* position. (We omit the details in this extended abstract.)

Despite the fact that convex configurations locally *maximize* the number of crossings, our novel strategy pays off in the global sense. To start with, we used our improved drawings for larger $n$, as given in Table 3. We obtained the following theorem.

THEOREM 4. $\overline{\nu}^* < 0.380739$.

It is worth mentioning that $\overline{\nu}^*$ is a geometric quantity of separate interest, by a connection to Silvester's four point problem, drawn in Scheinerman and Wilf [16].

## 3. REMARKS

Our experimental approach opened the door to various new results on the rectilinear crossing number of $K_n$. This constitutes one more example where computational experiments, when carried out carefully and combined with theoretical arguments, shed new light into notoriously difficult combinatorial problems. The interested reader will find detailed output data of the computations (in particular, point sets in small integer representation) at our web page [20].

A tantalizing question is that for the value of $\overline{cr}(K_{13})$. According to the new bounds in Table 2, $\overline{cr}(K_{13})$ is a member of $\{221, 223, 225, 227, 229\}$. Unfortunately, with current methods it seems out of reach to even perform a computation of all different order types of size 12. Still there is hope that an inspection of the currently 'best' drawings of $K_n$ for small $n$ might reveal regularities that lead to new theoretical insights. A small step in this direction is the observation that, as of yet, all these drawings showed triangular convex hulls. Also, from Tables 1 and 2 we see that, for $n$ odd, the number of inequivalent drawings with small numbers of crossings is quite large, probably because the lack of symmetries. This indicates that drawings attaining $\overline{cr}(K_n)$ might be easier to find for $n$ odd, and leads us to believe that $\overline{cr}(K_{13}) = 229$.

We currently are building up a data base for $n = 11$ of all *projective* order types, including their realizations if existent. This will enable us to access the (Euclidean) order types of size 11 faster while using reasonable sized disk space. An examination of questions similar to those in the present paper then will become easier in the future.

## 4. REFERENCES


[1] O.Aichholzer, F.Aurenhammer, H.Krasser, *Enumerating order types for small point sets with applications.* Proc. $17^{th}$ Ann. ACM Symp. Computational Geometry, Medford, MA, 2001, 11-18.

[2] O.Aichholzer, H.Krasser, *The point set order type data base: a collection of applications and results.* Proc. $13^{th}$ Ann. Canadian Conf. Computational Geometry CCCG 2001, Waterloo, Canada, 2001, 17-20.

[3] D.Bienstock, *Some provably hard crossing number problems.* Discrete & Computational Geometry 6 (1991), 443-459.

[4] A.Brodsky, S.Durocher, E.Gether, *The rectilinear crossing number of $K_{10}$ is 62.* The Electronic J. of Combinatorics 8 (2001), Research Paper 23.

[5] A.Brodsky, S.Durocher, E.Gether, *Toward the rectilinear crossing number of $K_n$: new drawings, upper bounds, and asymptotics.* Research report, Dept. Comput. Sci., UBC Vancouver, Canada, 2000.

[6] P.Erdös, R.K.Guy, *Crossing number problems.* Amer. Math. Monthly 88 (1973), 52-58.

[7] M.R.Garey, D.S.Johnson, *Crossing number is NP-complete.* SIAM J. Alg. Disc. Meth. 4 (1983), 312-316.

[8] J.E.Goodman, R.Pollack, *Multidimensional sorting.* SIAM J. Computing 12 (1983), 484-507.

[9] R.K.Guy, *A combinatorial problem.* Nabla (Bulletin of the Malayan Mathematical Society) 7 (1960), 68-72.

[10] F.Harary, A.Hill, *On the number of crossings in a complete graph.* Proc. Edinburgh Math. Society (2) 13 (1962), 333-338.

[11] H.F.Jensen, *An upper bound for the rectilinear crossing number of the complete graph.* J. Combinatorial Theory B 29 (1971), 212-216.

[12] F.T.Leighton, *Complexity issues in VLSI.* The MIT Press, Cambridge, 1983.

[13] J.Pach, G.Tóth, *Which crossing number is it, anyway?* J. Combinatorial Theory B 80 (2000), 225-246.

[14] J.Pach, G.Tóth, *Thirteen problems on crossing numbers.* Geombinatorics 9 (2000), 195-207.

[15] R.B.Richter, C.Thomassen, *Relations between crossing numbers of complete and complete bipartite graphs.* Amer. Math. Monthly 104 (1997), 131-137.

[16] E.R.Scheinerman, H.S.Wilf, *The rectilinear crossing number of a complete graph and Sylvester's "four point problem" of geometric probability.* Amer. Math. Monthly 101 (1994), 939-943.

[17] D.Singer, *The rectilinear crossing number of certain graphs.* Manuscript, 1971. Available at `http://www.cwru.edu/artsci/math/singer/`

[18] J.T.Thorpe, F.C.Harris, *A parallel stochastic optimization algorithm for finding mappings of the rectilinear minimal crossing number.* Ars Combinatorica 43 (1996), 135-148.

[19] W.T.Tutte, *Toward a theory of crossing numbers.* J. Combinatorial Theory 8 (1970), 45-53.

[20] `http://www.igi.TUgraz.at/oaich/triangulations/crossing.html`

# Planar graphs and metrically complete graphs*

J. Cáceres
Almeria Univ., Spain
jcaceres@ual.es

C. I. Grima
Seville Univ., Spain
grima@us.es

A. Márquez
Seville Univ., Spain
almar@us.es

A. Moreno-González
Huelva Univ., Spain
maria.moreno@dmat.uhu.es

## ABSTRACT

Given a set of points in the plane, we study those geometric graphs which have dilation 1 in the $l_1$ metric and that are planar. Firstly, we characterize the set of sites in the plane having a graph verifying both conditions and, if possible, we construct it. Secondly, we try to solve the opposite question, i.e., given a planar graph $G$, does there exist a set of sites $S$ such that $G$ is isomorphic to $M(S)$?

## 1. INTRODUCTION

There are many applications in which it is useful to find geometric graphs with few edges but with the property that the distance between two vertices measured in the graph is a good approximation of the actual Euclidean distance between those two vertices. This problem arises mainly in the design of networks or VLSI circuits. Given a set of sites $S$ in the plane, the *dilation* of a subgraph of the complete geometric graph is the largest ratio between the length of the shortest path from a pair of points of $S$ to the distance of those points in the plane. Since the metric that reflexes the distance between components in an electronic circuit is the $l_1$ metric, in previous works [1], we have presented some methods that allow to construct graphs with dilation very close to 1 in the $l_1$ metric. Nevertheless, in that metric we can think in graphs that are not the complete graph but they have dilation 1 (dilation free graphs); i.e., they give exactly the distance between any pair of vertices. In this way, given a set of sites $S$ in the plane, we call the *métrically complete graph* of $S$ (denoted $M(S)$) to the minimal dilation free graph (in the sense of smallest size). It is clear that in the Euclidean metric for points in general position the metrically complete graph is the complete graph, but we have proved that in the $l_1$ metric the metrically complete graph is strictly smaller than the complete graph; in fact, if $K(S)$ denotes the complete geometric graph on $S$, then $|K(S) - M(S)| \in O(N^{3/2})$.

*Partially supported by MCyT project BFM2001-2474

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

In this work, we continue that work studying some properties of the metrically complete graphs in the $l_1$ metric that are plane graphs. First of all, we try to know whether a set of sites has a planar metrically complete graph, obtaining a characterization that leads to an algorithm. Additionally, we present another algorithm that constructs the metrically complete graph of a set of sites if that graph is planar in $O(n \log n)$ time, in contrast to the quadratic time for the general case. Finally, we try to solve the question, given a planar graph $G$, does there exist a set of sites $S$ such that $G$ is isomorphic to $M(S)$?

## 2. PLANAR METRICALLY COMPLETE GRAPHS

As we have said above, the first question is to know whether a set of sites has a planar metrically complete graph. In the Euclidean metric this problem has been solved previously, and it is known that there exist only four infinite families of sites with that property plus an additional case [2]. In $l_1$ metric as $M(S)$ is smaller than in the Euclidean metric, we can wait that some new cases will appear. In fact this is true as we will see below.

In order to solve this first question we need to introduce some new concepts. First of all, we sort all points in $S$ into two list, one sorted by the ordinate and the other list sorted by the abscissa; so, given a point $u \in S$, among all points below $u$, we select the point immediately to its right ($r_b(u)$) and the point immediately to its left ($l_b(u)$); with those three points we can define a strip that will be called the *top strip* associated to $u$ (see Figure 1). In the same way we can define the bottom strip, the left strip, or the right strip. Additionally, if we proceed in the same way but with the points that are above $u$, we obtain another two points, $r_a(u)$ and $l_a(u)$. Now, we will see that those strips provide the information that we were sought.

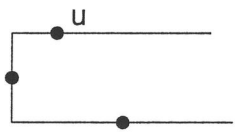



Figure 1: A top strip.

THEOREM 1. *Let $S$ be a set of $n$ sites in the plane. The following conditions are equivalent:*

1. *$M(S)$ is a planar graph.*
2. *Any quadrilateral with sides parallel to the coordinate axes defined by four points of $S$ in convex position contains another point of $S$ in its interior.*
3. *Any top strip is empty (contains no points of $S$).*
4. *Any bottom (left, right) strip is empty.*

This result allows us to describe an algorithm that checks whether $M(S)$ is planar in $O(n \log n)$ time. In order to obtain $M(S)$ we can use the following result

THEOREM 2. *If $M(S)$ is a planar graph, then graph obtained by joining each $u \in S$ to the vertices $r_b(u)$, $l_b(u)$, $r_a(u)$, and $l_a(u)$ is $M(S)$.*

Observe that $M(S)$ is the graph described in Theorem 2 only when $M(S)$ is planar.

## 3. WHICH PLANAR GRAPH ARE ISOMORPHIC TO A METRICALLY COMPLETE GRAPH

Once we know which metrically complete graphs are planar, the question that arises now is, given a planar graph $G$, does there exist a set of sites $S$ such that $G$ is isomorphic to $M(S)$? In Figure 2 we can see a planar graph $G$ such that the answer to this question for that graph is negative. However, though we can not give a characterization of these graphs, we can give some necessary conditions for a planar graph in order to be isomorphic to a metrically complete graph.

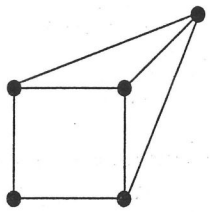


**Figure 2: A planar graph not isomorphic to a metrically complete graph.**

An *orthogonal embedding* of a graph $G$ is a planar embedding of $G$ such that each edge is represented by a chain of horizontal and vertical segments. A point where two segments part of the same edge meet is called a *bend.*

THEOREM 3. *Let $G$ be a planar graph with degree $\Delta(G) \leq 4$. If $G$ is isomorphic to a metrically complete graph, then:*

- *$G$ admits an orthogonal embedding with at most one bend in each edge.*
- *$G$ is a proper subgraph of a 4-connected planar graph.*

Regarding this theorem, we point out that there exists a characterization of the graphs which admit an orthogonal embedding with at most one bend in each edge in terms of forbidden graphs, (see [3]). And Thomassen proved that the graphs verifying the second condition of Theorem 3 are those graphs with a rectangular representation [4].

There are some natural questions that arise at this point. If $G$ is a planar graph that is not isomorphic to a metrically complete graph, is any subdivision of $G$ not isomorphic to a metrically complete graph? In Figure 3 we can see that the answer to this question is negative.

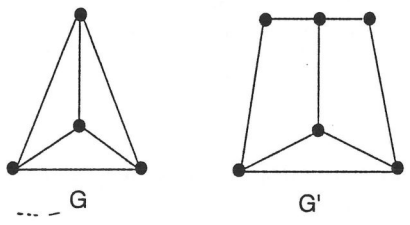



**Figure 3: The subdivision $G'$ of $G$ is isomorphic to a metrically complete graph.**

On the other hand, if $G$ is a planar graph isomorphic to a metrically complete graph, is any proper subgraph of $G$ isomorphic to a metrically complete graph? As in the previous question, the answer to this question is negative, (see Figure 4).

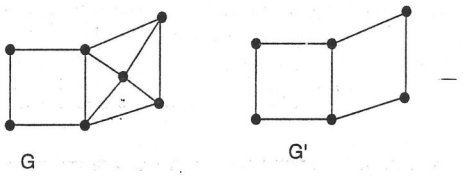



**Figure 4: The proper subgraph $G'$ of $G$ is not isomporphic to a metrically complete graph.**

However, although the answers to these questions are negative, we can know how is the structure of a planar graph isomorphic to a metrically complete graph.

THEOREM 4. *Let $G$ be a planar graph isomorphic to a metrically complete graph. Then, there exists a decomposition $G_1 \subseteq G_2 \subseteq \ldots \subseteq G$ where $G_{i+1}$ is constructed adding points to the exterior face of $G_i$ and $G_1$ is a cycle without chords.*

## 4. REFERENCES


[1] J. Cáceres, C. I. Grima, A. Márquez and A. Moreno-González, "Dilation free graphs in $l_1$-metric", *17th European Workshop on Computational Geometry, Berlin.*

[2] D. Eppstein, *http://www1.ics.uci.edu/ eppstein/junkyard/self.html.*

[3] Y. P. Liu, "Embeddability in Graphs", *Kluwer Academic Publishers*,1995.

