(CG'94)
10TH EUROPEAN WORKSHOP ON COMPUTATIONAL GEOMETRY

Santander, March 17-18, 1994

Editors:
M. Mazón, T. Recio and F. Santos

UNIVERSIDAD DE CANTABRIA
DEPARTAMENTO DE MATEMÁTICAS, ESTADÍSTICA Y COMPUTACIÓN

Universidad de Cantabria

Aula "Leonardo Torres Quevedo"
E. T. S. de Ingenieros de Caminos, Canales y Puertos

(CG'94)
10th European Workshop on Computational Geometry

Santander, March 17-18, 1994

Editors: M. Mazón, T. Recio and F. Santos

Departamento de Matemáticas, Estadística y Computación

Published by Departamento de Publicaciones

ISBN: 84-8102-054-0
D. L.: 38 - 1994

Foreword

This document contains the abstracts of the works presented at the Tenth European Workshop on Computational Geometry (CG'94), held in Santander (Spain), March 17th-18th, 1994.

This year, for the first time, this Workshop takes place in Spain, confirming the international projection of the spanish research in Computational Geometry. We hope that this fact contributes to increase even more the interest of the young spanish community.

In the other hand we are sure that this event will serve to strengthen the interaction between all the european people working in the subject as the previous ones did, promoting the growth of activity in this area.

The organizing commitee wishes to thank all the participants as well as all the people who contributed to make possible this meeting.

Organizing Commitee

Marisa Mazón
Tomás Recio
Francisco Santos

10th European Workshop on Computational Geometry (CG'94)

University of Cantabria. March 17-18, 1994

Contents and Program

Hamiltonian Abstract Voronoi Diagrams in Linear Time

Rolf Klein * Andrzej Lingas †

December 13, 1993

Abstract

Let H be an unbounded simple curve in the plane such that wherever we start walking along the curve, the distance to the starting point is strictly increasing. We show that the Voronoi diagram of any set of points on H can be computed in linear time, if the points are given in their order on H. This holds for any "nice" metric, in particular for any semialgebraic convex distance function. As a special case, it holds for $x-y$-monotone curves under the Euclidean metric, thereby generalizing previously known linear algorithms for the Voronoi diagrams of convex polygons and monotone histograms.

The above result follows from a more general theorem proved in this paper. Let $V(S)$ be an abstract Voronoi diagram, and let H be an unbounded simple curve that visits each of its regions exactly once. Suppose that each bisector $B(p,q)$, where p and q are in S, intersects H only once. Such a "Hamiltonian" diagram $V(S)$ can be constructed in linear time, given the order of Voronoi regions of $V(S)$ along H.

Key words: Computational geometry, Voronoi diagrams, abstract Voronoi diagrams, convex polygons

1 Introduction

The Voronoi diagram of a set S of n sites is one of the most useful structures in computational geometry. It partitions the plane into regions, one to each

*FernUniversität Hagen, Praktische Informatik VI, Elberfelder Straße 95, 58084 Hagen. e-mail: rolf.klein@fernuni-hagen.de. This work was partially supported by the Deutsche Forschungsgemeinschaft, grant Kl 655/2-2. It was begun during the first author's visit at Lund University in March 1993.

†Lund University, Department of Computer Science. e-mail: andrzej@dna.lth.se.

site p in S containing all points that are closer to p than to any other site in S. Shamos and Hoey [13] proved the construction of the Euclidean Voronoi diagram to be of complexity $\Theta(n \log n)$, giving a divide-and-conquer algorithm. Later, Fortune [8] introduced an $O(n \log n)$ sweep line algorithm, and Clarkson and Shor [4] presented a randomized incremental construction technique.

Voronoi diagrams can be generalized in many ways. Unifying concepts have been suggested by Edelsbrunner and Seidel [7] and Klein [9]. In the latter approach, the definition "point x is closer to site p than to site q" is replaced by "point x lies on the p-side of the bisector, $B(p,q)$, of p and q", this way abstracting from the concepts of both sites (as physical objects) and distance. For the resulting class of *abstract* Voronoi diagrams an optimal divide-and-conquer algorithm was given in [9], that works if the bisector of the two sets of sites generated by the divide step does not contain loops. An $O(n \log n)$ randomized algorithm that works without such assumptions has been presented in Klein, Mehlhorn, and Meiser [10].

It is very natural to ask whether Voronoi diagrams can be computed faster if more information about the position of the sites is available. A famous open problem mentioned by Preparata and Shamos [12] addresses the vertices of a convex polygon, given in cyclic order. It has been solved by Aggarwal, Guibas, Saxe, and Shor [1] by providing an $O(n)$ algorithm. Their algorithm first applies the geometric lifting mapping suggested in [7] to a paraboloid in 3D, and then constructs the convex hull of the transformed points. The dual of their (lower) convex hull gives the desired Voronoi diagram, after taking the projection to the plane. In their paper [5], Djidjev and Lingas show that the same algorithm applies to the Voronoi diagram of a monotone histogram, i.e. to points on an $x - y$-monotone curve.

The transformation to 3D makes the algorithm of [1] technically somewhat complicated. In [14] Yap and Alt gave an algorithm for the medial axis of a convex polygon under a polygonal convex distance function, that uses the same idea but works directly in the plane.

However, our approach is the first to show that the kernel of this powerful method is not really based on geometric properties, as suggested by the convexity assumption, but only on combinatorial properties of the underlying bisector system.

By working with abstract diagrams, we are able to cover situations more general than convex point sets. All we need to assume is the following. The sites are given in the order in which their regions in $V(S)$ are visited by an unbounded simple curve H that passes through each Voronoi region exactly

once. Also, for each subset S' of S, curve H is assumed to visit each region of $V(S')$ once. In presence of the former condition, the latter is equivalent to saying that each bisector $B(p,q)$, where $p,q \in S$, crosses H only once; see Lemma ??. We call a such a Voronoi diagram *Hamiltonian with respect to* H. On either side of H, its structure is that of a forest of binary trees (and possibly some halflines). We show that a Hamiltonian diagram can be computed in linear time, given the order of regions along H. As usual, elementary operations (like computing the intersection of two bisectors) are charged with unit cost in our model.

There are interesting examples of Hamiltonian diagrams. Let H be an unbounded curve, and let d denote a distance measure. Suppose curve H has the following property. When we start from an arbitrary point p on H and walk along H in either direction, then the d-distance from our current position to p is strictly increasing. Given a sequence of points on H, their d-Voronoi diagram is Hamiltonian with respect to H, so our result implies that it can be constructed in linear time. This holds for a large class of distance measures d including all semialgebraic convex distance functions.

Any $x-y$-monotone curve has the above property with respect to the Euclidean distance. Thus, our result includes the linear algorithms for convex polygons [1] and monotone histograms [5]. Yet, there are more curves H that have the increasing distance property without being monotone, for example the graph of the sinus function, or, more generally, any graph of a function whose derivative has an absolute value less or equal than 1 at each point.

The paper is organized as follows. After briefly introducing abstract Voronoi diagrams in Section ??, we prove the main result in Section ??. Finally, in Section ?? we give criteria for a curve to be Hamiltonian.

References

[1] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor. A Linear-Time Algorithm for Computing the Voronoi Diagram of a Convex Polygon. Discrete and Computational Geometry 2, 1987, Springer Verlag.

[2] F. Aurenhammer. Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys, 23(3):345–405, 1991.

[3] L. P. Chew and R. L. Drysdale III. *Voronoi diagrams based on convex distance functions.* In *Proceedings 1st ACM Symposium on Computational Geometry,* 1985, pages 235–244.

[4] C. Clarkson and P.W. Shor. Applications of Random Sampling in Computational Geometry II. Discrete and Computational Geometry, 4:387–421, 1989.

[5] H. Djidjev and A. Lingas. On Computing the Voronoi Diagram for Restricted Planar Figures. To appear in Proc. WADS'91, LNCS, Springer Verlag.

[6] O. Devillers. Randomization Yields Simple $O(n \log^* n)$ Algorithms for Difficult $\Omega(n)$ Problems. International Journal of Computational Geometry & Applications 2(1), pp. 97–111, 1992.

[7] H. Edelsbrunner and R. Seidel. Voronoi Diagrams and Arrangements. Discrete and Computational Geometry, 1:25–44, 1986.

[8] S. Fortune. A sweepline algorithm for Voronoi diagrams. Algorithmica, 2 (2), pp. 153–174, 1987.

[9] R. Klein. Concrete and Abstract Voronoi Diagrams. LNCS 400, Springer Verlag, 1989.

[10] R. Klein, K. Mehlhorn, and S. Meiser. Randomized Incremental Construction of Abstract Voronoi Diagrams. Computational Geometry: Theory and Applications 3, 1993, pp. 157–184.

[11] S. Meiser. Zur Konstruktion abstrakter Voronoidiagramme. Ph.D. Thesis, Universität des Saarlandes, Saarbrücken, 1993.

[12] F.P. Preparata and M.I. Shamos. Computational Geometry: An Introduction. Texts and Monographs in Theoretical Computer Science, Springer Verlag, New York, 1985.

[13] M.I. Shamos and D. Hoey. Closest Point Problems. In Proc. of the 16th Annual IEEE Symposium on Foundations of Computer Science, pp. 151–162, 1975.

[14] C. Yap and H. Alt. Motion Planning in the CL-Environment. in F. Dehne, J.-R. Sack, and N. Santoro (eds.), Algorithms and Data Structures (WADS '89), LNCS 382, Springer Verlag, pp.373–380.

[15] C. Yap. An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. Discrete and Computational Geometry 2, pp. 365–393, 1987.

Divide-and-Conquer Voronoi Diagram Construction

EXTENDED ABSTRACT

Lucian CUCU Mircea DRAGAN
Tudor JEBELEAN Viorel NEGRU

—

RISC–Linz
A-4040 Linz, Austria
mdragan@risc.uni-linz.ac.at

December 12, 1993

Abstract

We parallelize a wrap-around algorithm for planar Delaunay triangulation using the divide-and-conquer technique. The Voronoi diagram can be constructed simultaneously because the data structure we use represents them both. The implementation on shared memory architecture performs well on large number of points (more than 10,000) but the efficiency is high only on small number of processors (less than 8).

1 Divide and conquer

We present the algorithm for the Delaunay triangulation, and after this we explain how the Voronoi diagram can be constructed in the same time.

There are three distinct steps:

1. *preprocessing* of the data;
2. *divide* the set of points into two parts and *conquer*, that is construct the triangulation of each subset independently;
3. *merge* the two previous Delaunay triangulations.

Steps 2 and 3 can be applied recursively:

- at step 2 the triangulation of each subset can be done by further divide and conquer;
- at bottom-level of the recursion a sequential *local triangulation* is performed;
- at top-level of the recursion step 3 will finish the complete triangulation of the given set of points.

The *parallelization* of this algorithm is done by performing the conquering in parallel after each divide.

The **preprocessing** step is similar to the one described in [Fang and Piegl, 1993] and it consist in finding the smallest rectangle which contains all the sites, and then constructing an uniform grid by dividing this rectangle in small squared *boxes* of same size.

The **divide** step consists in splitting the grid into two sub-grids, named *hypercucs* in [Cucu *et al.*, 1993]. We **conquer** each hypercuc by constructing a Delaunay triangulation of its points, such that all the cells will be valid in the final triangulation. This is done either by further divide-and-conquer, or (at the bottom level of the recursion), by using a modified version of the sequential algorithm of [Fang and Piegl, 1993], also described in [Cucu *et al.*, 1993]. This algorithm starts by finding a *first-point* and a *second-point* which define a first edge of a Delaunay triangle, and then proceeds iteratively by finding a *third-point* corresponding to each existing oriented edge, such that they form a valid Delaunay cell. The algorithm terminates when no *third-points* can be found anymore.

The details of the old sequential algorithm had to be modified in order to find only valid Delaunay cells, although it does not "see" all the points in the given set. Essentially, a new test for detecting the *local boundary* situation is introduced, which consists in intersecting the circumcircle of a new-found Delaunay sell with the boundary of the hypercuc.

The **merging** step consists in constructing the Delaunay triangulation for the subset of points in two adjacent hypercucs, using the two partial triangulations received from the recursive divide-and-conquer step and possibly some isolated sites which are not included in these triangulations.

The basic triangulation algorithm described above is used, however a special scheme is devised for *unifying* the two triangulations when they "touch", since such a situation cannot be encountered in the sequential local-triangulation process.

The basis for construction of Voronoi diagram in the same time with the Delaunay triangulation is the fact that the centers of the circumcircles of the Delaunay cells are the vertices of the Voronoi cells. Moreover, two centers corresponding to adjacent cells determine a Voronoi edge. Hence, whenever a new Delaunay cell is found, one can construct the Voronoi edges corresponding to this cell and its neighboring cells. No computation overhead is introduced by this process, because:

- the neighbors of a new found Delaunay cell are also adjacent in the data structure;
- the coordinates of the centers of the Delaunay triangulation have to be computed anyway during the triangulation process.

2 Parallel implementation

The main advantage of the divide-and-conquer method presented in this paper is that the parallel tasks do not access the same points simultaneously, and this eliminates the need of monitors or message-passing during the triangulation of each hypercuc. This is to be contrasted with the approach used in [Popescu, 1993], where **each access** to the data-structure has to be monitored in order to avoid data inconsistencies. Also, in [Popescu, 1993] an intricate scheme for the inter-synchronization of processes has to be used in order to avoid conflicts between tasks.

The divide-and-conquer algorithm for Delaunay triangulation and Voronoi diagram construction was implemented on a Sequent Symmetry parallel machine (shared memory MIMD

architecture). We used C-language and the μSystem library for parallel processing (see [Buhr *et al.*, 1991]).

The running time of the program for 90,000 points is 1,862 seconds (sequential), 791 *s* (2 processors), 638 *s* (4 processors) and 588 *s* (8 processors). One can see that the efficiency of the 2-processor implementation is superlinear, but on more processors the efficiency decreases sharply. The explanation of this decrease resides in the performance of the sequential wrap-around triangulation technique, which decreases when the area where the points lie does not have a round shape.

Hence, it seems that for an efficient implementation of a divide-and-conquer method for Delaunay triangulation, one has to modify the triangulation algorithm during the *merging* step. Most promising seems to be a version of the algorithm presented in [Dwyer, 1991], which has a linear-time theoretical performance.

References

[Buhr *et al.*, 1991] P.A. Buhr, H.I. Macdonald, and R.A. Stroobosscher. *μSystem* Annotated Reference Manual. Version 4.4.1. Technical report, October 14 1991.

[Cucu *et al.*, 1993] L. Cucu, M. Dragan, T. Jebelean, V. Negru, and I. Popescu. Construction of Voronoi diagrams. Technical Report 52, RISC-Linz, A-4040 Linz, Austria, September 1993.

[Dwyer, 1991] R.A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discret Comput. Geom.*, 6:343–367, 1991.

[Fang and Piegl, 1993] T.-P. Fang and L.A. Piegl. Delaunay triangulation using a uniform grid. *IEEE Computer Graphics & Applications*, may:36–47, 1993.

[Popescu, 1993] I Popescu. Construction of Delaunay triangulation and Voronoi Diagram. Efficient Sequential and Parallel Implementation . Technical Report 54, RISC-Linz, A-4040 Linz, Austria, October 1993.

Visualization of Voronoi Diagrams Generated by VORONOI2

Waldir L. Roque*
Instituto de Matemática
Universidade Federal do Rio Grande do Sul
91501-970 Porto Alegre, RS
Brazil

Clovis Maliska Jr.
SINMEC - Dept. Engenharia Mecânica
Universidade Federal de Santa Catarina
88040-970 Florianópolis, SC
Brazil

Extended Abstract

Introduction: The geometric construction of Voronoi diagrams (VD) is one of the most fundamental concepts in Computacional Geometry [?]. Its powerfullness has been extensively demonstrated in many successful applications in a variety of problems in different areas.

Several approachs and algorithms have been proposed and implemented to generate planar Voronoi diagrams. Among them we can cite, for example, divide-and-conquer algorithm [?], sweepline algorithm [?], spiral-search algorithm [?], three-dimensional convex hull algorithm [?] and incremental-type algorithm [?]. In general, their implementation has been done through a numerical language which, by its very nature, are based on ordinary floating-point arithmetic.

One recent successful generation of planar Voronoi diagrams was presented by Sugihara and Iri [?] mainly based on an *incremental-type algorithm*. Their approach has been implemented in a program called VORONOI2 [?], which is able to generate a set of data that represents the coordinates of the Voronoi generators and their corresponding Voronoi edges. These data contain informations that suitably programmed can, for instance, move the pen of a plotter to print the diagram. However, for practical applications, this is a time consuming and expensive process in contrast to an automatic display and printing, or even better, an interactive generation of VD.

The intention of this communication is to report the development of a graphic interface to automate the visualization of VD generated by VORONOI2.

*Email: roque@mat.ufrgs.br

Voronoi diagrams: A planar Voronoi diagram is a partion of the plane into regions according to the principle of the *nearest-neighbor*. More precisely, let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of distinct points in the plane, and let $d(p_i, p_j)$ denote the Euclidean distance from p_i to p_j.

A *Voronoi region* $R(p_i)$ generated by the point p_i is defined by:

$$R(p_i) = \{p; d(p, p_i) < d(p, p_j), \forall j \neq i\}$$

The point p_i is called a *Voronoi generator*. The set of all Voronoi regions $\{R(p_1), R(p_2), \ldots, R(p_n)\}$ make a partition of the plane which is called the *Voronoi diagram* of P.

A common boundary of two Voronoi regions, say $R(p_i)$ and $R(p_j)$, is called a *Voronoi edge* and a point where the boundaries of three or more Voronoi edges meet is called a *Voronoi vertice*.

The program VORONOI2: VORONOI2 [?] is a program, written in *Fortran 77*, that generates VD data structure based on the incremental-type algorithm. It starts with a VD with three generators and adds to that diagram a new generator. This process is repeated in a one-by-one step until it reaches n generators. This program provides two output files, namely, *penmove* and *vortopology*. The *vortopology* file contains topological and metrical data that form the structure of the VD. The *penmove* file contains a sequence of pen movements for drawing the diagram.

The data in the *penmove* file consist of the following three types:

P x y , which means, draw a point at (x, y) coordinate values.

M x y , which means, for instance, move the pen to the point with coordinates (x, y).

D x y , which means, draw a straight line segment with starting point at the current pen position and end point at coordinate (x, y).