[4] C. Thomassen, "Interval representations of planar graphs", *J. Combin. Theory B*, Vol. 40, pp. 9-20, 1986.

# On Simplifying Dot Maps (abstract)

Mark de Berg* Prosenjit Bose‡ Otfried Cheong* Pat Morin§

## 1. INTRODUCTION

An important component in the area of cartography is the ability to represent and visualize the distribution or density of some phenomenon such as the population distribution over a certain region. The most common technique to achieve this is the *dot map*. The term *dot map* is self-explanatory—it refers to the use of dots or points placed on a map to represent a given distribution. Dot maps are quite important and their use has been extensively studied in cartography—see for instance Chapter 8 of the book by Dent [5].

There are many issues involved in the use of dot maps as a tool for representing distributions. For example, the radius of the dots used, or the decision to allow or disallow dots to overlap are important visual considerations. In this paper, we concentrate on two related issues with respect to the generation of dot maps. The first is: Given a point set $P$ representing a given distribution, how can we automatically simplify it, that is, generate a smaller representative point set $Q$? This question arises when one wishes to scale a map: the number of points in the map has to decrease when the size of the map is decreased, otherwise it would become too cluttered. It may also arise in the generation of the initial dot map. For instance, generating a dot map of 1,000 dots representing the population density in a country with say, 10,000,000 million inhabitants can be seen as simplifying a dot map of 10,000,000 points. The first question—how can we compute a good approximation?—immediately leads to the second: Given sets $P$ and $Q$, how can we determine the quality of $Q$ as an approximation to $P$? Both these questions have already been addressed in the GIS and cartographic communities [5].

The current approaches to solve the first question are heuristic in nature. The main reason for this stems directly from the second question: the GIS literature we are aware of uses qualitative measures of quality. Although visually many of these algorithms seem to perform well, it is difficult to compare them. What is lacking is a quantitative measure of quality. To this end, we propose the notion of $\varepsilon$-approximations [9] as the appropriate quantitative measure. A set $Q$ of $m$ points is called an *$\varepsilon$-approximation* of a set $P$ of $n$ points[1] with respect to a family $\mathcal{R}$ of ranges, if for any range $r \in \mathcal{R}$ we have

$$\left| \frac{|r \cap P|}{n} - \frac{|r \cap Q|}{m} \right| \leqslant \varepsilon.$$

In other words, if we approximate the number of points from $P$ inside a range $r$ by multiplying the number of points from $Q$ inside the range by $n/m$ (the *dot value*), then we make an error of at most $\varepsilon n$. This leads us to define $\Delta_{\mathcal{R}}(Q, P)$, the *approximation error of $Q$ with respect to $P$*, for a family $\mathcal{R}$ of ranges, as

$$\Delta_{\mathcal{R}}(Q, P) = \max_{r \in \mathcal{R}} ||r \cap P| - (n/m) \cdot |r \cap Q||.$$

In this paper we focus on squares and rectangles[2] as ranges. Of these types of ranges, squares are probably most natural in our application. Another natural range to consider would be discs.

$\varepsilon$-Approximations have been studied and used extensively in computational geometry—see e.g. Chazelle's book [4]—and algorithms are known to compute $\varepsilon$-approximations of asymptotically optimal size for a set $P$ and a given value of $\varepsilon$. Note that we want to solve a slightly different problem: in our case $\varepsilon$ is not given, but the desired number of points in the approximation $Q$. Still, one may use the same type of algorithms. For instance, in many cases it turns out that random sampling is expected to produce an approximation of asymptotically optimal size. (One caveat is in place here: the optimality here refers to the worst-case size of an $\varepsilon$-approximation over all point sets $P$ of $n$ points, not to the

*Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, the Netherlands.

‡School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada, K1S 5B6. Research supported in part by NSERC.

§School of Computer Science, McGill University, 3480 University Street, Suite 318, Montréal, Québec, Canada H3A 2A7

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

[1]Traditionally, in the definition of $\varepsilon$-approximation it is required that $Q \subset P$, but this is not necessary.

[2]In this paper squares and rectangles are always axis-parallel.

minimum size needed for the given set $P$. These two sizes need not be the same.) Thus, for our problem we could simply take a random subset $Q \subset P$ of the desired size. This raises the question of computing the approximation error of given sets $P$ and $Q$. To our surprise, we couldn't find any efficient algorithms in the literature for this problem.

We obtain the following results on computing the approximation error for given sets $P$ and $Q$. If $\mathcal{R}$ is the family of all possible unit squares, then we can compute the approximation error of $Q$ with respect to $P$ in $O(n \log n)$ time. If $\mathcal{R}$ is the family of all possible squares, then we present a simple $O(m^2 n + n \log n)$ algorithm and an $O(n^2 \sqrt{n} \log n)$ time algorithm which is more efficient in the worst case. If $\mathcal{R}$ is the family of all possible rectangles, then we present two algorithms, a simple $O(m^2 n \log n)$ algorithm and a more efficient $O(mn \log n)$ time algorithm.

After giving algorithms to compute the approximation error for given $P$ and $Q$, we turn our attention to the experimental component of the paper. The goal is to develop heuristics to generate for a given set $P$ an approximation $Q$ of the desired size with as small an error as possible. We concentrate on the case of square ranges, as this seems most relevant to our application.

Our heuristics use as a subroutine an algorithm to compute the error for given $P$ and $Q$. Unfortunately, our algorithm for arbitrary squares is rather slow, and some of the heuristics call this subroutine many times. Hence, we first show experimentally that the exact error with respect to squares can be approximated well by computing the error with respect to fixed-size squares for a number of different sizes. After having established this, we turn our attention to the heuristics.

We consider four heuristics. The first is simply random sampling: take a random sample of the desired size. The second is repeated random sampling: take many samples and keep the best one. The third is an approach that we call the swapper: it tries to repeatedly improve a given approximation by deleting a point from $Q$ in an area where $Q$ oversamples $P$ and replace it by a point in an area where $Q$ undersamples $P$. This approach performs significantly better than (repeated) random sampling. The fourth heuristic we try was described by Dobkin and Tal [6]. They were originally interested in the dual setting: given a set of lines, find a smaller set of lines whose arrangement approximates the original arrangement. Dobkin and Tal solve the problem using dualization, so they arrive exactly at our problem. Although their approach seems more suited to minimizing the Hausdorff distance between $P$ and $Q$—indeed, they prove bounds on the minimal Hausdorff distance they achieve—they also use their algorithm in an application that is closely related to ours. Namely, they want to approximate the continuous half-plane discrepancy [4] of $P$ with the discrepancy of $Q$. (The continuous half-plane discrepancy of a set of points in the unit square is defined as the maximum, over all half-planes, of the absolute difference between the fraction of points in the half-plane and the fraction of the unit square covered by the half-plane.) Now if we would consider half-planes as regions, then the approximation error of $Q$ with respect to $P$ is an upper bound on the difference between the continuous discrepancies of $P$ and $Q$. Dobkin and Tal claim that for some distributions of $P$ the continuous discrepancy of $P$ can be estimated better by a set $Q$ computed with their algorithm than with a random sample. For this reason we also consider their algorithm in our experiments. In our setting, however, their algorithm gives poor results, especially for clustered data.

## 2. COMPUTING THE ERROR

Let $P$ be a set of $n$ points and $Q$ be a set of $m$ points in the plane, with $m < n$. We give algorithms to compute the approximation error of $Q$ with respect to $P$ for three different families of ranges: unit squares, arbitrarily sized squares, and rectangles. In this abstract, however, we omit the description of the algorithms for rectangles. By $\delta := n/m$ we denote the dot value of the points in $Q$.

### 2.1 Unit squares as ranges

Let $\mathcal{R}$ be the family of all possible unit squares. When we want to compute the approximation error of $Q$ with respect to $P$ for unit squares, it can make a difference whether we consider open or closed squares. In the description of the algorithm, we will consider the squares to be closed; it is easy to adapt the algorithm to the case of open squares.

Recall that we use the absolute value of the error in the definition of approximation error. It is convenient to compute separately the maximum positive error and the maximum negative error. Below we describe how to compute the maximum positive error; computing the maximum negative error can be done in a similar way.

A unit square contains a point if and only if the center of the unit square is contained in the unit square centered at the point. Hence, instead of considering the point sets $P$ and $Q$ and the family of all unit squares as ranges, we can use the sets $S_P$ and $S_Q$ of unit squares centered at the points in $P$ and $Q$, and all points in the plane as ranges. Call the squares in $S_P$ the *red squares*, and the squares in $S_Q$ the *blue squares*. The (positive) approximation error of a point $x$ in the plane is now: (number of red squares containing $x$) - $\delta$·(number of blue squares containing $x$). The approximation error of $S_Q$ with respect to $S_P$ is the maximum approximation error over all points in the plane, which from the discussion above is the same as the approximation error of $Q$ with respect to $P$ for the family $\mathcal{R}$ of unit squares as ranges.

The arrangement formed by the squares $S_Q$ and $S_P$ partitions the plane into faces where the approximation error of any point in a face of the arrangement is the same. Therefore, finding the maximum approximation error amounts to finding the face with maximum approximation error. We do this with a plane-sweep algorithm. In this algorithm, we sweep a vertical line $\ell$ from left to right over the arrangement. As we sweep the arrangement, we maintain the maximum approximation error over the faces of the arrangement intersected by $\ell$. Since the arrangement is formed by squares, the only events are when the sweep line reaches a left or right edge of a square. At each event we compute the maximum error of all points on $\ell$ and of all points slightly to the right of $\ell$ (but to the left of the previous event). The maximum error found in all the events will be the max-

imum error of $S_Q$ with respect to $S_P$. We now describe the information we maintain during the sweep—the status structure—and how to handle the events.

*A dynamic 1-dimensional structure.* The status structure is a dynamic data structure for solving the following 1-dimensional version of the problem. We are given a set $I_R$ of red segments and a set $I_B$ of blue segments on the real line, and a parameter $\delta$. The (positive) approximation error of a point $x \in \mathbb{R}$ is defined as: (number of red segments containing $x$) $-$ $\delta$·(number of blue segments containing $x$). We want to maintain the maximum error over all points in $\mathbb{R}$ under insertions and deletions of segments.

The structure we use is essentially the structure introduced in [3] in the context of grid placement problems. The structure maintains a function $f : \mathbb{R} \to \mathbb{R}$. Initially, it is assumed that $f(x) = 0$, for all $x \in \mathbb{R}$. The following update and query operations are allowed on the structure:

1. Insert($[a : b], d$): Increase the value of $f(x)$ by $d$ over the interval $[a : b]$.
2. Delete($[a : b], d$): Decrease the value of $f(x)$ by $d$ over the interval $[a, b]$.
3. Max(): Return $\max\{f(x) : x \in \mathbb{R}\}$.

The first two operations can be performed in $O(\log n)$ time, where $n$ is the number of intervals currently inserted, and the third operation takes $O(1)$ time.

With this structure, the 1-dimensional problem is easily solved. When inserting (resp. deleting) a red segment, we increase (resp. decrease) the value of $f(x)$ by 1 over this segment. Similarly, when inserting (resp. deleting) a blue segment, we decrease (resp. increase) the value of $f(x)$ by $\delta$ over this segment. Max() allows one to recover the maximum approximation error over the currently inserted segments.

Going back to the plane-sweep algorithm for the 2d case, we see that every event takes $O(\log n)$ time to process and that the initialization takes $O(n \log n)$ time. Since there are $O(n)$ events to process, we get the following theorem.

THEOREM 2.1. *Let $P$ be a set of $n$ points in the plane, and let $Q$ be a set of $m$ points in the plane, with $m \leqslant n$. The approximation error of $Q$ with respect to $P$ for the family of all unit squares can be computed in $O(n \log n)$ time.*

## 2.2 Arbitrary squares as ranges

The case of squares of arbitrary size as ranges is probably the most interesting in our application. In this abstract we only discuss a fairly simple algorithm that runs in $O(m^2 n + n \log n)$ time. We also have an algorithm whose worst-case running time is subcubic—to be precise, it is $O(n^2 \sqrt{n} \log n)$—but this algorithm is much more complicated and we don't expect it to be any faster in practice. As before, we show how to compute the positive approximation error; the negative error can be computed in a similar way.

We first give a lemma which restricts the number of candidate squares. Let $B$ be the bounding box of $P \cup Q$.

LEMMA 2.2. *There is an open square with maximum positive error such that two opposite sides of the square each either contain a point from $Q$ or are contained in the boundary of $B$.*

Lemma 2.2 allows us to take the following approach. The square of maximum discrepancy must have a blue point (i.e. a point from $Q$) on two opposite sides. Given two blue points, if the absolute value of the difference in their $x$-coordinate is larger than the absolute value of the $y$-coordinate difference, then the two points can only lie on the left and right sides of a square. Similarly, if the $y$-coordinate difference is larger, than the two points can only lie on the top and bottom edges of the square. Finally, if the differences are the same, then there is a unique square with the points at the opposite corners. This implies that a given pair of blue points determines the size of the square and the direction of search. Since there are $m$ blue points, there are $\binom{m}{2}$ candidate pairs. Select one such pair, $q_i, q_j$, and assume without loss of generality that the $y$-coordinate difference is larger. The case where the $x$-coordinate difference is larger is symmetric, and the other case is trivial.

Given $q_i$ and $q_j$, let $h_i$ and $h_j$ be the horizontal lines through the respective points. We have to find the maximum error over all squares whose top end bottom edges are contained in those lines. In order to find this maximum, we will sweep (i.e. move) the square from left to right through the strip.

Consider the points in $P \cup Q$ that lie within this strip. Sort these points by their $x$-coordinates, and let $S$ represent this set in sorted order. Start with the left boundary of the square on the left boundary of $B$. Compute the discrepancy of this square by finding the points of $S$ in this square. Now, sweep the square from left to right until the right boundary reaches the right boundary of $B$ and maintain the maximum at each step. The events in this sweep are that either a point leaves the square or a point enters the square. The order in which the points enter as well as the order in which the points leave is the sorted order. Processing an event amounts to adding or subtracting the appropriate amount to the current discrepancy, depending on which point enters or leaves. Note that we do not need to sweep the whole strip but only the portion of the strip where $q_i$ and $q_j$ are on the top and bottom edges of the square. However, this optimization does not make a difference asymptotically. Since each event can be processed in $O(1)$ time given the sorted order, we can compute in $O(n)$ time the maximum discrepancy given a candidate pair of points provided the points in the strip are sorted. If we pre-sort the points of $P \cup Q$ in $O(n \log n)$ time then $O(n)$ time the sorted order of points of $P \cup Q$ within a strip can be obtained. Since there are $O(m^2)$ possible candidates and each candidate can be verified in $O(n)$ time, the total time for the algorithm is $O(m^2 n + n \log n)$.

THEOREM 2.3. *Let $P$ be a set of $n$ points in the plane, and let $Q$ be a set of $m$ points in the plane, with $m \leqslant n$.*

| | Uniform | | | | | | | Clustered | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | 1000 | | | 5000 | | | | 1000 | | | 5000 | | | |
| $m$ | 50 | 100 | 250 | 50 | 100 | 250 | 500 | 50 | 100 | 250 | 50 | 100 | 250 | 500 |
| $\delta$ | 20 | 10 | 4 | 100 | 50 | 20 | 10 | 20 | 10 | 4 | 100 | 50 | 20 | 10 |
| RS | 198 | 137 | 80 | 985 | 704 | 454 | 319 | 203 | 141 | 79 | 1043 | 722 | 456 | 327 |
| Best of 50 | 131 | 97 | 62 | 654 | 495 | 332 | 203 | 124 | 80 | 53 | 802 | 439 | 311 | 203 |
| Dobkin-Tal | 157 | 99 | 73 | 756 | 541 | 300 | 244 | 205 | 166 | 133 | 1272 | 1279 | 1003 | 917 |
| Swapper | 104 | 82 | 49 | 668 | 443 | 307 | 227 | 110 | 72 | 54 | 634 | 364 | 265 | 171 |

**Table 1: Various heuristics**

*The approximation error of $P$ with respect to $Q$ for squares can be computed in $O(m^2n + n\log n)$ time.*

## 3. FINDING GOOD APPROXIMATIONS

We now turn our attention to finding good approximations of a specified size $m$ for a given set $P$ of $n$ points. We will concentrate on square ranges, as this seems most natural in our application.

*Data sets.* The input sets $P$ that we use consist of $n$ points in the unit square for various values of $n$. We use three types of distributions: uniform, clustered, and real-world data. The clustered data sets were generated by as follows. After deciding on the number of clusters (ten for $n = 1000$ and twenty for $n = 5000$), we randomly choose that many cluster centers, draw a circle around each center (the radius depends on the number of clusters), and generate points randomly within that circle according to a distribution that generates more points close to the center. Which fraction of the points goes to which cluster is also determined randomly.

*Computing the error.* Some of our heuristics call a subroutine to compute the error for given $P$ and $Q$ many times. We have implemented the $O(m^2n\log n)$ algorithm for computing the error for square ranges bescribed in the previous section, but for large $n$ and $m$ it is rather slow. To speed up the heuristics we therefore want to replace the subroutine by a faster one. We do this by computing the error for squares of a fixed size, for several different sizes; for a fixed size we used the $O(n\log n)$ algorithm described earlier for unit squares. The hope is that if the number of sizes is large enough, the error we find is close enough to the real error, so that it will not harm our heuristics. Our first experiment is to test whether this hope is justified: we compare the real error, computed with the $O(m^2n\log n)$ algorithm, to the error computed by looking at squares of $k$ different sizes only, for various values of $k$. We did this by generating between 5 and 10 different sets $P$, and for each $P$ between 8 and 20 different sets $Q$. Half of the choices for $Q$ were taken as random samples from $P$, the other half was generated using another distribution. We then computed the average difference between the error for arbitrary squares and the error for $k$ different fixed sizes, where the sizes were equally spaced. (We also tried sizes on a logarithmic scale, but obtained poorer results.) If we take $k = 60$, then average difference between the real error and the estimated error is always close to (and often smaller than) the dot value, and the maximum difference is close to twice the dot value. We conclude from this that it is safe to use the estimated error in our heuristics.

*The heuristics.* We then experimented with the four heuristics described below for generating an approximation $Q$ of a desired size $m$ for a given set $P$ of $n$ points. Table 1 shows our preliminary results. All errors were computed using squares of 60 fixed sizes. Each column was computed for only one set $P$.

*Heuristic 1: Random sampling.* Here we simply take a random sample $Q$ of $P$. The table shows the average error obtained (we have taken 50 samples and averaged).

*Heuristic 2: Best of $k$.* Here we take $k$ random samples $Q_1, \ldots, Q_k$ of $P$, compute the approximation error for each of them, and return the best sample. Here $k$ is a parameter. The table shows the result for $k = 50$.

*Heuristic 3: Dobkin-Tal.* This algorithm proposed by Dobkin and Tal [6] produces an approximation that is not a subset of $P$, by repeatedly finding closest pairs and replacing them by their midpoint.

*Heuristic 4: Swapping.* We first obtain an approximation $Q$ by random sampling, and then try to improve it as follows. We compute a range $r_{\text{pos}}$ with the largest positive error and a range $r_{\text{neg}}$ with the largest negative error. We remove a random point in $Q \cap r_{\text{neg}}$ from $Q$, and add a random point in $(P \setminus Q) \cap r_{\text{pos}}$ to $Q$.

*Other heuristics.* We also tried heuristics that construct an approximation incrementally. Suppose we have already constructed an approximation $Q_i$ consisting of $i$ points. Then we create the approximation $Q_{i+1}$ by choosing $k$ points are random from $P \setminus Q_i$, evaluate the error in $Q_i \cup \{q\}$ for each chosen point $q$, and add the best $q$ to $Q_i$ to obtain $Q_{i+1}$. In another approach, we first obtain a random sample that is too large (say, of $2m$ points), and then iteratively remove a random point in a range with the largest negative error. However, neither approach leads to satisfactory results.

The swapper is clearly the best of our heuristics.

Choosing the best out of a small number of samples—a technique often mentioned in the literature—is *not* such a good strategy, and Dobkin-Tal is not so bad for uniformly distributed points but fails to preserve information for non-uniformly distributed ones.

However, these are preliminary results, and we hope to develop better heuristics in the future. For instance, the swap-

per strategy bears similarity to many common local optimizers, so standard techniques such as simulated annealing or genetic algorithms may improve it. We also plan to experiment with data for which we know the optimal approximation—one-dimensional data for instance—to see how close to the optimal solution out heuristics get.

# 4. REFERENCES


[1] J. Bentley. Programming Pearls: Algorithm Design Techniques *Communications of the ACM*, 27(9), p. 865–871, 1984.

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

[3] P. Bose, M. van Kreveld, A. Maheshwari, P. Morin and J. Morrison. Translating a Regular Grid over a Point Set *Computational Geometry: Theory and Applications*, to appear.

[4] B. Chazelle. The Discrepancy Method: Randomness and Complexity Cambridge University Press, 2000.

[5] B.D. Dent. *Cartography: Thematic Map Design*. WCB McGraw-Hill, 1999.

[6] D.P. Dobkin and A. Tal. Efficient and small representations of line arrangements with applications. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 293–301, 2001.

[7] M.H. Overmars and C.K. Yap. New Upper Bounds in Klee's Measure Problem. *SIAM J. Comput.* 20:1034–1045 (1991).

[8] Resource Assesment Division. Natural Resources Conservation Service – USDA. Acres of harvested cropland, 1992. `http://www.nhq.nrcs.usda.gov/land/meta/m2815.html`

[9] V. N. Vapnik and A. J. Chervonenkis. On The Uniform Convergence of Relative Frequencies of Events to the Probabilities. *Theory of Probability and Its Applications*. 16:264–280 (1971).

# Improved Output-Sensitive Construction of the Vertical Decomposition of Three-Dimensional Arrangements*

Hayim Shaul
School of Computer Science
Tel Aviv University
hayim@post.tau.ac.il

Dan Halperin
School of Computer Science
Tel Aviv University
danha@post.tau.ac.il

## ABSTRACT

We describe a new output-sensitive algorithm for computing the vertical decomposition of arrangements of $n$ "well-behaved" surfaces in $\mathbb{R}^3$ in $O(n \log^2 n + V \log n)$ time, where $V$ is the complexity of the decomposition. This improves significantly over the best previously known algorithms. We implemented the algorithm for the case of arrangements of triangles (in 3-space) and we report on experiments with this implementation.

## 1. INTRODUCTION

Given a finite collection $\mathcal{S}$ of geometric objects such as hyperplanes or spheres in $\mathbb{R}^d$, the *arrangement* $\mathcal{A}(\mathcal{S})$ is the decomposition of $\mathbb{R}^d$ into connected open cells of dimensions $0, 1, \ldots, d$ induced by $\mathcal{S}$. Arrangements have been studied intensively in combinatorial and computational geometry [1],[7],[10]. Besides being interesting in their own right, they are the underlying structure of geometric problems in a variety of application domains such as robot motion planning, computer vision, structural molecular biology and more.

A raw arrangement is often too complicated to handle and use as it may have cells with many features and complex topologies. What is typically needed is a further refinement of the arrangement into cells each homeomorphic to a ball and having small combinatorial complexity (that is, a small constant number of features). Additionally we would like the refinement to be economical and not to increase the complexity of the arrangement by much. A refinement that satisfies these requirements is the so-called *vertical decomposition*. Figure 1(a) depicts an arrangement of segments (in bold lines) refined by vertical decomposition: we extend a vertical line upwards and downwards from every vertex of the arrangement (either a segment endpoint or the intersection of two segments) until it hits another segment or extends to infinity (in the figure the decomposition is clipped within a rectangle). Vertical decompositions are defined for any dimension and for arrangements of any collection of "well-behaved" objects [4],[10],[11],[13].

In this work we are concerned with three-dimensional arrangements and primarily with arrangements of triangles. Let $T = \{t_1, t_2, \ldots, t_n\}$ be a collection of triangles in 3-space. For a curve $\gamma$ in $\mathbb{R}^3$ let $H(\gamma)$ denote the vertical wall through $\gamma$, namely the union of vertical lines intersecting $\gamma$. Here and throughout the paper by vertical we mean parallel to the $z$-axis. For an edge $e$ of the arrangement we define the wall of the edge, denoted $W(e)$, as the union of points in $H(e)$ that can be connected to $e$ with a vertical segment that does not cross any of the triangles in $T$. The vertical decomposition of $\mathcal{A}(T)$ is obtained as follows. First we erect walls from triangle boundary edges. Second, walls are erected from the intersection edges between pairs of triangles to produce a finer decomposition. Finally we refine the decomposition, in a straightforward manner, into a convex subdivision consisting of trapezoidal prisms. (See [6] for details.) We call the refined subdivision the *vertical decomposition* of the arrangement.

The maximum combinatorial complexity of the vertical decomposition is the same as that of the arrangement, that is $\Theta(n^3)$ [6]. de Berg et al. showed that the complexity of the vertical decomposition is sensitive to the complexity of the underlying arrangements. They gave a bound

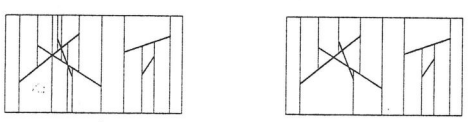



**Figure 1: Vertical decomposition (a) and partial decomposition (b) of an arrangement of segments**

*This work has been supported in part 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), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

$O(n^{2+\epsilon} + K)$ where $K$ is the complexity of the arrangement, improved by Tagansky [15] to $O(n^2\alpha(n)\log n + K)$. The near-quadratic overhead term is close to optimal in the worst case as there are arrangements with linear complexity whose vertical decomposition has quadratic complexity. They also gave an output-sensitive algorithm to compute the decomposition running in time $O(n^2 \log n + V \log n)$, where $V$ is the complexity of the decomposition.

The new result that we describe here is an improvement of that algorithm. The new algorithm runs in time $O(n \log^2 n + V \log n)$—an improvement of close to a factor $n$ in the overhead term. It is also conceptually somewhat simpler than the original algorithm as it constructs the vertical decomposition in one pass of a space sweep. This in turn makes the new algorithm easier to implement.