Visualization of VD: Some applications of VD demand for a best fit of the Voronoi diagram generated in a region of the plane according to the particular especifications of the problem (symmetries, boundary shape, etc). This imposes the necessity of an efficient visualization facility to avoid spending too much time in generation and plotting of a large number of Voronoi diagrams. In this regard, the visualization of the diagrams becomes an important process of analysis and improvement for the generation of new diagrams.

VORONOI2 does not provide an automatic display facility, demanding the drawing of the VD in a plotter each time you need to visualize them. This is a very time

consuming and expensive process. Therefore, a program, called VDVIEW, has been developed to avoid such detoir and to allow a straightforward visualization of the VD generated.

VDVIEW is written in C^{++} and makes use of the graphics library X-lib, running for any system that supports X-Windows. An alternative version has been implemented for the DOS operating system.

As VORONOI2 and VDVIEW can run under the UNIX system, we have created a sequence of commands that allows the automatic display of the VD generated in an individual window. Thus, the user can draw as many Voronoi diagram as necessary easing the comparison/improvement process to obtain better results.

For those who have made use of VORONOI2, it is only necessary to provide the `number of the generators =` and `seed for random numbers =` to generate the penmove data file. Now, with VDVIEW the Voronoi diagram pops up as the final result.

Conclusion: In this communication we have presented VDVIEW, a program written in C^{++}, that automates the visualization of Voronoi diagrams generated by VORONOI2 [?]. The printout of the VD is now straightforward as it can be done, for instance, using the *snapshot* facility of the SUN workstation.

Although VORONOI2 and VDVIEW can provide an easy way for the generation and visualization of VD, they are not flexible enough for many applications in engineering where the technique of numerical simulations with finite volume analysis is needed. In such case, the domains considered are in general irregular and the discretization of these domains with a structured grid is not the best one. There is a need to conform the discretization according to symmetries, shape of contour and physical considerations, which has led to the search for a better method to discretize the domain with non-structured grid. The VD technique has show to be an useful tool to get such discretization [?, ?].

Motivated by this problem, we have started the developement of a VD generator which can interactively construct a VD inside any polygonal contour. The interactive Voronoi diagram generator (IVDG) makes use of an incremental-type algorithm to construct the diagram. With the IVDG one can choose an arbitrary polygonal domain and construct the VD inside it by inserting the generators one-by-one with a simple mouse click. An experimental prototype of the IVDG is running on X-Window system.

Further work is in progress to extend the interactive generation assuming arbitrary contours and dynamical generation of VD.

References

[1] Aurenhammer, F., Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, **23**, 345-405, 1991.

[2] Bentley, J. L., Weide, B. W, and Yao, A. C., Optimal Expected-Time Algorithms for Closest Point Problems. *ACM Trans. on Math. Soft.*, **6**, 563-580, 1980.

[3] Edelsbrunner, H. and Seidel, R., Voronoi Diagrams and Arrangments. *Discrete and Computational Geometry*, **1**, 25-44, 1986.

[4] Fortune, S., A Sweepline Algorithm for Voronoi Diagrams. *Algorithmica*, **2**, 153-174, 1987.

[5] Shamos, M. I. and Hoey, D., Closest-point problems. In *Proceedings of the 16th Annual IEEE Symposium on FOCS*, pp. 151-162, 1975.

[6] Sugihara, K. and Iri, M., Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic. *Proceedings of the IEEE*, **80**, 1471, 1992.

[7] Sugihara, K. and Iri, M., VORONOI2 Reference Manual - Topology-Oriented Version of the Incremental Method for Constructing Voronoi Diagrams.

[8] Tanigushi, N., Arakawa, C. and kobayashi, T., Cosntruction of a Flow-Simulating Method with Finite Volume Based on a Voronoi Diagram. *JSME Int. Jour.*, **34**, 18-23, 1991.

[9] Tanigushi, N. and kobayashi, T., Finite Volume Method on the Unstructured Grid System. *Computers & Fluids*, **19**, 287-295, 1991.

THE COMPUTATION OF VORONOI DIAGRAMS ON THE EUCLIDEAN AND SPHERICAL 2-ORBIFOLDS *

(EXTENDED ABSTRACT)

M. MAZON †and T. RECIO ‡
Dpto. Matematicas, Estadística y Computación, Universidad de Cantabria
Santander 39071, Spain

1.Introduction.

The Voronoi Diagram is a fundamental data structure in Computational Geometry useful to solve many proximity problems. The problem of computing the Voronoi Diagram, initially considered for a collection of sites in the Euclidean plane E^2, has been generalized in many directions that include, among others, Voronoi Diagrams with respect to convex distances [1,2,3], weighted Voronoi Diagrams [4], higher order Voronoi Diagrams [5], dynamic Voronoi Diagrams [6] and Voronoi Diagrams on metric surfaces [7,8,9].

Several optimal algorithms exist to compute Euclidean Voronoi Diagrams for finite collections of sites on the plane [9,10,11] but only for some particular cases of curved surfaces embedded in the Euclidean space E^3, the problem has been solved, namely:
-On the Riemann sphere S^2, Brown [9] proposed two solutions of the problem;
-On the surface of a cone, Dehne and Klein [8] generalize the sweepcircle technic of the plane to work on a cone

From a theoretical point of view, the case of the complete and simply connected Riemannian manifolds [12] has been explored obtaining interesting general properties of the geometrical and topological structure of the Voronoi Diagrams for discrete collections of sites on these metric spaces.

In this work we study, both computational and theoretical aspects of Voronoi Diagrams for a relevant class of metric surfaces that are the Euclidean and spherical two orbifolds. This class of surfaces includes, among many others, all the locally-euclidean and locally-spherical surfaces (i. e.: cylinders, Moebius bands, Klein bottles, flat toruses, projective plane, etc.)

Our method of constructing Voronoi Diagrams on such surfaces, uses mainly the fact that all these surfaces are isometrically covered by the Euclidean plane E^2 or the Riemann sphere S^2, and using the well known algorithms to compute Voronoi Diagrams in E^2 or in S^2, we are able to obtain the Voronoi diagram on the given surface.

---

* Partially supported by CICyT-PB 89/0379/C02/01.

†e-mail: mazon@ccucvx.unican.es

‡e-mail: recio@ccucvx.unican.es

2. Discrete groups of isometries and two orbifolds.

Let M denote the Euclidean plane E^2 or the two sphere S^2, the last with the riemannian metric inherited from E^3. Let $Isom(M)$ be the full group of isometries of M. If G is a discrete subgroup of $Isom(M)$, then the quotient M/G, whose points are the orbits of points of M under the action of G on M, inherits a natural metric from the metric in M: distance $d(p,q)$ between two orbits p and q being given as the distance between the sets p and q, that is, as the infimum of the distances in M, between points $P \in p$ and $Q \in q$. With this distance defined on M/G, it becomes a metric space, that can be thought of as the surface obtained from a fundamental domain D_G for G by identifying or glueing together those points in its boundary that belong to the same orbit. The metric surfaces obtained in this way are all the connected euclidean or spherical two-orbifolds. As there are only a finite number of discrete groups of $Isom(M)$, unless conjugation in the affine group of transformations, there are also a finite number of connected euclidean or spherical two orbifolds

Among the surfaces obtained via a quotient from the plane by a discrete subgroup of $Isom(E^2)$, the most well-known are the cones (of angle $\frac{2\pi}{n}$, G being equal to the cyclic group C_n), the cylinders (G is generated by a translation and two paralel reflections), the Moebius bands (G is generated by a glide reflection and two paralel reflections), the flat toruses (G is generated by two independent translations), the pillow cases (G is generated by two half turns and two paralel reflections) or the pillows (G being generated by two half turns and one translation).

Althought in what follows an infinite collection of points can be involved, we will use here the extended definition of Voronoi Diagram [12] that applies to discrete collections of points, namely: given a discrete subset $S = \{P_i : i \in I\}$ of M, the Voronoi region $V_S(P_i)$ of point P_i with respect to the set S is defined as

$$V_S(P_i) = \{Q \in M : d(Q,P_i) < d(Q,P_k), \quad for \quad every \quad k \neq i\}$$

3. The computation of Voronoi Diagrams for saturated sets of points in the plane.

Given $S = \{P_1, ..., P_n\}$ any finite collection of points in E^2, and given any discrete group G of euclidean isometries, consider the saturation GS of S by the action of G, that is:

$$GS = \{gP_i : g \in G, P_i \in S\}$$

We can suppose the discrete group G being given by some generator system for G. Let D_G be a fundamental domain of the form $Cl\, V_{GP}(P)$, for P a point in E^2 with trivial stabilizer (here $Cl\, A$ means the topological closure of set A) .

Most of discrete groups of isometries of the plane are infinite and in consequence, most of the times the set GS is an infinite but discrete subset of the plane. As points in GS are regularly distributed, the Voronoi Diagram for GS has some kind of regularity that allows to compute it by computing only the Voronoi Diagram of a certain finite subcollection of points of GS as following theorem states:

Theorem 1

Let G be a discrete group of euclidean isometries given by some generator system for G. Let D_G be a fundamental domain of the form $Cl\,V_{GP}(P)$, for P a point in E^2 with trivial stabilizer.

Suppose $S = \{P_1, ..., P_n\}$ is a subset of D_G and consider its saturation GS by the action of G. Then, there exists a finite subset S^* of GS such that S^* contains S and:

$$Vor_M(GS) = G(Vor(S^*) \cap D_G)$$

4. The computation of Voronoi Diagrams in the Euclidean and spherical 2-orbifolds.

Now the general problem we want to solve can be stated as follows:

PROBLEM: Given an euclidean or spherical 2-orbifold M/G, where G is a discrete subgroup of $Isom(M)$ and M denotes E^2 or S^2, and given a finite collection s of points in M/G, $s = \{p_i : 1 \leq i \leq n\}$, find the Voronoi Diagram $Vor_{M/G}(s)$ of s in M/G.