An algorithm that achieves a sub-quadratic overhead time was also given in [6]. This algorithm is substantially more complicated and the savings are small. It runs in time $O(\min(n^{4/5+\epsilon}V^{4/5}, n^2 \log n) + V \log n)$, for any $\epsilon > 0$. Thus the new algorithm improves on the sub-quadratic overhead algorithm as well.

The improved-overhead algorithm extends to the case of arrangements of well-behaved surface patches (see [10],[13] for a precise definition), requiring the same asymptotic time.

We implemented the algorithm as well the algorithm of [6] for the case of triangles. The new algorithm runs considerably faster. We briefly discuss the implementation and experiments below.

**Remarks.** (i) In [14] we also propose and investigate an alternative sparser refinement, which we call the *partial* vertical decomposition and has the advantages that it produces fewer cells and requires lower degree constructors. The partial decomposition is the three-dimensional extension of the (much simpler) decomposition depicted in Figure 1(b) for a planar arrangement of segments. (ii) For alternative decompositions see [2],[3],[5].

## 2. IMPROVED OUTPUT-SENSITIVE CONSTRUCTION

Our algorithm is an improvement of the sweep-based algorithm by de Berg et al. [6]; for brevity we call the algorithm of [6] the NQO (near-quadratic overhead) algorithm. We start this section by discussing certain aspects of NQO and then we explain the modifications that bring about the improvement.

The major innovation in NQO was the computation in an output-sensitive manner of a specific type of features of the full decomposition, namely vertical edges that are the intersection of a vertical wall emanating from the ceiling of a cell with a vertical wall emanating from the floor of the same cell, where each of these walls is induced by an edge of the arrangement where two triangles intersect. (The description here is rather brief and we refer the reader to [6] for further details.) These features are identified during a space sweep. Prior to the sweep, NQO computes other features of the decomposition and inserts all their vertices as events to the sweep's queue: (i) the vertical walls through the triangle boundary edges and (ii) for each triangle, 2D arrangements consisting of the intersection segments of the triangle with other triangles as well as the boundaries of walls from step (i) that abut on the triangle. These preparatory steps induce the overhead term $O(n^2 \log n)$ in the running time of NQO.

The crucial observation leading to the improvement is the following: Hardly any preprocessing is necessary before the sweep. Almost all the events, with the exception of exactly $n$ events, can be *fully predicted* during the sweep. By fully predicted we mean that not only do we know that the event is going to take place and at a certain $x$ coordinate, we also have sufficient information that will allow us to efficiently access the exact location of the event on the sweep plane when its time comes. The event type for which we do not have this information is the appearance of a new triangle (namely the first time that the sweep plane hits a new triangle). We next review the key steps and new ideas of the algorithm.

We assume that the input triangles in $T$ are in general position. For convenience of exposition, we also assume that the triangles in $T$ are bounded inside a big simplex (extra four triangles) and we are only interested in the decomposition inside this bounding simplex. The output of the algorithm is a graph $G = (U, E)$ where each node in $U$ describes one trapezoidal prism of the decomposition and there is an edge $(u_1, u_2)$ in $E$ if the two prisms corresponding to $u_1$ and $u_2$ share a vertical wall.

The algorithm consists of one pass of a space sweep with a plane orthogonal to the $x$-axis moving from $x = -\infty$ to $x = \infty$. Let $P_{x_1}$ denote the plane $x = x_1$. Let $\mathcal{A}_{x_1}$ denote the following subdivision: it is the partial two-dimensional decomposition of the arrangement $\mathcal{A}(P_{x_1} \cap T)$ of segments induced on the plane $P_{x_1}$ by intersecting it with the triangles in $T$ and the bounding simplex and adding vertical extensions only through segment endpoints (which looks, up to the bounding simplex, like Figure 1(b)). We use $\mathcal{A}_x$ to denote this subdivision for an arbitrary $x$-value.

Besides the graph $G$ in which the output is constructed, the algorithm maintains three data structures. A dynamic structure $\mathcal{F}$ which describes the subdivision $\mathcal{A}_x$, a standard event queue $\mathcal{Q}$ which maintains the events of the sweep ordered by their $x$-coordinate, and a dictionary $\mathcal{D}$ that connects between $\mathcal{F}$ and $\mathcal{Q}$ as we explain next.

The structure $\mathcal{F}$ supports efficient insertion or deletion of vertices, edges, and faces of the subdivision. In addition it efficiently answers vertical ray shooting queries. In order for the overall algorithm to be efficient, we wish to refrain from point location in the subdivision $\mathcal{A}_x$ as much as we can, since in the dynamic setting point location queries are costly. We achieve this by using the dictionary $\mathcal{D}$. To each feature in the current subdivision $\mathcal{A}_x$ we give a unique combinatorial label (we omit the straightforward details in

this abstract). We keep a dictionary of all these features with cross pointers to their occurrence in $\mathcal{F}$. When we add an event that will occur later at $x'$ to $\mathcal{Q}$ we also insert the combinatorial labels of features of $A_{x'}$ that are related to the event. This way, when we come to handle the event, we could use the dictionary (paying $O(\log n)$ to search in the dictionary) and with the information thus obtained we directly access $\mathcal{F}$.

If we do not compute events in advance (as NQO does) how do we predict all the events together with the extra labels needed? We illustrate it for the following event; see Figure 2. A segment endpoint $q$ crosses from one face of $\mathcal{A}_x$ to a neighboring face. Suppose that we know the location information related to this event. We now show how to handle it and how to predict future events induced by this event. Handling it is easy once we know the faces $f_1$ and $f_2$. We merge the faces split by the downward extension from $q$ into one face. Notice that we did not compute the vertical walls in advance therefore in order to know where the vertical extension from $q$ hits the top boundary of $f_2$ we need to do vertical ray shooting (which $\mathcal{F}$ supports). We split $f_2$ into the relevant three new faces. How do we predict for example a change in the vertical wall that the upward vertical extension from $q$ induces, say if its top endpoint crosses over from $e_1$ to $e_2$? The answer is that at the time we introduced the new vertical wall from $q$ we look at the two neighboring vertices of its top endpoint ($w_1$ and $w_2$ in Figure 2). Each vertex $v$ of $A_x$ represents the intersection of some edge $e(v)$ of the three-dimensional arrangement with the sweep plane. We project $e(q)$ and $e(w_2)$ onto the $xy$-plane and check whether the projections intersect; say they intersect at $x$-coordinate $x'$ which is greater than the current $x$. If so there is a potential 'change of wall boundary' event at $x'$. If this event will indeed take place, we know that the vertex $w_2$ will be a feature of the subdivision $A_{x'}$ and this will serve as a handle to access the right features of $\mathcal{F}$ when the new event is handled. Therefore we store it in $\mathcal{Q}$ together with the other parameters of the event. (We similarly check the intersection of the projection of $e(q)$ and $e(w_1)$.)

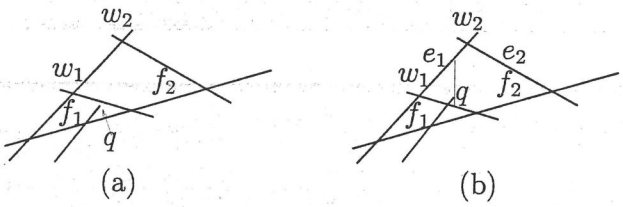



**Figure 2: Handling an event related to vertical walls emanating from a triangle boundary edge: (a) before the event and (b) after**

The new observation is that almost all the events can be predicted in this way, namely, every time we update the structure $\mathcal{F}$ we have to go over a constant-length checklist involving the modified features of $\mathcal{F}$. The overall cost of the prediction is $O(\log n)$ (this is a property of $\mathcal{F}$) and at most a constant number of new events is created and inserted into the queue. Some of these events may later turn out to be 'false alarms.' However, a false event never spawns a new event, the prediction of a false event can be charged to an actual event, and no actual event is charged for more than a constant number of false events. The full list of events, how they are detected and handled, is given in [14].

NQO also computes all the features of the arrangement $\mathcal{A}(T)$ prior to the sweep. It is not difficult to verify that most of these features/events can be fully predicted during the sweep. There is one type of events, however, for which we cannot obtain the full event information from previous events. This is obviously the appearance of a new triangle $t$, namely the first time that the sweep plane hits $t$. This is what determines the choice of structure for $\mathcal{F}$. We use the dynamic point location structure of Goodrich and Tamassia [9] which pertains to monotone subdivisions (ours is even convex) and takes $O(\log n)$ per update and $O(\log^2 n)$ per point location query. Augmenting it to support vertical ray shooting in a known face in time $O(\log n)$ is trivial. We use the structure for point location exactly $n$ times.

Now we can summarize the performance of the algorithm. Since we need less preprocessing the overhead term $O(n^2 \log n)$ in the running time of NQO is dropped; the prediction work is absorbed in the $O(V \log n)$ term (recall that $V$ is the complexity of the decomposition). The extra machinery required for handling the appearance of new triangles gives rise to the new overhead term $O(n \log^2 n)$ and it incurs additional work (of the point location structure, which although we use scarcely, needs to be maintained) that is also absorbed in the $O(V \log n)$ term. The storage required by the algorithm is $O(V)$.

THEOREM 2.1. *Given a collection $T$ of $n$ triangles in general position in three-dimensional space, the time needed to compute the vertical decomposition of the arrangement $\mathcal{A}(T)$ is $O(n \log^2 n + V \log n)$, where $V$ is the combinatorial complexity of the vertical decomposition.*

As observed above, the subdivision $A_x$ is convex. The algorithm however does not rely on this fact in any way. For the algorithm to apply, it suffices that the subdivision $A_x$ be $y$-monotone (where $y$ is the horizontal axis on the plane $P_x$). The structure of Goodrich and Tamassia can handle monotone subdivisions [8]. Hence we can generalize the result to the case of well-behaved surface patches. These are for example a collection of algebraic surfaces patches of bounded degree each bounded by at most some constant number of algebraic curves of bounded degree and each decomposed into a constant number of $xy$-monotone patches.

THEOREM 2.2. *Given a collection $S$ of $n$ well-behaved surface patches in general position in three-dimensional space, the time needed to compute the vertical decomposition of the arrangement $\mathcal{A}(S)$ is $O(n \log^2 n + V \log n)$,*

| Scene no. | # of triangles | NQO without normalize (in sec.) | NQO with normalize (in sec.) | the new algorithm (in sec.) |
|---|---|---|---|---|
| 1 | 20 | 452 | 463 | 33 |
| 2 | 5 | 42,809 | 29,045 | 19 |
| 3 | 95 | 7,595 | 6,888 | 96 |
| 4 | 24 | 149,069 | 84,226 | 230 |
| 5 | 9 | 208,259 | 210,452 | 59 |
| 6 | 6 | 4350 | 2.3 | 0.5 |
| 7 | 99 | 2068 | 2654 | 155 |
| 8 | 150 | 1967 | 1760 | 429 |

**Table 1: Summary of results: construction time of the full decomposition while using NQO (without and with normalize) and the new algorithm**

*where $V$ is the combinatorial complexity of the vertical decomposition.*

## 3. IMPLEMENTATION FOR TRIANGLES

We implemented our new algorithm as well as the NQO algorithm for computing the vertical decomposition of arrangements of triangles. We also augmented each implementation with the option to compute the partial decomposition. Both algorithms were implemented in C++. The implementation of the NQO algorithm spans roughly 14,000 lines of code whereas the implementation of the new algorithm has only roughly 8,000 lines of code.

We used LEDA's rational numbers in our implementation to achieve exact computation [12]. LEDA represents a rational number by a numerator and a denominator each being an integer of arbitrary precision. LEDA also provides a "normalize" function which divides the numerator and the denominator by their greatest common divisor.

Recall that the new algorithm resorts to dynamic point location because of the $n$ events where a new triangle appears. In practice we carry out these $n$ point-location queries naively. The time of this computation is negligible.

In a previous implementation of the NQO we did not use the normalize function. In a more recent implementation we incorporated calls to the normalize function resulting in a significant speedup.

We tested the two implementations on eight scenes (which are described in [14]), and measured the time it took to construct the full and decomposition. (We also experimented with the partial decomposition; we omit these results here.) When creating the scenes, we made sure that the scenes are degeneracy free. The tests were run under Linux on a computer with a Celeron 400MHz CPU and 512 MByte RAM. The results are given in Table 1. Although we do not describe the scenes here, the table still conveys the improvement that the new algorithm brings about.

As can be seen in Table 1 the running time of the new algorithm is significantly better than that of NQO. The reader should however take into consideration that the new algorithm was implemented after we gained considerable experience with the implementation of NQO and this partly contributed to the improvement. There are however other factors.

The implementation of NQO consists of three steps: (i) preprocessing, (ii) sweep, and (iii) final refinement. That is, the third step in the NQO algorithm further refines the cells computed in the second step into trapezoidal prisms. (This step is applied only when computing the full decomposition.) In NQO the decomposition is refined in the third step by first erecting walls from intersection edges, and then by adding walls parallel to the $yz$-plane emanating from one-dimensional features that are parallel to the $y$-axis. This refinement is carried out (in NQO) by constructing $n$ full two dimensional-arrangements which is highly time consuming. In the new algorithm these walls are simply added during the sweep in a fairly straightforward manner.