THE ALGORITHM: The Voronoi Diagram of s in M/G, $Vor_{M/G}(s)$ is obtained as follows:

STEP 1. Compute $Vor_M(GS)$ (use Th. 1)

STEP 2. Take off the edges of $Vor_M(GS)$ between regions of equivalent points and call $Vor^*_M(GS)$ the resulting partition of M.

STEP 3. Intersect $Vor^*_M(GS)$ with D_G.

STEP 4. Identify equivalent points in $Vor^*_M(GS) \cap D_G$

Next theorem assures the correctness of the algorithm.

Theorem 2.

The partition of M/G you get after STEP 4, is the Voronoi Diagram $Vor_{M/G}(s)$.

5. References

1. L. P. Chew and R. L. Drysdale,"Voronoi diagrams based on convex distances functions", *Proc. 1st ACM Symposium on Computational Geometry* (1985).
2. D. T. Lee, "Two dimensional Voronoi diagrams in the L_p metric", *Journal Assoc. Comput. Mach.*, 27,604-618 (1980).
3. M. L. Mazón, "Diagramas de Voronoi en caleidoscopios", Ph. D. Thesis. Universidad de Cantabria, Spain (1992).
4. F. Aurenhammer, *Voronoi Diagrams-A Survey of a Fundamental Geometric Data Structure*, ACM Comp. Surveys, Vol. 23, No. 3, (1991).

5. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Monography. on Theor. Comp. Science, (1987).
6. T. Roos, "Dynamic Voronoi Diagrams", Ph. D. Thesis. Würzburg University, (1991).
7. D. M. Mount, "Voronoi Diagrams on the surface of a polyhedron", Rep. No. 1496, Univ. of Maryland (1985).
8. F. Dehne and R. Klein, "An optimal algorithm for computing the Voronoi Diagram on a cone", Report SCS-TR-122, Carleton Univ., Ottawa, Canada (1987).
9. K. Q. Brown, "Geometric transforms for fast geometric algorithms", Ph. D. Thesis. Rep. CMU-CS-80-101, Carnegie-Mellon Univ., Pittsburgh, Pennsylvania, (1980).
10. M. I. Shamos and D. Hoey, "Closest-point problems", *Proc. 16th Ann. IEEE Symp. FOCS*, 151-162, (1975).
11. S. Fortune, "A sweepline algorithm for Voronoi Diagrams", *Algorithmica 2*, 153-174, (1987).
12. P. E. Ehrlich and H. C. Im Hof, "Dirichlet regions in manifolds without conjugate points", *Comment. Math. Helvetici*, 54, 642-658, (1979)

Hierarchical Surface Representations using Constrained Delauney Triangulations

Andreas Voigtmann, Ludger Becker and Klaus Hinrichs
Westfälische Wilhelms-Universität, Institut für numerische Mathematik - Informatik
Einsteinstr. 62, D-48149 Münster, Germany
Phone & FAX: (++49) 251 / 83 - 3755; email: {avoigt, beckelu, khh}@math.uni-muenster.de

1. Introduction

The problem of surface representation occurs in a variety of disciplines, including computer graphics, computer-aided design (CAD), and geographic information systems (GIS). We restrict ourselves to surface representations in geographic applications.

Given a set of topographical data, we want to reconstruct the shape of the surface at a variety of predefined resolutions. In mathematical terms, the reconstruction process at a fixed level of detail can be considered as the interpolation of a function of two variables. Our topographical data is given as a finite set of points $V = \{ (x, y, z) \in \mathbf{R}^3 \mid z = H(x, y) \}$, where $z = H(x, y)$ is the height of the surface at (x,y). This set S of two-dimensional points (x, y) may be distributed in the plane non-uniformly. In addition, we assume the existence of a set L of non-intersecting line segments describing surface-specific features like coast-lines, river banks, ridges (i.e. lines connecting peaks with passes) or valleys. We assume that the end points of the line segments are elements of S, and no point of S lies in the relative interior of a line segment, i.e. the line segments may only intersect in their endpoints. The set L provides constraints for the surface-reconstruction process. Since we use triangulations to reconstruct the surface, the constraints help us to preserve the surface-specific features. Without these constraints triangles are created whose edges cross characteristic lines of the surface. These triangles produce undesirable errors in the reconstructed surface (e.g. rivers flowing over little hills, etc.).

We consider the surface at different levels of detail to provide a significant data reduction for low resolutions which is useful for browsing very large data sets. The use of a hierarchical structure allows a tree-like traversal of the generated levels and therefore supports zooming in a given part of the surface without browsing large lists of triangles.

Many hierarchical surface models have been proposed (see [Flo89] and [Flo87] for a survey) including ternary hierarchical triangulations [PF87],[Flo84], quaternary hierarchical triangulations [BV84],[GG79], quadtree-based models [CT86],[SG85] and other triangulation methods like [SP92]. Most of these models use triangulations of the given point-data to reconstruct the surface but none of them considers the existence of constraints for the reconstruction process. Furthermore, some models suffer from the requirement of regularly sampled data points (quadtree-based models, quaternary hierarchical triangulation and [SP92]) or from triangulations which contain elongated triangles and therefore cause numerical instability during interpolation (ternary hierarchical triangulation).

A "good" model should allow non-uniform data distributions and should be based on an "optimal" triangulation, i.e. a triangulation where the resulting triangles have a regular shape. A well known triangulation of this kind is the Delauney triangulation [PS85]. A hierarchical surface model based on the Delauney triangulation is the Delauney pyramid [Flo89]. Since the Delauney pyramid cannot deal with the constraints defined by a set of line segments [Flo89], we introduce the constrained Delauney pyramid.

2. The constrained Delauney pyramid

The Delauney pyramid cannot deal with the constraints defined by a set L of line segments which defined the so called *constrained graph* G_c. We present the *constrained Delauney pyramid*, a multiresolution surface model based on the Delauney pyramid, which considers the constraints during the reconstruction process. Two vertices s and t of G_c are called mutually visible if and only if either $\overline{st} \cap l = \emptyset$ for all line segments $l \in L$ or if $\overline{st} \in L$.

Similar to the (standard) Delauney triangulation we can define the constrained Delauney triangulation by the *constrained circumcircle property*: A constrained triangulation CT, i.e. a maximal planar straight line graph which contains G_c as a subgraph, is a *constrained Delauney triangulation* CDT if and only if no ver-

tex which is mutually visible to each vertex of a triangle $t \in CT$ lies inside the circumcircle of triangle t. Algorithms to compute constrained Delauney triangulations are presented in [Che89], [FP92] and [MV93]. A *constrained Delauney sequence* $CDS = [CDT_0, ..., CDT_m]$ is a sequence of constrained Delauney triangulations with the following properties:

1. CDT_i is a constrained Delauney triangulation of subsets $S_i \subseteq S$ and $L_i = \{\overline{st} \in L | s, t \in S_i\}$, $i = 0, ..., m$
2. $S_{i-1} \subset S_i$, $i = 1, ..., m$
3. $E(CDT_i) \leq \varepsilon_i$, $i = 1, ..., m$, where ε_i denotes the predefined maximum error at level i
4. $E(CDT_i) \leq E(CDT_{i-1})$, $i = 1, ..., m$

Starting with an initial triangulation CDT_0, we obtain each constrained Delauney triangulation CDT_{i+1} from its predecessor by inserting points into S_i until property 3 is satisfied. The constraints must be inserted if the corresponding endpoints have been included in the triangulation. To maintain the constrained Delauney triangulations during the insertion of points and line segments we use the algorithm of [FP92]. This algorithm computes the constrained Delauney triangulation for sets $S' = S \cup \{p\}$ and $L' = L \cup \{l\}$ based on a constrained Delauney triangulation for sets S and L. Similar to the algorithm for the Delauney triangulation ([Flo87]) only local modifications are required to obtain the new triangulation.
The relationship between CDT_{i-1} and CDT_i is given by the following two difference sets. The *difference set* D_{i-1} between CDT_{i-1} and CDT_i is the set of all triangles of CDT_{i-1} which do not belong to CDT_i, i.e. the union of all triangles violating the circumcircle property due to the point set $S_i \backslash S_{i-1}$. The difference set D_i' between CDT_i and CDT_{i-1} is the set of all triangles of CDT_i which do not belong to the predecessor CDT_{i-1}.
A *constrained Delauney pyramid CDP* is the representation of a constrained Delauney sequence $CDS = [CDT_0, ..., CDT_m]$ having the following properties:

1. CDT_0 is the top level of *CDP*
2. Level i in *CDP* corresponds to CDT_i
3. Each triangle t of CDT_{i-1} which does not belong to D_{i-1} is contained in CDT_i. We connect these identical triangles of CDT_{i-1} and CDT_i by a conceptual link.
   Each triangle t of D_{i-1} is replaced in CDT_i by the subset of triangles of D_i' intersecting t. Triangle t is connected with each element of this subset by a conceptual link.