Another difference in the implementation is that in the implementation of the NQO algorithm (with normalize), we normalize every result at each step of every calculation. These normalization turned out to be costly and unnecessary. In the new algorithm we normalize only results that are the final results of calculations and not temporary values created during the calculation. Consider for example scenes 7 and 8. These scenes are simple (with no intersections) and adding normalization increased the running time, from 2068 seconds to 2654 seconds in scene 7 and from 1967 seconds to 3024 seconds in scene 8.

## 4. REFERENCES


[1] P. K. Agarwal and M. Sharir. Arrangements. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

[2] B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. *Combinatorica 10(2)*, pages 137–173, 1990.

[3] B. Aronov and M. Sharir. Castles in the air revisited. *Discrete Comput. Geom.*, 12:119–150, 1994.

[4] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. *Theoret. Comput. Sci.*, 84:77–105, 1991.

[5] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[6] M. de Berg, L. Guibas, and D. Halperin. Vertical decomposition for triangles in 3-space. *Discrete Comput. Geom. 14*, pages 35–62, 1995.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.

[8] M. Goodrich. Personal communication.

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

[10] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.

[11] V. Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. In *Proc. 42nd IEEE Symposium on Foundatioons of Computer Science*, pages 56–65, 2001.

[12] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, Cambridge, UK, 2000.

[13] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, 1995.

[14] H. Shaul. Improved output-sensitive construction of vertical decompositions of triangles in three-dimensional space. M.Sc. thesis, School of Computer Science, Tel Aviv University, Tel Aviv, Israel, 2001.

[15] B. Tagansky. A new technique for analyzing substructures in arrangements of piecewise linear surfaces. *Discrete Comput. Geom.*, 16:455–479, 1996.

# A better approximation algorithm for covering polygons with squares *

[Extended Abstract]

Grażyna Zwoźniak
Institute of Computer Science
Wrocław University, Poland
grazyna@ii.uni.wroc.pl

## ABSTRACT

We study the problem of covering a polygon without any acute interior angles, using a preferably minimum number of squares. The squares are allowed to overlap and they must lie entirely within the polygon. We show an $O(n \log n + \mu(P))$ time algorithm which covers any $n$-vertex input polygon with at most $10.5n + \mu(P)$ squares, where $\mu(P)$ denotes the minimum number of squares required to cover $P$. In the hole-free case our algorithm runs in linear time.

## 1. INTRODUCTION

One of the fundamental topics of computational geometry is how to decompose polygons into simpler objects, such as triangles, squares, rectangles, convex polygons. Considered polygons can contain polygonal holes. If the objects are allowed to overlap, then we call the decomposition a *covering*.

Considerable attention has received the problem of covering the polygon $P$ with a minimum number of squares, all internal to the polygon, whose union is $P$. One application for this problem is storage images [13].

The *rectilinear case*, i.e. when the polygon and the squares have sides that are vertical or horizontal, has been treated in several papers [1, 2, 11, 13]. Authors use a boolean (zero-one) matrix, where one represents a point inside the polygon and zero – a point outside it. A complexity is measured in terms of the number of points in the matrix, denoted $p$. For most practical applications $p \gg n$. Aupperle, Conn, Keil and O'Rourke [1] show that the rectilinear case is NP-hard for polygons containing holes. For the hole-free case they present an $O(p^{1.5})$ algorithm. Bar-Yehuda and Ben-Hanoch [2] provide a linear time algorithm for such the case. Scott and Iyenger [13] present an algorithm to find, in $O(n \log n)$ time, a minimal cover for polygons with holes. However, their algorithm does not yield a globally minimum cover. Morita [11] presents a parallel algorithm, which finds a minimal (not minimum) square cover for such polygons. The sequential running time of this algorithm is $O(p)$.

The problem of covering polygons with a minimum number of rectangles has also been treated in several papers [3, 5, 7, 8, 10]. One application for it is the fabrication of masks for VLSI chips. Culberson and Reckhow [12] show that covering orthogonal polygons with the minimum number of ortogonal rectangles is NP-hard, even when the given polygon is hole-free. Levcopoulos [7] presents an algorithm which covers the polygon $P$ with $O(n \log n + \mu'(P))$ rectangles in $O(n \log n + \mu'(P))$ time, where $\mu'(P)$ is the minimum number of rectangles needed to cover $P$. In [8] a different heuristic is presented, guaranteeing an $O(\log n)$ approximation factor in polynomial time ($\Omega(n^6)$), provided that the vertices of the polygon have polynomially bounded integer coordinates.

In our paper [15] we consider the problem of covering polygons with a minimum number of squares. Our research was motivated by the paper of Levcopoulos and Gudmundsson [9]. They present two algorithms which cover an arbitrary polygon $P$, without any acute interior angles, with squares. Algorithm $A_1$ covers $P$ using $14\mu(P)$ squares in time $O(n^2 + \mu(P))$, where $\mu(P)$ is the minimum number of squares needed to cover $P$. Algorithm $A_2$ uses $12n + \mu(P)$ squares and the running time is $O(n \log n + \mu(P))$. In our paper we present algorithm which covers polygon $P$ with at most $10.5n + \mu(P)$ squares in time $O(n \log n + \mu(P))$.

## 2. THE VORONOI DIAGRAM OF P

Let $S$ be a set of points and segments in the plane, such that:

- segments intersect only at endpoints,
- two endpoints of each segment belong to the set $S$.

Every point is assigned to the nearest element from $S$. The set of points equidistant from at least two elements is the *generalized Voronoi diagram* $V(S)$. The diagram is the union of lines, half-lines, segments and sections of parabo

*Partially supported by Komitet Badań Naukowych, grant 8 T11C 044 19.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

Let $P$ be an arbitrary polygon without any acute interior angles, with $n$ vertices and $w$ concave vertices. $V(P)$ is the part of generalized Voronoi diagram $V(S)$ lying within $P$, where $S$ consists of all edges and vertices of $P$. $V(P)$ partitions $P$ into $n + w$ *regions*, where each segment and each concave vertex induces a Voronoi region. Every point lying in a region induced by $d$, where $d$ is either an edge or a concave vertex, lies at least as close to $d$ as to any other point of the boundary of $P$. Each *Voronoi circle* $c_V(p, r)$ with the center in $p \in V(P)$ and the radius $r$ equal to the shortest distance from $p$ to the boundary of $P$, lies entirely within $P$. The diagram can be computed in $O(n \log n)$ time [4, 6] for an arbitrary polygon and in linear time for a hole-free polygon [14].

## 3. PARTITIONING INTO CELLS

Let $f$ be a Voronoi region induced by $g \in S$. If $g$ is an edge of $P$, then we partition $f$ into cells by drawing segments, called *sides* of cells, which connect Voronoi vertices of $f$ with their perpendicular projections on $g$. The part of $g$ bounding the cell is called its *base*. If $g$ is a concave vertex of $P$, then we partition region $f$ by drawing segments connecting all its Voronoi vertices with $g$. A cell may be a triangle, a trapezoid, an area bounded by two segments and a part of paraboloid (the cell induced by a concave vertex), or an area bounded by three segments and a part of paraboloid (the cell induced by an edge of $P$). Note that every cell is bounded by a proper Voronoi edge.

Let $\mathcal{T}$ be the set of all trapezoidal cells in $P$. A cell $k \in \mathcal{T}$ is a *funnel*, if the angle between the proper Voronoi edge and the shorter side of the cell is smaller than $135^o$, and the length of its base is greater than the length of the shorter side; otherwise it is an *ordinary trapezoid*. If a cell $k \notin \mathcal{T}$, then $k$ is called a *non-trapezoid*. In this paper $\mathcal{F}$ denotes the set of all funnels in $P$ and $\mathcal{O}$ denotes the set of all ordinary trapezoids in $P$.

Two cells sharing a proper Voronoi edge are called a *pair of cells*.

THEOREM 1. *The number of proper edges in V(P) is at most 3n, where n is the number of vertices in the polygon P.*

COROLLARY 1. *The number of pairs of cells is not greater than 3n.*

We classify each pair of cells $k_1k_2$ as follows:

1. a pair of trapezoids; $k_1$ and $k_2$ are induced by edges of $P$, Fig. 1(a);
2. a pair of non-trapezoids:
   (a) a pair of triangles; $k_1$ and $k_2$ are induced by edges of $P$, Fig. 1(b);
   (b) a pair of triangles; $k_1$ and $k_2$ are induced by concave vertices of $P$, Fig. 1(c);
   (c) a pentagon; $k_1$ is induced by a concave vertex of $P$ and $k_2$ is induced by an edge of $P$, Fig. 1(d).

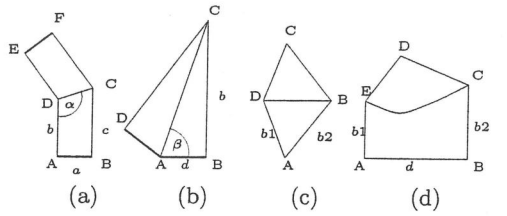



**Figure 1: (a) A pair of trapezoids. (b) Triangles induced by edges of $P$. (c) Triangles induced by concave vertices of $P$. (d) A pentagon.**

A *sequence of trapezoids* is a maximal sequence $p_1, \ldots, p_k$, $k \geq 2$ such that:

- $p_i \in \mathcal{T}$, $i = 1, \ldots, k$,
- the longer side of $p_i$ is the shorter side of $p_{i+1}$, $i = 1, \ldots, k-1$,
- $p_i \in \mathcal{F} \Rightarrow p_{i+1} \in \mathcal{O}$, $i = 1, \ldots, k-1$.

We say that a trapezoid $t$ is *adjacent* to a pair of non-trapezoids $n_1n_2$ if a side of either $n_1$ or $n_2$ is the shorter side of $t$. Even if two sides of $t$ have the same length, we call one of them the shorter side (we fix at the beginning which one is shorter). Note that at most four trapezoids may be adjacent to $n_1n_2$.

## 4. THE ALGORITHM

Our algorithm is based on the following scheme.

1. Compute $V(P)$.
2. Partition each Voronoi region into cells.
3. Assign each cell to one of the groups as follows:
   (a) Fix sequences of trapezoids and then divide each of them into groups consisting of two consecutive cells (if the number of trapezoids in the sequence is odd, then the last group has three cells).
   (b) Assign each non-trapezoid $n_1$ to the group induced by the pair of non-trapezoids $n_1n_2$ ($n_2$ shares with $n_1$ a proper Voronoi edge).
   (c) Assign the remaining trapezoids to one of the following group:
      i. if $t$ is adjacent to a pair of non-trapezoids $n_1n_2$, then add $t$ to the group induced by $n_1n_2$;
      ii. if $t$ is a funnel that has not been assigned to any sequence of trapezoids and any pair of non-trapezoids, then $t$ is a separate group.
4. Cover each group with squares.

The following lemma guarantees that each cell will be covered by our algorithm.

LEMMA 1. *Each cell belongs to exactly one of the groups.*

In the fourth step our algorithm covers groups of cells. At the beginning it covers all funnels, and then the remaining cells in the groups. The covering of funnels may require usage of large number of squares. Fortunately, we can make use of their appearance in the group and reduce the number of squares that cover the remaining cells. In [15] we give a detailed analysis of all the cases. We also prove that our algorithm uses $\mu(P)$ squares plus - perhaps - $O(n)$ additional squares. In average the surplus of squares is not greater than 3.5 per one pair of cells.

THEOREM 2. *Our algorithm covers $P$ with at most $10.5n+\mu(P)$ squares.*

There are two main reasons why our algorithm uses less squares than the algorithm presented in [9]. Firstly, we can better cover some pairs of non-trapezoids. Secondly, if we consider groups of cells, we can use a smaller number of larger squares that lie within the polygon $P$.

## 4.1 Time analysis

We can construct $V(P)$ in $O(n \log n)$ time, if $P$ is an arbitrary polygon [6, 4], and in $O(n)$ time, if $P$ is a hole-free polygon [14]. Because there are at most $6n$ cells (see Corollary 1), we can split Voronoi regions into the cells in $O(n)$ time, and then assign each cell to the group in $O(n)$ time.

In full version of our paper [15] we show that each group of cells may be covered in $O(s)$ time, where $s$ is the number of squares used for covering the considered group of cells. There are at most $6n$ cells, so $P$ may be covered in $O(n+s) = O(n+\mu(P))$ time.

# 5. REFERENCES


[1] L. J. Aupperle, J. M. Keil, J. O'Rourke. Covering orthogonal polygons with squares. In *26th Annual Conference on Communication, Control and Computing*, pages 97–106, 1988.

[2] R. Bar-Yehuda, E. Ben-Hanoch. A linear-time algorithm for covering simple polygons with similar rectangles. In *International Journal of Computational Geometry and Applications, vol. 6, no. 1*, 1996.

[3] B. M. Chazelle. Computational geometry and convexity. *Ph. D. Thesis, Department of Computer Science, Yale University, New Haven, C.T., Carnegie-Mellon Univ. Report CS-80-150*, 1979.

[4] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *2nd Annual ACM Symposium on Computational Geometry*, pages 313–322, 1986.

[5] A. Hegedus. Algorithms for covering polygons by rectangles. In *Computer Aided Design, vol. 14, no. 5*, 1982.

[6] D. G. Kirkpatrick. Efficient computation of continuous skeletons. In *20th Annual IEEE Symposium on Foundation of Computer Science*, pages 18–27, 1979.

[7] C. Levcopoulos. A fast heuristic for covering general polygons with rectangles. In *Fundamentals of Computer Theory, Cottbus, GDR, LNCS 199, Springer-Verlag*, 1985.

[8] C. Levcopoulos, J. Gudmundsson. Close approximation of minimum rectangular coverings polygons. In *FST-TCS'96, LNCS 1180, Springer-Verlag*, pages 135–146, 1996.

[9] C. Levcopoulos, J. Gudmundsson. Approximation algorithms for covering polygons with squares and similiar problems. In *RANDOM'97, LNCS 1269, Springer-Verlag*, pages 27–41, 1997.

[10] C. Levcopoulos, A. Lingas Covering polygons with minimum number of rectangles. In *Symposium of Theoretical Aspects of Computer Science, LNCS 166, Springer-Verlag*, pages 63–72, 1984.

[11] D. Morita. Finding a minimal cover for binary images: an optimal parallel algorithm. *Technical Report No. 88-946, Department of Computer Science, Cornell University*, 1988.

[12] R. A. Reckhow, J. Culberson. Covering a simple orthogonal polygon with a minimum number of orthogonally convex polygons. In *3rd Annual ACM Symposium on Computational Geometry*, pages 268–277, 1987.

[13] D. S. Scott, S. S. Iyenger. Tid: a translation invariant data structure for storing images. In *Communications of the ACM, vol. 29, no. 5*, 1986.

[14] J. Snoeink, C. A. Wang, F. Chin. Finding the medial axis of a simple polygon in linear-time. In *International Symposium on Algorithms and Computation, LNCS 1004, Springer-Verlag*, pages 382–391, 1995.

[15] G. Zwoźniak. A better approximation algorithm for covering polygons with squares. *Technical Report, Institute of Computer Science, University of Wrocław*, 1998.

# Approximation by skin curves

## [Extended Abstract]

Nico Kruithof *
Dept. of Mathematics and Computing Science
University of Groningen
The Netherlands
nico@cs.rug.nl

Gert Vegter
Dept. of Mathematics and Computing Science
University of Groningen
The Netherlands
gert@cs.rug.nl

## 1. INTRODUCTION

Approximation of simple closed curves in the plane by curves from a restricted class, like polygons or spline-curves, is a well-studied problem in geometric modeling. We present a method for approximating a simple closed curve in the plane by a skin curve. Skin curves, introduced by Edelsbrunner in [5], are defined by a finite set of weighted points, and have nice combinatorial properties related to the Voronoi, Delaunay and Alpha complexes of their defining set of points. These properties can easily be generalized to higher dimensions. In general, skin curves may be considered as the union of a set of circles with hyperbolic patches connecting two 'adjacent' circles. These patches are such that the curve is tangent continuous. Their geometric properties, like tangent continuity, continuous dependence on the defining set of points, and deformability, make this class of curves very attractive for modeling purposes. Furthermore, they have been used in morphing applications, cf [5, 3, 4].

The starting point of our approach is the observation that a regular, simple closed curve in the plane is the envelope of its maximal circles, see Figure 1(a). (In this paper, a curve is regular if it has a $C^2$ parametrization with nowhere vanishing derivative. It is simple if it has no self-intersections.) A maximal disk is a disk that lies inside the curve, and is not contained in a larger disk inside the curve. A maximal circle is the boundary of a maximal disk. The medial axis of a curve is the set of centers of its maximal disks. Equivalently, it is the closure of the set of points inside the curve, which have more than one closest point on the curve. The medial axis transform is the representation of the curve as the boundary of the infinite union of its maximal balls.

The medial axis transform, or, rather, an approximation of it, also plays a prominent role in methods for the reconstruction of suitably sampled curves and surfaces; See, e.g., [1]. From this sample Amenta et al. first determine an approximation of the medial axis transform, and subsequently compute a piecewise-linear approximation of the curve or surface. Our approach is different, in the sense that we start from a suitable approximation of the medial axis transform, from which we construct a $C^1$ (skin) approximation of the sampled curve.

It is intuitively clear that a sufficiently dense, but finite, sample of the set of maximal circles (the medial axis transform) of a curve will have a union, whose boundary is a good approximation of the curve, see Figure 1(b). In general, this boundary is not smooth, since it does not have a unique tangent at intersection points of two circles in the sample. However, the skin curve, associated with the set of circles, is a smooth curve that also approximates the input curve. As will be explained below, we have to determine a suitable shrink factor for the skin curve, and also blow up the input circles by a factor related to this shrink factor. The result of this procedure is depicted in Figure 1(c).

The main features of our approximation scheme are:

- The computed $C^1$-curve is an approximation of the original curve, which has smaller distance to the original curve than the union of the set of circles in the input.
- The approximation error decreases with increasing sample density.
- Existing methods for morping skin curves transfer to the input curve. (Using the medial axis for morphing purposes has also been suggested by Boissonnat in [2].)

In the next section we recall the definition of the medial axis and present some of its properties. In Section 3 we will introduce skin curves and show how they can be used to approximate a closed curve using the medial axis.

## 2. MEDIAL AXIS

The medial axis $\mathcal{M}$ of a simple closed curve is the locus of the centers of the maximal circles of the curve.

We also define a pointwise radius function $r : \mathcal{M} \to \mathbb{R}$ that associates the radius of the maximal circle corresponding

*Partially 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).

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

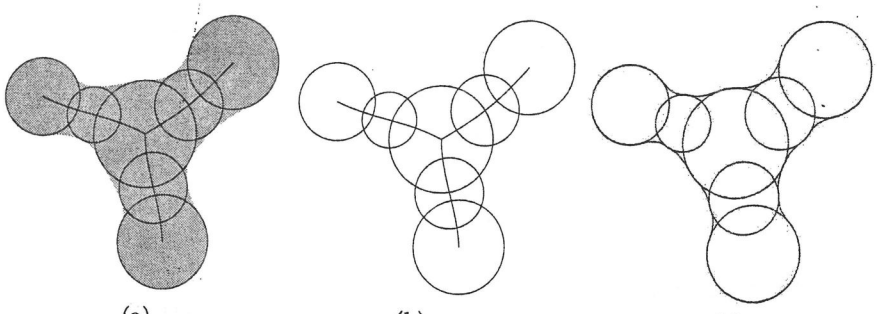



**Figure 1: (a) The medial axis and some maximal circles. (b) A finite set of maximal circles is the input of our algorithm. (c) The output is a skin curve approximating the input curve in Figure 1(a).**

to a point on the medial axis. The pair $(\mathcal{M}, r)$, called the medial axis transform of the curve, implicitely represents the curve.

According to [1] the medial axis of a set of points can be computed using the Voronoi diagram of the point set. A $C^2$ curve has a piecewise $C^2$ medial axis, cf [6]. Generically, the piecewise smooth parts of $\mathcal{M}$ end in a triple point or in a single point. The radius function $r : \mathcal{M} \to \mathbb{R}$ is $C^2$ on the smooth parts of the medial axis. An approximation of the curve is obtained by taking the boundary of the union of a finite subset of the maximal circles, whose centers form a sufficiently dense sample of $\mathcal{M}$. Starting from this finite set of maximal circles we construct a tangent continuous approximation in the class of skin curves, as explained in the next section.

## 3. APPROXIMATION USING SKIN CURVES

A weighted point $p$ in the plane is a pair $\hat{p} = (p, \mathrm{P})$ with location $p$ and weight $\mathrm{P}$. The weight $\mathrm{P}$ is not necessarily non-negative, but if so, the weighted point corresponds to a circle with center $p$ and radius $\sqrt{\mathrm{P}}$. The space of weighted points inherits a vector space structure from $\mathbb{R}^3$ via the bijective map $\Pi : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3$, defined by

$$\Pi(\hat{p}) = (\xi_1, \xi_2, |p|^2 - \mathrm{P}), \text{ with } p = (\xi_1, \xi_2).$$

See also [5]. The notion of *shrinking* and growing of circles can be defined in terms of this vector space structure. Given a shrink factor $s \in \mathbb{R}$, we associate with a weighted point $\hat{p}$ the shrunken weighted point $\hat{p}^s$, defined as follows. Let $\hat{p}' = (p, 0)$. Then

$$\begin{aligned} \hat{p}^s &= s \cdot \hat{p} + (1-s) \cdot \hat{p}' \\ &= \Pi^{-1}(s \cdot \Pi(\hat{p}) + (1-s) \cdot \Pi(\hat{p}')) \\ &= (p, s \cdot \mathrm{P}) \end{aligned} \tag{1}$$

If $\mathcal{Q}$ is a set of weighted points, we denote by $\mathcal{Q}^s$ the set obtained by shrinking every point of $\mathcal{Q}$ by a factor $s$. The skin curve $\mathrm{skn}^s\mathcal{P}$, associated with a set $\mathcal{P}$ of weighted points, is defined by

$$\mathrm{skn}^s\mathcal{P} = \mathrm{bd} \bigcup \mathtt{ucl}(\mathtt{conv}(\mathcal{P})^s). \tag{2}$$

Here $\mathtt{conv}(\mathcal{P})$ is the convex hull of a set of weighted points $\mathcal{P}$ and $\mathtt{ucl}(\hat{p})$ the upward closure, i.e., the set of weighted points $\{(p, \mathrm{P}') | \mathrm{P}' \leq \mathrm{P}\}$, which is the disk bounded by the weighted point $\hat{p}$. For the skin curve of two weighted points see Figure 2

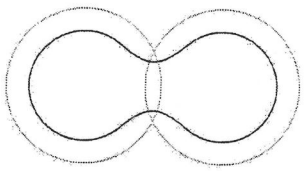


**Figure 2: Two weighted points and their skin curves for $s = 1/2$.**

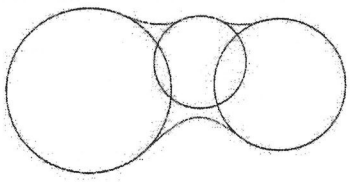


**Figure 3: A problem can occur when decreasing the shrink factor. Every circle on the medial axis should touch the skin curve on both sides.**

We now smoothen the boundary of the union of circles by growing hyperbolic caps over the points of intersection of adjacent circles, basically by applying a shrinking procedure. However, since we don't want to change the radii of the circular arcs that contribute to the skin curve, we first blow each circle (weighted point), in such a way that the shrunken version of the blown-up circle coincides with the original circle. More precisely, let $\mathcal{P}_0$ be the sample set of weighted points obtained from the medial axis and the pointwise radius function. Once we have determined the shrink factor $s$, the input set of weighted points for the skin curve is determined by $\mathcal{P} = \{\hat{p}^{1/s} | \hat{p} \in \mathcal{P}_0\}$.

So it remains to determine the only free parameter, viz the shrink factor $s$. Our goal is to minimize $s$ in order to minimize the maximal curvature. We minimize $s$ under the condition that the *shrunken circles are maximal circles of the skin curve.* Figure 3 illustrates a situation in which the shrink factor is too small, since the middle circle is not maximal.

Conceptually, we gradually decrease the shrink factor (starting from value 1) until a point in the (shrink factor dependent) input set of weighted points touches the skin curve in exactly one point. This is the minimal shrink factor for

which all points in the input set are in the medial axis of the skin curve.

A careful analysis of this situation at every point of the medial axis allows us to determine the minimal shrink factor globally. To do so, we assume that the medial axis is parametrized as a unit speed curve $t \mapsto \mathcal{M}(t)$, $t \in \mathbb{R}$. Let $r(t)$ be the radius of the maximal circe centerd at $\mathcal{M}(t)$. We assume that both $\mathcal{M}$ and $r$ are $C^2$ functions. Using a third order Taylor approximation of the medial axis and the pointwise radius function, we derive the following formula for the minimal shrink factor

$$\frac{\kappa(t)\sqrt{4r(t) - r'(t)^2} + r''(t)}{2}$$

where $\kappa(t)$ is the curvature of the medial axis at $\mathcal{M}(t)$ and $r'$ and $r''$ are respectively the first and second derivative of the pointwise radius function $r$.

**Remark.** We also developed a similar formula for three subsequent circles in the sampled subset of the medial axis. In that case we used the mixed complex as described in [5].

## 4. CONCLUSIONS AND FUTURE WORK

In this abstract, we discuss how skin curves can be used to approximate a closed curve and describe a way to find a local optimal shrink factor.

As we vary the shrink factor, the topology of the skin curve may change, which means that the skin curve and the original curve may not be homeomorphic. Future work is concerned with determining a minimal shrink factor for which the topology of the skin curve is similar to that of the original curve.

Another area of interest for future work is the extension of the algoritm to surfaces in three-space.

## 5. REFERENCES


[1] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Internat. J. Comput. Geom. Appl.*, 19:127–153, 2001.

[2] D. Attali and J.-D. Boissonnat. Complexity of the Delaunay triangulation of points on the polyhedral surfaces. Technical Report 4232, INRIA, 2001.

[3] S. Cheng, H. Edelsbrunner, and P. Fu. Shape space from deformation. *Report rgi-tech-98-011*, 1998.

[4] S. Cheng, H. Edelsbrunner, P. Fu, and K. Lam. Design and analysis of planar shape deformation. *Proc. 14th Ann. Sympos. Comput. Geom.*, 1998.

[5] H. Edelsbrunner. Deformable smooth surface design. *Discrete & Computational Geometry*, 21:97–115, 1999.