Since the line segments are defined for the finest resolution of the data points, only few constraints will be inserted into the triangulations at coarse resolutions. This results in a loss of information of the topography for many levels. To prevent these errors, we have to generalize the constraints in order to obtain a *generalized constraint graph* for each resolution. These generalized constraints are used to construct the final pyramid.
During the generalization process we search paths in the constraint graph G_c connecting points a and b which have already been inserted into the triangulation. If these paths have the following properties, edge $\overline{ab}$ is said to be a *generalized constraint*:

(1) The paths may have small gaps. These gaps, e.g. small valleys, do not change the shape of the reconstructed surface at coarser resolutions.
(2) The path needs not directly connect a and b. It is sufficient if the path intersects a circle $U(\bullet)$ around the vertices.
(3) The distance between the path of the constraint graph which is generalized and the generalized edge must be less than γ.

According to the above criteria we have to perform tests on the neighborhoods of vertices to identify the generalized constraints permanently. Since this is too expensive, we split the computation of the generalized constraints into a preprocessing step and a search process.
The search for the generalized constraints is performed by a *depth first search* on the *search graph SG* created in the preprocessing step. *SG* is similar to the constraint graph but contains additional edges closing gaps in the constraint graph according to the above property 1. In addition edges are either rearranged or split according to the above property 2. In *SG* a generalized constraint between vertex p and vertex q is represented by a path connecting p and q which is close to edge $\overline{pq}$ (property 3). A detailed algorithm can be found in [Vo93].

Problems arise during this process since the generalization of constraints is a heuristic approach to surface reconstruction.

3. The constrained Delauney pyramid in database systems

Topographic data sets usually consist of a large number of data items (e.g. topographic data derived from remote sensing data). The GIS which is used to access the topographic data stores these data sets in an underlying database system. If we use the constrained Delauney pyramid to browse and zoom in such sets of geometric data, the size of these data sets may exceed the main memory size. Hence we have to provide secondary storage structures and algorithms working on such structures to integrate our multiresolution surface model into a database system.

In the following we assume that the topographic data are stored in a database using spatial data structures. For our purposes we further assume that this spatial data structure supports a quadtree-like subdivision of the data space (a comprehensive overview on spatial data structures is given in [Sam90a] and [Sam90b]). Since the divide-&-conquer paradigm requires to divide the given data set efficiently, we assume that there is a function which enables splitting the given data set on the internal boundaries of the data structure.

The divide-&-conquer algorithm can be described by the following steps:

- *Divide phase (divide the topographic data)*

  Let $\Im$ be an index set. We consider a rectangular subdivision of the data space which induces a partitioning of S into pairwise disjoint sets S^k, $k \in \Im$. The set L of line segments is divided into sets L^k, $k \in \Im$ where L^k contains all constraints $\overline{ab}$ with $a \in S^k$ or $b \in S^k$. Obviously, these subsets fulfill the property $S = \cup_{k \in \Im} S^k$, $L = \cup_{k \in \Im} L^k$. Since the upper bound of memory consumption of a constrained Delauney pyramid can be estimated based on the size of the sets S^k, L^k (see [Vo93] for further details), we can make the subsets S^k, L^k small enough such that their (constrained) Delauney pyramid can be computed in main memory.
- *Construction phase*

  For each $k \in \Im$: Compute the (constrained) Delauney pyramid of S^k, L^k in main memory and store the resulting data structure on secondary storage.
- *Merge phase*

  Perform a merge of the (constrained) Delauney pyramids on secondary storage.

In the merge phase the algorithm merges the triangulations of the pyramids level by level starting at level 0. To merge two triangulations, we use the algorithm presented in [MV93] which is an extension of the merge algorithm for constrained Delauney triangulations presented in [Che89]. This extension fixes several problems of the original algorithm. Details can be found in [MV93], [Che89] and [LS80]. To perform the merge, it is sufficient to load the set $T = T_l \cup T_r$ of required triangles into main memory. T_r consists of the following triangles:

- All triangles in the current level of the pyramid having at least one vertex which lies on the boundary of the convex hull of S^r and is visible[1] from S^l (S^l and S^r denote the point data of the triangulations to be merged).
- All triangles t in the triangulation of S^r such that there is a point of S^l lying inside the circumcircle of t.
- All triangles in the triangulation of S^r intersected by a constrained edge $l = \overline{ab}$ with $a \in S^r$ and $b \in S^l$.
- All triangles in the triangulation of S^r sharing at least one edge with a triangle chosen by the criteria above.

Since we use a merge algorithm for constrained Delauney triangulations we have to insert all constraints ("real" and generalized) connecting the two partial triangulations into both triangulations and then we have to apply the algorithm of [MV93] to merge the triangulations. Let G_c' be the subgraph of the constraint graph G_c consisting of all line segments $l \in L^l \cup L^r$ which intersect the set of triangles defined above and

1. We call $b \in S^r$ *visible from* S^l if there is a $a \in S^l$ with the property $\overline{ab} \cap t = \emptyset$ for all triangles t in the triangulation of S^r.

have not yet been inserted into the triangulation. The subgraph G_c' is used to search generalized constraints between the triangulations of S^l and S^r as follows: For each level k of the pyramid processed by the merge step we use G_c' to compute a search graph SG. In SG we search for generalized constraints $\overline{ab}$ with $a \in S^l$ and $b \in S^r$ or vice versa.

The whole merge process of two constrained Delauney pyramids may be described by the following algorithm:

For each level i of the pyramid do:

- Let T_l, T_r be the sets of triangles as defined above, and let $S_l(T_l) \subseteq S^l$, $S_r(T_r) \subseteq S^r$ be the corresponding sets of vertices. Load these sets into main memory.
- Let L^1 be the set of "real" constraints $l = \overline{ab}$ with $a \in S_l(T_l)$ and $b \in S_r(T_r)$.

  Remove all triangles from T_l, T_u intersected by any $l \in L^1$. Insert each $l \in L^1$ into both triangulations.
- Let L^2 be the set of generalized constraints computed by the method presented in the previous paragraph. For each $l \in L^2$: If l does not intersect any "real" constraint then remove from both triangulations all previously inserted generalized constraints which intersect l. Further, remove all triangles from T_l, T_r intersected by l and insert l into both triangulations. If l intersects any "real" constraint, discard l.
- Use the algorithm presented in [MV93] to merge both triangulations.

Literature

[BV84] R. Barrara, A.M. Vazques; "A Hierarchical Method for Representing Relief"; *Proc. Pecora IX Symposium in Spatial Information Technologies for Remote Sensing Today and Tomorrow*, Sioux Falls, South Dakota, Oct. 1984, pp. 87-92

[Che89] L.P. Chew; "Constrained Delauney Triangulations"; *Algorithmica*; Vol. 4, 1989; pp. 97-108

[CT86] Z.T. Chen, W.R. Tobler; "Quadtree Representation of Digital Terrain"; *Proc. Autocarto*, London, 1986; pp. 475-484

[Flo84] L. De Floriani et. al.; "A Hierarchical Structure for Surface Approximation"; *Computer and Graphics*; Vol. 8, No. 2, 1984; pp. 183-193

[Flo87] L. De Floriani; "Surface representations based on triangular grids"; *The Visual Computer*; Vol. 3, No. 1, Feb. 1987; pp. 27-50

[Flo89] L. De Floriani; "A Pyramidal Data Structure for Triangle-Based Surface Description"; *IEEE Computer Graphics & Applications*; Vol. 9, No. 2, Mar. 1989; pp. 67-78

[FP92] L. De Floriani, E. Puppo; "An On-Line Algorithm for Constrained Delauney Triangulation"; *CVGIP: Graphical Models and Image Processing*; Vol. 54, No. 3, Jul. 1992; pp. 290-300

[GG79] D. Gomez, A. Guzman, "Digital Model for Three-Dimensional Surface Representation"; *Geo-Processing*; Vol. 1, 1979; pp. 53-70

[LS80] D.T. Lee, B.J. Schachter; "Two Algorithms for Constructing a Delauney Triangulation"; *Int. Journal of Computer and Information Sciences*; Vol. 9, No. 3, 1980; pp. 219-242

[MV93] J.-M. Moreau, P. Volino; "Constrained Delauney Triangulation Revisited"; *Proceedings 5th Canadian Conference on Computational Geometry*; Waterloo, 1993

[PF87] J. Ponce, O. Faugeras; "An Object Centered Hierarchical Representation for 3D Objects: The Prism Tree"; *Computer Vision, Graphics and Image Processing*; Vol. 38, No. 1, Apr. 1987

[PS85] F.P. Preparata, M.I. Shamos; *Computational Geometry - An Introduction*; Springer, Berlin, 1985

[Sam90a] H. Samet; *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*; Addison-Wesley, Reading, MA, 1990

[Sam90b] H. Samet; *The Design and Analysis of Spatial Data Structures*; Addison-Wesley, Reading, MA, 1990

[SG85] F. Schmitt, B. Gholizadeh; "Adaptive Polyhedral Approximation of Digitized Surfaces"; *Proc. SPIE Conf. on Computer Vision in Robots, SPIE*, Bellingham, Wash.; Vol. 595, 1985; pp. 167-171

[SP92] L. Scarlatos, T. Pavlidis; "Hierarchical Triangulation Using Cartographic Coherence"; *CVGIP: Graphical Models and Image Processing*; Vol. 54, No. 2, Mar. 1992; pp. 147-161

[Vo93] A. Voigtmann; "Hierarchische Repräsentation diskreter geometrischer Daten"; Diplomarbeit; Fachbereich Mathematik, Universität-GH Siegen, 1993

On Delaunay Oriented Matroids for Convex Distance Functions.

F. Santos [†,‡]

Abstract: For any finite set S of points in the Euclidean d-space E^d one can define an oriented matroid in terms of how Euclidean hyperspheres partition it. We show that these "Delaunay oriented matroids" can be generalized to some other definitions of "sphere". In particular, for the pseudo-circles which are obtained by translations and scalings of a smooth, strictly convex, closed curve in the plane (the so-called δ-spheres of a convex distance function δ). Surprisingly, in this case we obtain that non-representable Delaunay oriented matroids can be obtained.

Keywords: Voronoi diagram, Delaunay triangulation, oriented matroid.

1. The Euclidean case.

One of the fields where the theory of oriented matroids has an application is in Voronoi diagrams and their dual Delaunay triangulations. If S is a finite set of points (*sites*) in the Euclidean d-space E^d, an oriented matroid can be constructed that contains the vicinity information between points of S in much the same way that Voronoi diagrams do. We call it the *Delaunay oriented matroid* of S (notated DOM(S)) and it is defined as follows, in terms of covectors. A signed partition (C^+, C^0, C^-) of S is a covector of DOM(S) if and only if a hypersphere C exists such that $C^0 = C \cap S$ and C^+ and C^- are the two parts in which C divides $S \setminus C^0$. A hypersphere is meant to be any scaled translation of the unit hypersphere $\{P \in E^d : |OP| = 1\}$.

See [1] for a different definition of these oriented matroids (in terms of circuits instead of covectors) in the planar case. The definition shown there was introduced by Bland and Las Vergnas [2] as an orientation of the matroids that Cheung and Crapo called *Möbius geometries* [4].

One of the main properties of these Delaunay oriented matroids is that they are *representable*. This can be shown via an indirect construction similar to the one used by Brown to reduce the computation of Delaunay triangulations to the computation of convex hulls in the Euclidean space with one extra dimension E^{d+1} (cf. [3]).

Consider the map $f : E^d \to E^{d+1}$ defined by $f(x_1, \ldots, x_d) = (x_1, \ldots, x_d, \sum x_i^2)$. This map induces a lifting of the sites S into a set $\tilde{S}$ lying in a paraboloidal hypersurface in E^{d+1}. The Delaunay oriented matroid DOM(S) of S coincides with the oriented matroid of the image points $\tilde{S}$ as an affine point configuration. In other words, the covectors of DOM(S) are the signed partitions of $\tilde{S}$ induced by all the hyperplanes in E^{d+1}.

The fact that the two constructions are equivalent comes from the fact that the map f lifts E^d into a paraboloidal hypersurface in E^{d+1} whose intersection with any hyperplane projects down onto a hypersphere or a hyperplane in E^d.

---

† Departamento de Matemáticas, Estadística y Computación. Facultad de Ciencias. Universidad de Cantabria. 39071-SANTANDER. Spain. E-mail: santos@ccucvx.unican.es

‡ Partially supported by DGICyT PB 92/0498-C02 and the David and Lucile Packard Foundation.

Some other easy-to-proof properties of Delaunay oriented matroids are summed in the following statement.

Let S be a finite set of points in E^d and let k be the dimension of the affine subspace spanned by S. Then the Delaunay oriented matroid of S is acyclic, polytopal and representable. Its rank equals $k+1$ if S is contained in a hypersphere and $k+2$ otherwise.

The definition of the Delaunay oriented matroid of S in terms of hyperspheres shows its connection with the Voronoi diagram of S and its dual the Delaunay diagram. We use the term *Delaunay diagram* to refer to the topological dual of the Voronoi diagram (as opposed to the term *Delaunay triangulation* which is usually defined to be any triangulation of this Voronoi dual). In this setting the Delaunay diagram of S is a polyhedral complex in E^d whose cells are the convex hulls $< T >$ of those subsets T of S for which a hypersphere exists passing through all the points in T and having $S \setminus T$ outside. Then, for any set $T \subset S$ whose convex hull is a cell in the Delaunay diagram of S, DOM(S) contains a covector $(\emptyset, T, S \setminus T)$.

Unfortunately the converse is not true and thus the Delaunay oriented matroid does not contain all the information to recover the Delaunay diagram. In fact, a covector as described above associated to a subset T can be produced by a hypersphere passing through T but having $S \setminus T$ inside. Technically speaking we can say that the Delaunay oriented matroid completely describes the *Delaunay complex* as introduced in [8], but not the Delaunay diagram. We recall that the Delaunay complex is the convex hull of the lifted set $\tilde{S}$ in E^{d+1}. Its lower part projects onto the Delaunay diagram and its upper part projects onto the *furthest site Delaunay triangulation*).

An oriented matroid containing all the combinatorial information of the Delaunay diagram can be obtained from the Delaunay oriented matroid by just introducing a new point ∞ which is considered to lie in any hyperplane and in the exterior region of any hypersphere. The signed partitions of $S \cup \{\infty\}$ induced by all hyperspheres and hyperplanes are again the covectors of an oriented matroid that we will call the *extended Delaunay oriented matroid* EDOM(S) of S. It has almost the same properties as the Delaunay oriented matroid.

Let S be a finite set of points in E^d and let k be the dimension of the affine subspace spanned by S. Then the extended Delaunay oriented matroid EDOM(S) *of S is acyclic, polytopal and representable and has rank $k+2$. Its deletion at ∞ is the Delaunay oriented matroid* DOM(S) *of S and its contraction at ∞ is the usual affine oriented matroid of S.*

This new oriented matroid EDOM(S) contains the combinatorial information of the Voronoi and Delaunay diagrams, in the following sense.

Let S be a finite set of points in E^d and consider the Voronoi diagram $Vor(S)$ of S and the extended Delaunay oriented matroid EDOM(S) *of S. Then,*

i) The Voronoi regions of two points $A, B \in S$ are adjacent (i.e, their boundaries intersect) if and only if EDOM(S) *contains a covector (C_+, C_0, C_-) with $C_+ = \emptyset$ and $A, B \in C_0$.*

ii) The convex hull $< T >$ of a subset $T \subset S$ is a cell in the Delaunay diagram if and only if EDOM(S) *contains the covector $\emptyset, T, S \cup \{\infty\} \setminus T)$ is a covector of* EDOM(S).

2. The case of convex distance functions

The study of Voronoi diagrams for metrics other than the Euclidean one is of interest in many applications. See, for example, [10] for some developments in this area. The class of metrics that we are going to study here is the so-called *convex distance functions*. Convex distance functions were

introduced in the context of Voronoi diagrams by Chew and Drysdale [5] who gave an algorithm for computing the Voronoi diagrams produced by them and some of their applications. See also [6], [10] and [11] for some properties of convex distance functions.

Convex distance functions are usually defined as coming from a convex body. Instead, we are going to give an axiomatic definition to stress the properties they enjoy and then show the way they are related to convex bodies.

Definition 2.1 A *convex distance function* in the Euclidean d-space E^d ($d \geq 2$) is a map $\delta : E^d \times E^d \longrightarrow \mathbb{R}_{\geq 0}$ satisfying the following properties.

(D0) $\delta(A,B) \geq 0 \quad \forall A,B \in E^d$ and $\delta(A,B) = 0$ iff $A = B$.

(D1) $\delta(A,B) + \delta(B,C) \geq \delta(A,C) \quad \forall A,B,C \in E^d$ (*triangle inequality*).

(D2) $\delta(A,B) = \delta(A+v, B+v) \quad \forall A,B \in E^d, \forall v \in \mathbb{R}^d$ (*invariance under translations*).

(D3) $\delta(A,B) + \delta(B,C) = \delta(A,C) \quad \forall B \in [A,C]$, , where $[A,C]$ denotes the line segment between A and C (*additivity on segments*).

Let δ be a convex distance function. For any point A and any positive real number r we call δ-sphere with center A and radius r the set of points with distance r from A. (Note that convex distance functions need not be symmetric and we must distinguish 'distance from' from 'distance to'). It follows from axioms (D0)–(D3) that the unit δ-sphere K (with center O and radius 1) is the boundary of a bounded convex body with O in its interior and that all other δ-spheres are scaled translations of it. Moreover, the distance $\delta(A,B)$ can be computed from the unit δ-sphere K as being the unique scaling factor r that makes $A + rK$ pass through B. The converse is also true. For any boundary K of a bounded convex body in E^d with the origin in its interior this procedure defines a map $E^d \times E^d \to \mathbb{R}_{\geq 0}$ which is the *convex distance function induced by* K. This is an alternative definition of convex distance functions. (In fact, the usual one).

Some additional properties of convex distance functions are:

i) If the unit δ-sphere is strictly convex, (i.e. it contains no line segments) then the triangle inequality is strict: $\delta(A,B)+\delta(B,C) = \delta(A,C)$ iff $B \in [A,C]$. We say in this case that δ is *strictly convex.*

ii) If the unit δ-sphere is smooth (i.e. it has a unique tangent hyperplane at any point) then every $d+1$ points not lying in any hyperplane are δ-cospherical (lie in a common δ-sphere). We say then that δ is *smooth.* (Remark: we only have a proof for this up to $d = 3$. For any dimension the converse is true: if δ is not smooth there exist $d+1$ points which are not δ-cospherical nor co-hyperplanar).