[6] P. Giblin and S. Brassett. Local symmetry of plane curves. *American Mathematical Monthly*, 92(10):689–707, 1985.

# An Application of Free Form Deformation to Phytoplankton Cells Modeling

Anton M. Lyakh
Dept. of Biophysical Ecology, Institute of Biology of the Southern Seas
2, Nakhimov av., Sevastopol, Crimea, 99011, Ukraine
+380-692-545945
anton@ibss.iuf.net, antonberlin@yahoo.com

## ABSTRACT

In the paper the main stages of the process of 3D phytoplankton model construction based on their digitized images is described. The process of 3D phytoplankton model creation consists of sequential scaling, rotating and fitting of the outline of a base model, which represents a phytoplankton species with average dimensions, to the cell outline on the digitized image. The model fitting is provided by Free Form Deformation (FFD), which is defined by parametric functions whose values are determined by the location of control points. In first case, when dimensions of an organism are given, the distances between control points are equated to the given dimensions. In second case, when a photo of an organism is given, a scientist should: (1) rotate and scale a base model to achieve a visual coincidence of its orientation and dimensions with the orientation and dimensions of species on the digitized photo, and (2) deform the base model by displacing control points of FFD. Our algorithm allows studying morphological changes of microorganisms and describing their shapes by control points coordinates.

## General Terms

Algorithms.

## Keywords

Free Form Deformation, Phytoplankton Species Modeling, Outline Fitting.

## 1. INTRODUCTION

Phytoplankton is tiny unieellular algae, which live anywhere in the water where there is enough of light. They are an important part of the World Ocean food web, without which the life in the ocean will die out. Tiny dimensions of algae (less than one mm) produce difficulties for their studying.

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

The volume and surface area of microorganism need to be measured practically at all microbiological researches. Both the tiny dimensions of phytoplankton cell and the complex shape of their body complicate calculation of the cell volume and surface area. Till now biologists approximate the shape of phytoplankton species by a set of simple geometrical primitives, such as a sphere, a cone, an ellipsoid, etc. (Figure 1). At such approach values of cell volume and surface area will be insufficiently exact. At the same time application of 3D modeling will allow to precisely approximate a shape of a cell and to receive more exact values of its volume and surface area.

The main stages of 3D phytoplankton cell construction are first described by Lyakh, et. al. in 2001 [1]. The main problem at 3D cell building is the fitting of a model shape to the shape of natural phytoplankton cell. For the solution of the given problem we determine the location of Free Form Deformation (FFD) control points on 3D model surface, and, displacing these points, fitting a model shape to the shape of a natural cell.

## 2. FREE FORM DEFORMATION

FFD is defined by parametric functions (3D splines) whose values are determined by the location of deformation lattice control points (Figure 2). Once control point is moved, the new

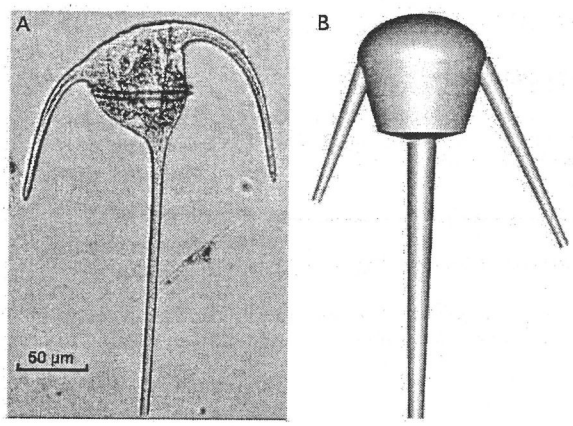



**Figure 1. The comparison of the *Ceratium tripods* phytoplankton cell body shape (A) with the 3D model created from different geometric primitives, which is used for the cell volume and surface area calculation (B).**

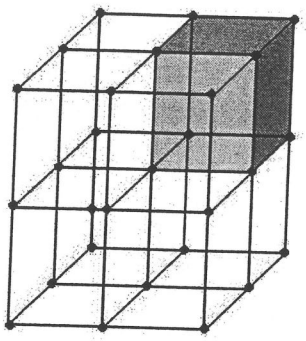


**Figure 2. The deformation lattice: the black circles are the control points (or the knots) of the deformation lattice; the gray box is the deformation cell.**

location of object vertexes is determined by the weighted sum of the control points [2].

The FFD is a mapping operating from the world space of a model to the local space of a deformation lattice, and conversely to the world space:

$$FFD: R^3 \rightarrow R^3 \rightarrow R^3. \quad (1)$$

Let function F: $R^3 \rightarrow R^3$ maps vertex $X = (x, y, z)$ of the object in the world space to vertex $U = (u, v, w)$ of an object in the space of deformation lattice. Let function F': $R^3 \rightarrow R^3$ maps vertex $U = (u, v, w)$ to vertex $X' = (x', y', z')$ of an object in the deformed world space. Then the composition of these functions defines FFD:

$$FFD(X) = F'(F(X)) = X'. \quad (2)$$

The form of a deformation lattice defines function F, which assigns a set of local coordinates to each object vertices. We use the parallelepiped-shape coordinate system of a deformation lattice. Therefore F is the transformation of the world coordinate system to the coordinate system of a parallelepiped, which sides are considered equal to unit length. The function F' is a sum of control points $P_{i,j,k}$ weighted by polynomial basis functions $B_i(\ )$:

$$F'(u,v,w) = \sum_{i=0}^{Ni} \sum_{j=0}^{Nj} \sum_{k=0}^{Nk} B_{i,Ni}(u) \cdot B_{j,Nj}(v) \cdot B_{k,Nk}(w) \cdot P_{i.j.k} \quad (3)$$

The basis functions B are defined as follows:

$$B_{i,N}(u) = C_N^i \cdot (1-u)^{N-i} \cdot u^i \quad (4)$$

FFD proceeds in three steps:

1. Local coordinates $(u,v,w)$ are assigned to each object vertex (F is applied). Since a local parallelepiped has sides with unit length, therefore local coordinates $(u,v,w)$ varies from 0 to 1.
2. Control points are displaced that cause the distortion of the local space. But the model local coordinates don't change. They never change

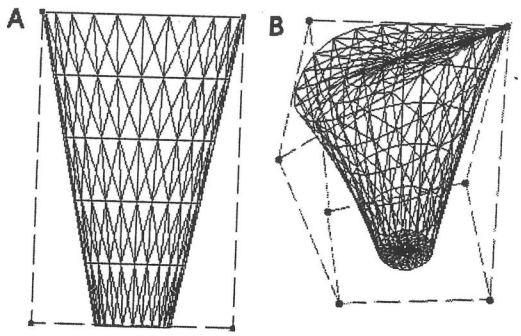



**Figure 3. FFD of 3D object. A – the cone; B – displacement of the control points (black circles) causes the deformation of the cone.**

3. Equation 3 is applied to all object vertices that produce the deformation of 3D model (Figure 3).

## 3. THE DEFORMATION OF A PHYTOPLANKTON MODEL

### 3.1 Location and Classification of Control Points

Initial location and amount of control points (CP) plays the defining role during the model deformation. When CP number increase the ability of a model to accept the necessary shape is increased. However, many CP will hamper deformation.

#### *3.1.1 Location of control points*

Location of many CP is chosen so that a phytoplankton model would be convenient for deforming. Control points are located on the following parts of a cell model:

- One control point is located at the concave and convex parts of a model, at the top of cell growth, etc. (Figure 4, A).
- Five CP are located on the zone where a cell element contacts with a cell body: one CP is located at the center of a contact zone; another four on the edges of a contact zone (Figure 4, B).
- A set of control points is uniformly distributed along the edges of a cell grooves and keels: two points are located at the groove/keel ends; two points on the groove/keel edges; one point at the hollow/crest; and the last one at the center of a formed triangle (Figure 4, C).

#### *3.1.2 Classification of control points*

The application of one FFD operator to whole phytoplankton model will essentially hamper the process of model deformation. Therefore, a phytoplankton model is divided into elements, and a single FFD operator is applied to each of them. Deformation of cell element consists of the distortion of an element surface and the deformation of an element medial axis. The distortion of an element surface is caused by the displacement of *an ordinary control point* (white circles on Figure 4). The deformation of an element medial axis is caused by the displacement of *the main control point* (black circles on Figure 4), which is situated at the

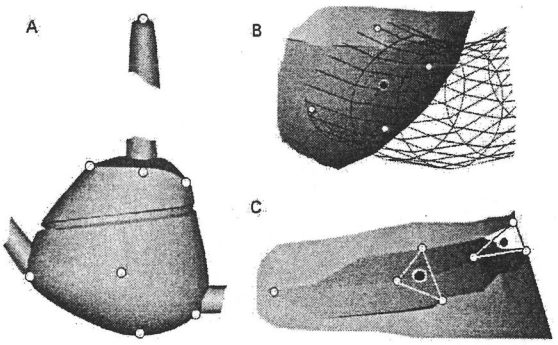



**Figure 4. The location of control points on different parts of phytoplankton cell: A – on the cell growth and body; B – at the zone of contact the cell element with the cell body; C – on the cell groove (the set of points, which are put along the groove edges, are delineated by triangles).**

center of contact zone, or along an axial line of cell groove/keel. The main control point is connected with ordinary control points, and the changing of the main CP position produces a shift of ordinary CP that causes the deformation of a model. The difference between *an ordinary* and *the main* CP is that the first one makes direct model deformation, and the second one makes indirect model deformation, by means of the first. Using the two types of control points facilitates 3D model deformation.

## 3.2 Construction of a Deformation Lattice

It is important to determine dimensions of a deformation cell so that the displacement of a control point does not cause an effect unnecessary to a scientist. If a scientist expects the deformation of certain model part, that this model part must be deformed.

Traditional methods of a deformation lattice construction not provide performance of this task. Traditional method assumes that deformation cells have equal dimensions, and they are uniformly distributed inside a parallelepiped, which bounds 3D model (Figure 2). The position of deformation lattice knots in similar construction not always coincides with the position of model control points. Therefore, for the displacement of these control points it is necessary to displace knots of a deformation lattice. But scientists must displace control points not knots. Therefore the position of knots must coincide with the position of control points. We use the following method of deformation lattice construction, which provides the coincidence of knots with control points:

1. A bounded parallelepiped is circumscribed around the model, and an arbitrary control point is selected.
2. Three planes, parallel to the parallelepiped sides, are drawn through the first control point up to the bounds of the parallelepiped (Figure 5, A).
3. Next control point is selected, and three planes, parallel to the parallelepiped sides, are drawn through this control point. The planes are drawn up to intersection with other planes (Figure 5, B).
4. Step three is repeated for each control point.

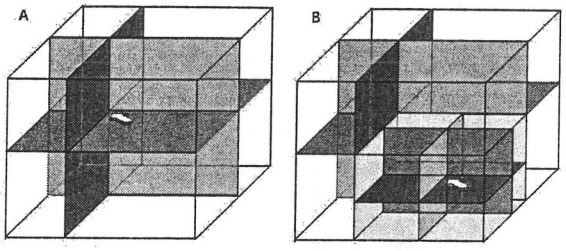



**Figure 5. The deformation lattice construction. White arrows show the active control point. The points of lines intersection are knots, which cannot be displaced by a scientist. A – the first step: three planes are passed through the CP till they reach the bounding parallelepiped; B – the following steps: three planes are passed through the CP till they reach another planes.**

During the deformation lattice construction new knots are created (the points of lines intersection on the Figure 5, which are not marked by arrows). But these knots are not control points. A scientist cannot displace they. They only determine the vertices of a deformation cell.

## 3.3 Two Methods of Phytoplankton Model Deformation

FFD procedure is used at the construction of a phytoplankton model of a cell observed through a microscope. The digitized image of phytoplankton cell is used as a pattern of its shape. The shape of a cell model is deformed till its outline coincides with the outline of a pattern. The process of a model deformation consists of sequential movements of control points (an application of FFD operator). In many cases it is impossible to obtain the digitized image of a phytoplankton cell. Therefore, initial data determines a method of model deformation.

In case when a scientist cannot obtain the image of a phytoplankton cell, but can measure its dimensions, the measured dimensions are used as the values of distances between control points of a model. Setting of distances causes the deformation of a model.

In case when a scientist obtains the digitized image of a phytoplankton cell, he should achieve coincidence of the model borders with the outline of a microorganism on the image. The coincidence is achieved by an application of FFD, i.e. by the displacing model control points.

These two methods are the main methods of the FFD operator application to the phytoplankton cell modeling (Figure 6). Initially a scientist has *a phytoplankton cell base model*, which represents an average shape of a single phytoplankton species or genus. After that a scientist gets the initial data (the dimensions or the digital microphotograph of a phytoplankton cell), and deforms a base model, which corresponds to observed species, that results to obtaining the model of the specific phytoplankton species. Using this specific model a scientist calculates the volume and the surface area of a phytoplankton cell and makes its morphological analysis. Undoubtedly, calculated surface and volume values will be most exact.