The results we present here concern convex distance functions in the plane E^2. We will see that smooth, strictly convex distances produce Delaunay oriented matroids in the same way than the Euclidean distance. Now, the oriented matroids we obtain can be *non-representable.*

Proposition 2.2 *Let δ be a smooth, strictly convex distance function in E^2. For any set S of sites call* $\mathrm{DOM}_\delta(S)$ *the collection of signed partitions of S defined by all hyperplanes and δ-spheres. Then* $\mathrm{DOM}_\delta(S)$ *is an oriented matroid for all possible sites S if and only if δ is strictly convex and smooth. In this case the extension of this oriented matroid to the infinity point is again an oriented matroid* $\mathrm{EDOM}_\delta(S)$.

These Delaunay oriented matroids have the same properties we showed for the Euclidean distance, *except for the representability.*

Proposition 2.3 *Let δ be a smooth, strictly convex distance function in E^2. Let S be a finite set of points in the plane and call k the dimension of the affine subspace spanned by S. Then the oriented matroids* $\mathrm{DOM}_\delta(S)$ *and* $\mathrm{EDOM}_\delta(S)$ *introduced in the previous result are both acyclic and*

polytopal. The rank of $\mathrm{EDOM}_\delta(S)$ *equals* $k+2$ *and the rank of* $\mathrm{DOM}_\delta(S)$ *equals* $k+1$ *if* S *is contained in a certain* δ*-sphere and* $k+2$ *otherwise.*

Let us now assume that our distance function is not only smooth and strictly convex but also symmetrical. If it is an affine transform of the Euclidean distance (i.e. if its unit δ-sphere is an ellipse) then the only Delaunay oriented matroids that we can obtain are those obtained for the Euclidean distance. The following result shows that in any other case, different oriented matroids can always appear and, moreover, for any δ some configurations of points produce *non-representable* ones. Our example with eight points is minimal because any rank 4 oriented matroid with less than eight points is representable (cf. [1]). We want to remark that the symmetry assumption on δ makes things easier and is used in our construction, but it is probably not needed for the result to be true.

Theorem 2.4 *Let* δ *be a symmetric smooth strictly convex distance function in the plane whose unit* δ*-sphere* K *is not an ellipse. Then the Delaunay oriented matroid of some set* S *of eight points with respect to* δ *is not representable.*

Proof: The proof is based on lemmas 2.5 and 2.6, that we give without proof. Lemma 2.5 has been used in [6] (see also the extended abstract in [7]). Lemma 2.6 can be proved in a purely computational way, although geometric proofs exist. For simplicity, in lemma 2.5 we will assume that the ellipse L is actually an Euclidean circle. This is no lose of generality, because it can always be achieved by an affine transform to our distance function δ, which does not affect the Delaunay oriented matroids obtained. We can then obtain eight points as in figure 1.(a), where points A, B, C and D are the four points in lemma 2.5 and E, F, G and H are chosen along the circle in such a way that $[A,E]=[B,F]=[C,G]=[D,H]$ and not lying on K. (Remark: four such points E,F,G,H might not exist if, for example, K coincides with the circle L along the arcs (A,B) and (C,D). In this case a slight affine transformation of K would permit to find eight points in the same conditions, although the unit δ-sphere K could be not-completely contained in the circle L).

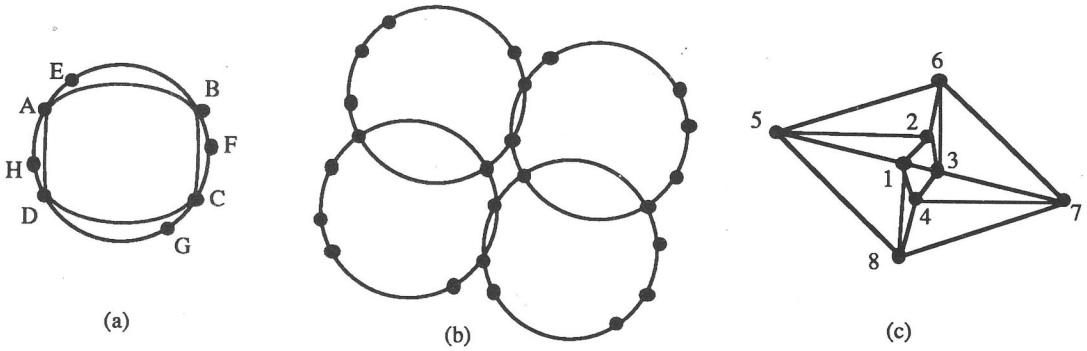


Figure 1

If we patch four copies of the eight points as in figure 1.(b), then the Delaunay diagram of the eight central ones with respect to distance δ is as shown in figure 1.(c). This implies that the eight first covectors of lemma 2.6 are in the corresponding Delaunay oriented matroid. The two last ones come from collinearity of points $\{1,3,5,7\}$ and $\{2,4,6,8\}$. Thus, the Delaunay oriented matroid obtained is not representable. ∎

Lemma 2.5 *Let* K *be a closed convex simple curve containing the origin in its interior and centrally symmetric. Then there exists an ellipse* L *centered at the origin which contains* K *and intersects it in at least two pairs of opposite points.*

Lemma 2.6 *Let $\mathcal{V}$ be the set of covectors of an oriented matroid of rank at most 4 in 8 points. If $\mathcal{V}$ contains the covectors*

$$(00++0+++) \quad (+00++0++) \quad (++00+0+) \quad (0++0+++0)$$
$$(+0++00++) \quad (++0++00+) \quad (+++0++00) \quad (0+++0++0)$$
$$(+0+0+0+0) \quad (0+0+0+0+)$$

then $\mathcal{V}$ is not representable.

An interesting corollary of theorem 2.4 (actually, of the proof) comes from the fact that a diagram like the one in figure 1.(c) is not a regular triangulation (provided that points $\{1,3,5,7\}$ and $\{2,4,6,8\}$ are collinear. This means that it is not the projection of the lower envelope of any lifting of the eight points into 3-space. This property is usually known as the lifting property of Delaunay triangulations for the Euclidean distance (cf. [3]) and implies that the computation of Delaunay diagrams in E^d is equivalent to the computation of a convex hull in E^{d+1}. This is not possible for convex distance functions

Corollary 2.7 *The only (symmetric) smooth strictly convex distance functions in the plane which have the lifting property are affine transforms of the Euclidean distance.*

References

[1] **A. Bjorner, M. Las Vergnas, B. Sturmfels, N. White, G.M. Ziegler** *Oriented Matroids.* Cambridge University Press, 1992.

[2] **R.G. Bland, M. Las Vergnas** *Orientability of Matroids.* J. Combinatorial Theory, ser. B 23, 1978. p. 94–123.

[3] **K.Q. Brown** *Voronoi Diagrams from Convex Hulls.* Information Processing Letters 9, 1979. p. 223–228.

[4] **A.L.C. Cheung, H.H. Crapo** *A Combinatorial Perspective on Algebraic Geometry.* Advances in Math 20, 1976. p. 388–414.

[5] **L.P. Chew, R.L. Drysdale** *Voronoi Diagrams Based on Convex Distance Functions.* In *Proceedings 1st ACM Symposium on Computational Geometry*, 1985. p. 235–244.

[6] **A.G. Corbalán, M. Mazón, T. Recio, F. Santos** *On the Topological Shape of Planar Voronoi Diagrams.* In *Proceedings 9th ACM Symposium on Computational Geometry*, 1993. p. 109–115.

[7] **A.G. Corbalán, M. Mazón, T. Recio, F. Santos** *On the Topological Shape of Planar Voronoi Diagrams.* In *Proceedings 9th European Workshop on Computational Geometry.* Universität Hagen, 1993.

[8] **H. Crapo, J.P. Laumond** *Hamiltonian cycles in Delaunay complexes.* In *Journées Géometrie et Robotique LAAS/CNRS.* Lect. Notes in Comp. Sci. 1416, 1988.

[9] **C. Icking, R. Klein. N.M. Le, L. Ma** *Convex Distance Functions in 3-space are Different.* In *Proceedings 9th ACM Symposium on Computational Geometry*, 1993. p. 116–123.

[10] **R. Klein** *Concrete and Abstract Voronoi Diagrams.* LNCS 400, Springer-Verlag, Berlin, 1989. p. 116–123.

[11] **M. Mazón** *Diagramas de Voronoi en Caleiodoscopios.* Ph.D. Thesis. Univ. de Cantabria, 1992.

Simple search for nearest foreign neighbors in arbitrary L^t-metrics

Thorsten Graf and Klaus Hinrichs

Institut für numerische und instrumentelle Mathematik -INFORMATIK-
Westfälische Wilhelms-Universität Münster
e-mail: graf@math.uni-muenster.de khh@math.uni-muenster.de

Abstract

Solving proximity problems with *simple* algorithms is of great importance in fields like traffic-controlling, motion-planning, and VLSI-design where the simplicity of an algorithm is measured by the complexity of its description and its implementation.

In the *all-nearest-foreign-neighbors* problem one has to find for each point in a given planar configuration of different colored points a nearest foreign neighbor, i.e. a nearest neighbor with a different color. We present an optimal plane-sweep algorithm for solving the all-nearest-foreign-neighbors problem with respect to an arbitrary L^t-metric ($1 \leq t \leq \infty$). This improves the results for the L^1- and L^∞-metrics presented in [GH93a].

A competetive algorithm is based upon point location in several Voronoi diagrams. Our algorithm is superior to this algorithm concerning its simplicity since nearest foreign neighbors are computed *directly* without computing any Voronoi diagram. An implementation of our algorithm only uses elementary data structures such as balanced binary search trees and some kind of priority search tree.