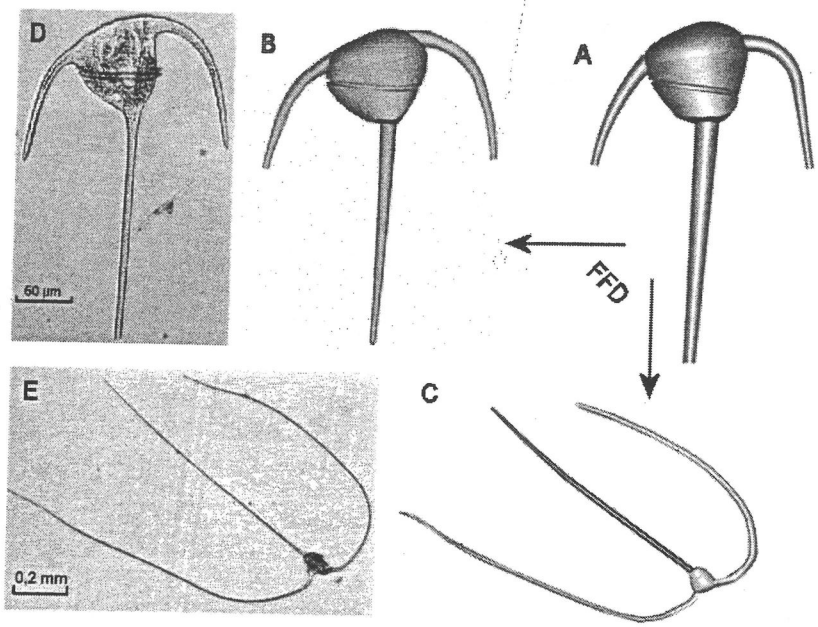



**Figure 6. Some examples of using FFD operator to phytoplankton cell modeling: A – the base model, which is deformed by a scientist; B,C - 3D models of phytoplankton cell created using FFD; D,E – the microphotographs of phytoplankton cells *Ceratium tripods* (D), and *Ceratium sp.* (E), which are patterns for the base model fitting.**

## 4. CONCLUSION

The considered procedure is the main stage of the 3D phytoplankton cell building process. The application of 3D simulation in biology allows receiving exact evaluations of a microorganism volume and surface area. The application of FFD to 3D phytoplankton model allows demonstrating the microorganism morphological changes during its life cycle. The location of the phytoplankton model control points describes a shape of the cell, and it may be used at cell morphological analysis.

## 5. ACKNOWLEDGMENTS

Research partially supported by INTAS grant No. 99-10390. My special thanks to Vladimir Mukhanov, who has helped me during thinking over this paper.

## 6. REFERENCES

[1] Lyakh, A. M., Mukhanov, V. S., and Kemp, R. B. The Virtual Cell Project for the Investigation of Microalgal Morphology and Dispersity of Natural Phytoplankton in 17th European Workshop on Computational Geometry, Program & Abstracts (Berlin, March 2001), Berlin, 57-58.

[2] Sederberg, T.W., Parry, S.R. Free-Form Deformation of Solid Geometric Models. Computer Graphics, 20 (1986), 151-160.

# Isomorphic–free generation of some classes of triangulations without repetitions *

[Extended Abstract]

Lyuba Alboul
University of Twente
P.O. Box 217
7500 AE Enschede
the Netherlands
alboul@math.utwente.nl

Alexandre Netchaev
University of Twente
P.O. Box 217
7500 AE Enschede
the Netherlands
netchaev@math.utwente.nl

## ABSTRACT

We present an algorithm to generate all non–isomophic dissectible polyhedra without repetitions. Then we discuss a generalisation of this algorithm for the following class of simplicial polyhedra: 4–connected simplicial polyhedra.

## Keywords

Triangulation, graph isomorphism, group automorphism

## 1. GENERAL DESCRIPTION

This work contributes to the research direction, which concerns enumeration and generation of various types of objects (graphs, polyhedra, triangulations etc.) We are able to generate explicitly non-isomorphic simplicial polyhedra of the following two types: dissectible polyhedra and 4–connected simplicial polyhedra - without repetitions. They are also special types of the triangulations of the 2-sphere. No control on isomorphism is required, and there is only a very restricted check on automorphism. Our approach can be called "voluminous", because we deal with tetrahedra or pentahedra as generating blocks. We came across this approach while trying to construct a reasonable initial $2.5D$ (closed) triangulation from scattered $3D$ data. Our method might also be useful in simulations of growing processes of certain structures (molecular structures, corals, fractals). There are several algorithms to enumerate and/or generate non–isomorphic triangulations of the 2-sphere (or some classes of those triangulations), but it seems there is yet no one that completely avoids the check on isomorphism [1, 2, 3]. One of the first recursive methods for generating triangulations of the 2-sphere was developed by Robert Bowen and Stephen Fisk in 1967 [2]. They developed an algorithm for constructing all (non isomorphic) triangulations of the 2-sphere with $N$ vertices from those with $N-1$ vertices. Let $T$ be a triangulation with $N$ vertices, $E$ edges, $F$ faces. It is easy to see that there is only one triangulation with 4 vertices and one with 5. Let us denote the degree of vertex $V$ (its valency) by $d(v)$ and the number of vertices in $T$ with this degree by $V_{d(v)}$ By using the Euler formula:

$$N - E + F = 2$$

and the following evident equation:

$$\sum_{v \in N} V_{d(v)} d(v) = 2E;$$

we get:

$$\sum_{v \in N} V_{d(v)} (6 - d(v)) = 12$$

From the last equation we can easily deduce the following statement: for any triangulation of the 2–sphere there is at least one vertex with degree three, four or five, because the left side of the equation must be positive. On the basis of this statement Bowen and Fisk introduced three operations whose applications to all possible triangulations with $N-1$ vertices yield all triangulations with $N$ vertices. These operations consist of inserting a new vertex of degree three, four or five only to an appropriate triangulation with $N-1$ vertices. Namely, a vertex of degree three can be added to all triangulations, of degree four - only to triangulations with the property that the minimal valency of all vertices is equal or larger than 4, and a vertex with degree 5 - only to triangulations with the property that the minimal valency of all its vertices is equal to 5. We have to add the vertex of degree 5 in such a way that the minimal valency 5 of all the vertices is kept. Therefore, we can distinguish the following three operations:

- Operation 1. Adding a new vertex of degree three. We add a new vertex by connecting it to the three vertices of the same face.
- Operation 2. Adding a new vertex of degree four. We add a new vertex by connecting it first to the three vertices of some face as in the previous case, and then

*The research of the both authors is partially supported by the NWO (STW) (Dutch Organization for Scientific Research), project No. TWI4816

*The Eighteenth European Workshop on Computational Geometry (EWCG 2002) April 10-12, 2002, Warsaw, Poland*

connecting it to the one of the three remaining vertices of the adjacent faces. The internal edge, which is formed as a result of this procedure, is deleted.

- Operation 3. Adding a new vertex of degree five. First we implement Operation 2 (for appropriate triangles) and then connect a new vertex to the fifth vertex, which is the one of the four remaining vertices of the adjacent faces. The minimal valency of the vertices may not decrease. The formed internal edge is deleted.

We must keep, of course, in mind, that those faces of the previous triangulation to whose vertices new vertices were connected, are not more the faces of the new triangulation.

The procedure to construct all non-isomorphic triangulations with $N$ vertices then is the following: first apply the three operations in an appropriate way to all triangulations with $N-1$ vertices and then check the obtained triangulations on isomorphism.

In our approach we prefer to avoid any check on isomorphism. For any triangulation with $N$ vertices the correspondent graph is determined, and on the base of this graph we can define the groups of automorphisms of the triangulation. We simplify this procedure by introducing the labelling of vertices in a special way.

To a triangulation $T$ of the 2-D sphere corresponds a certain polyhedron $P$. It is easy to see that the insertion of a vertex of degree three in $T$ is equivalent to the procedure of adding a new tetrahedron to $P$, which has precisely an exterior triangle in common with $P$.

As we deal with non-isomorphic triangulations, the positions of vertices have no role, and we can deform our triangles as we like. Hence, we can form from two adjacent triangles a square, and from three - a pentagon. Consequently, the insertion of a vertex of degree 4 in $T$ is the same as the addition to $P$ a 4–pyramid with four triangles as the lateral faces and a square as the base (a pentahedron). The base of the pyramid is "glued" to two adjacent triangles in $P$. The adjacent edge of these two triangles (the " extra" edge) is deleted. The insertion of a vertex of degree 5 is the same as the addition to $P$ a 5–pyramid with five lateral faces–triangles and the base as a pentagon (a hexahedron). Two "extra" edges are deleted.

By adding a tetrahedron, a 4–pyramid and a 5–pyramid in a different way and in a different order, one can obtain all non-isomorphic polyhedra (triangulations of the 2–sphere). A tetrahedron is "glued" to a polyhedron with $N-1$ vertices along a triangle, which we call a *3-cut*. Consequently a 4–pyramid is "glued" along a *4-cut*, and a 5–pyramid - along a *5-cut*. Hence, a polyhedron can be constructed via a sequence of 3–, 4–, and 5–cuts. Unfortunately, we were unable to avoid the check on isomorphism.

We changed our strategy and decided to try to generate triangulations (polyhedra) of three different classes: those that can be generated only from tetrahedra, those – from 4–pyramides, and those from 5–pyramides.

We were able to generate all triangulations of the first class without any repetition. The method consists of inserting simultaneously a certain number of vertices (or adding several tetrahedra). This number is not less than $k$, where $k$ is the number of vertices of degree three in a triangulation $T_N$ with $N$ vertices. We add simultaneously $l$ new vertices to a triangulation $T_N$ with $N$ vertices where $k \leqslant l \leqslant 2(N-2)$. $k$ vertices must be always added to each star of a vertex with degree 3 (to one of the three faces of the star) in $T_N$. The remaining $l-k$ vertices are added to other faces of $T_N$. This approach can be generalized in a peculiar way to the next class of polyhedra (generated from 4–pyramids). All details are presented in the full version of the paper.

## 2. REFERENCES


[1] L. Beineke and R. Pippert. Enumerating dissectible polyhedra by their automorphism groups. *Can. J. Math.*, 26(1):50–67, 1974.

[2] R. Bowen and S. Fisk. Generation of triangulatios of the sphere. *Maht. Comp.*, 21:250–252, 1967.

[3] G. Brinkmann and B. McKay. http://cs.anu.edu.au/ bdm/plantri/. November 2001.

# Author Index

## List of Participants

| | |
|---|---|
| Oswin Aichholzer | oaich@igi.tu-graz.ac.at |
| Lyuba Alboul | alboul@math.utwente.nl |
| Mattias Hans Andersson | mattias@cs.lth.se |
| Francois Anton | fanton@cs.ubc.ca |
| Franz Aurenhammer | auren@igi.tu-graz.ac.at |
| Peter Brass | brass@inf.fu-berlin.de |
| Natalia de Castro | natalia@us.es |
| Jae-Sook Cheong | jaesook@cs.uu.nl |
| Otfried Cheong | otfried@cs.uu.nl |
| Eric Colin de Verdiere | Eric.Colin.de.Verdiere@ens.fr |
| Mark de Berg | markdb@cs.uu.nl |
| Annette Ebbers-Baumann | ebbers@informatik.uni-bonn.de |
| Herbert Edelsbrunner | edels@cs.duke.edu |
| Eti Ester Ezra | estere@post.tau.ac.il |
| Clara I Grima | grima@us.es |
| Joachim Gudmundsson | joachim@cs.uu.nl |
| Dan Halperin | danha@post.tau.ac.il |
| Michael Hoffmann | hoffmann@inf.ethz.ch |
| Frank Hoffmann | hoffmann@inf.fu-berlin.de |
| Samuel Hornus | Samuel.Hornus@imag.fr |
| Ferran Hurtado | hurtado@ma2.upc.es |
| Jerzy W. Jaromczyk | jurek@cs.uky.edu |
| Tom Kamphans | kamphans@cs.uni-bonn.de |
| Rolf Klein | rolf.klein@uni-bonn.de |
| Christian Knauer | knauer@inf.fu-berlin.de |
| Vladlen Koltun | vladlen@tau.ac.il |
| Mirosław Kowaluk | kowaluk@mimuw.edu.pl |
| Hannes Krasser | hkrasser@igi.tu-graz.ac.at |
| Nico Kruithof | nico@cs.rug.nl |
| Elmar Langetepe | elmar.langetepe@informatik.uni-bonn.de |
| Laura Nicoleta Heinrich-Litan | litan@inf.fu-berlin.de |
| Anton Mikhaylovich Lyakh | anton@ibss.iuf.net |
| Zbigniew Marciniak | zbimar@mimuw.edu.pl |
| Alberto Márquez | almar@us.es |
| Alexandre Netchaev | netchaev@math.utwente.nl |
| Colm O'Dúnlaing | odunlain@maths.tcd.ie |
| Mohammadreza Razzazi | razzazi@ce.aku.ac.ir |
| Carlos Seara | seara@ma2.upc.es |
| J. Antoni Sellarès | sellares@ima.udg.es |
| Micha Sharir | sharir@cs.tau.ac.il |
| Bettina Speckmann | speckman@inf.ethz.ch |

| | |
|---|---|
| Andreas Spillner | spillner@minet.uni-jena.de |
| Gert Vegter | gert@cs.rug.nl |
| Pawel Winter | pawel@diku.dk |
| Alexander Wolff | awolff@uni-greifswald.de |
| Grażyna Zwoźniak | grazyna@ii.uni.wroc.pl |