1 Introduction

The *all-nearest-neighbors* problem (ANN problem) is a fundamental problem in computational geometry. It is well-known that this problem can be solved in time $O(n \log n)$ which is optimal in the algebraic decision tree model of computation. Optimal algorithms for the ANN problem can be found in [Sha75, Vai89, HNS92]; a survey is given in [FKP92]. Consider the following modification of the ANN problem:

Let $S = \cup_{i=1,\ldots,m} S_i$ be a set of points. Find for each $p \in S_i$ $(\forall 1 \leq i \leq m)$ a nearest neighbor in $S \setminus S_i$.

We reformulate the problem in an intuitive way: Let us *color* each point set with a unique color. Denote by $c_1, \ldots, c_m$ the colors of the sets $S_1, \ldots, S_m$ and let $c(p)$ denote the color of a point $p \in S$. Now we have to find a nearest neighbor of each point with the additional difficulty that the points are "color-blind" for their own color, i.e. each point $p \in S$ cannot "see" all other points in $S_{c(p)}$.

This problem is referred to as the *all-nearest-foreign-neighbors problem* (ANFN problem). Since algorithms which solve the ANFN problem also solve the ANN problem by choosing configurations that do not contain two points with the same color, the problem ANFN is also in $\Omega(n \log n)$. Distances between points are measured by an arbitrary but fixed L^t-metric. For two points $p, q \in \mathbb{R}^2$ their L^t-distance (Minkowski-distance) is given by

$$d_t(p,q) := (|p.x - q.x|^t + |p.y - q.y|^t)^{\frac{1}{t}} \quad 1 \leq t < \infty \qquad d_\infty(p,q) := \max\{|p.x - q.x|, |p.y - q.y|\}$$

Applications for algorithms solving the ANFN-problem for some metrics are found, e.g. in special traffic-control systems: Controlling vehicles of different types each of which is represented by a point one has to avoid collisions of two vehicles with different types. Coloring the points of the same type with a common and unique color the knowledge of a nearest foreign neighbor for each vehicle at discrete instants can be used to avoid collisions. The metrics d_1 and d_∞ are relevant to various applications, such as modelling of arm movements, disc transport mechanisms [LW80], and integrated circuit layout.

[AERT89] presents an algorithm for the ANFN problem with respect to the L^2-metric making use of several Voronoi diagrams; the main idea of the algorithm is that if q is a nearest foreign neighbor of p then q is Voronoi neighbor of some point in $S_{c(p)}$. Therefore for each color c_i the algorithm [AERT89] computes the sets $T_i \subset S \setminus S_i$ of those points which have a Voronoi neighbor in S_i and constructs the Voronoi diagrams $\mathcal{V}(T_i)$. The argument above then shows that point location with each point $p \in S_i$ in the Voronoi diagram $\mathcal{V}(T_{c(p)})$ finds nearest foreign neighbors correctly.

The ANFN problem for two different colors and with respect to the Euclidean metric is mentioned in [HNS92]. As we will see in section 4 this algorithm can be extended to arbitrary L^t-metrics. We will use this generalized version as part of our algorithm in order to avoid point location.

For the L^1- and $L^\infty-$ metrics [GH93a] presents an optimal algorithm for solving the ANFN-problem. This algorithm is used in the algorithm presented in this paper in a preprocessing step. This work improves the results

in [GH93a, AERT89].
The algorithm presented in this paper uses the plane-sweep principle. It inserts the points of S into sets T_i ($1 \leq i \leq m$) where each point $p \in S$ is contained in at most two of these sets. A nearest foreign neighbor of p can be found in $T_{c(p)}$. This can be done without computing any Voronoi diagram by extending the algorithm [HNS92] to arbitrary L^t-metrics; the idea is that point location with each point of a set in a Voronoi diagram of an other set of points can be understood as an ANFN problem for two different colors where one is only interested in nearest foreign neighbors for the points in the query set.

2 The algorithm PSANFN

Our algorithm "PSANFN" (plane-sweep all nearest foreign neighbors) uses the well-known plane-sweep principle (see, e.g.[PS85, GH93a]). In a step of preprocessing we use the algorithm in [GH93a] to solve the ANFN problem for the configuration S with respect to the L^∞-metric which requires $O(n \log n)$ time. This preprocessing step returns a L^∞-nearest foreign neighbor $\tilde{p}$ of each point $p \in S$. Due to the equivalence of the remaining Minkowski-metrics and the L^∞-metric, which means that for any $1 \leq t < \infty$ and for any points $p, q \in \mathbb{R}^2$ we have $d_\infty(p,q) \leq d_t(p,q) \leq 2^{\frac{1}{t}} d_\infty(p,q)$, the point $\tilde{p}$ might be sufficient as an approximation of the exact result in some applications.
For a point $p \in S$ the two diagonal lines (with slopes ± 1) through p subdivide the plane into four quadrants. Let us denote these quadrants by $QR(p) := \{q.x > p.x : |p.x - q.x| \geq |p.y - q.y|\}$, $QL(p) := \{q.x < p.x : |p.x - q.x| \geq |p.y - q.y|\}$, $QB(p) := \{q.y < p.y : |p.x - q.x| < |p.y - q.y|\}$, and $QT(p) := \{q.y > p.y : |p.x - q.x| < |p.y - q.y|\}$.
The algorithm PSANFN performs four sweeps, one sweep from left to right, from right to left, from top to bottom, and from bottom to top. We will show in section 3 that in the left-to-right sweep a nearest foreign neighbor of $p \in S$ is found if such a neighbor of p is contained in $QL(p)$. Together with similar arguments for the three other sweeps and the corresponding quadrants this proves the correctness of the algorithm. It therefore suffices to describe the first sweep from left to right.
Fix an arbitrary point $p \in S$. The situation does not change for p if we insert the points into S which we obtain by cyclic mirroring the point p along the diagonals and the axis. Denote these points by $p_1, \ldots, p_8$. If e.g. $\tilde{p} \in QL(p)$ and $\tilde{p}.y > p.y$ then $p_1 = \tilde{p}$, $p_2 = (\tilde{p}.x, -\tilde{p}.y)$, $p_3 = (-\tilde{p}.y, \tilde{p}.x)$, $p_4 = (\tilde{p}.y, \tilde{p}.x)$, $p_5 = (-\tilde{p}.x, -\tilde{p}.y)$, $p_6 = (-\tilde{p}.x, \tilde{p}.y)$, $p_7 = (\tilde{p}.y, -\tilde{p}.x)$, and $p_8 = (-\tilde{p}.y, -\tilde{p}.x)$.
Note that $p_i = \tilde{p}$ for any $1 \leq i \leq n$, but in the following it is not important to know for which value of i this is true. Actually we do not insert the points p_i into S since this might change the situation for points in S; we will use these *virtual* points to select a set of points which are considered when processing an insertion event. In the first sweep we only use the points p_1 and p_2, the other points $p_3, \ldots, p_8$ are used in the other sweeps analogously.
Unlike other plane-sweep algorithms (e.g. [BGH93, For87, GH93a, GH93b, HNS88, HNS92]) the only type of event which has to be processed are *insertions*. An insertion event for $p \in S$ has to be processed when the sweep-line reaches p. Insertion events which coincide in space can be processed in an arbitrary order. To process an insertion event we do the following: First insert p into the sweep-line structure. The question which has to be answered now is, which of the points in $QL(p)$ come into question for being a nearest foreign neighbor of p.
The points p_1 and p_2 are either virtual or non-virtual. However, there is no nearest foreign neighbor of p in $QL(p)$ with a x-coordinate greater than $p_1.x = p_2.x$. The L^t-circle $K_{d_\infty(p,\tilde{p})}(p)$ with center p and radius $d_\infty(p,\tilde{p})$ does not intersect the regions $Q_1 := \{(x,y) | x < p_1.x, y \geq p_1.y\}$ and $Q_2 := \{(x,y) | x < p_2.x, y \leq p_2.y\}$. Therefore we obtain the following

Lemma 2.1 *Nearest foreign neighbors of p in $QL(p)$ which are better than $\tilde{p}$ are contained in the region $R_p := \{(x,y) | x \leq p_1.x, p_2.y < y < p_1.y\}$.*

Here "better than $\tilde{p}$" means that the distance to p is strictly smaller than $d_t(p,\tilde{p})$. For each color c_i ($1 \leq i \leq m$) we have an initially empty dictionary set T_i. What we do now is the following: for the moment we do not matter about how to extract the points contained in R_p. Take all these points and insert those points into the color sets $T_{c(p)}$ which have a different color than p; then remove them all from the sweep-line structure. During the sweep we therefore maintain the following sweep invariant: If there is a nearest foreign neighbor q of p better than $\tilde{p}$ in $QL(p)$ which is still contained in the sweep-line structure immediately before the insertion event of p is processed, then q is inserted into $T_{c(p)}$.
Our sweep-line structure consists of two components each of which holds a set of points. When processing an insertion event of p the point p is inserted into both components. In the first component only the points r in R_p with $p.y \leq r.y < p_1.y$ are removed and in the second component only the points r in R_p with $p.y \geq r.y > p_2.y$ are removed when processing the insertion event of p. The two components of the sweep-line structure use common color sets T_i.
The sweep is complete after processing the last insertion event. After the four sweeps have been performed we complete the algorithm by determining a nearest neighbors of each point $p \in S$ in $T_{c(p)}$; e.g. this could be done by

point location in $\mathcal{V}(T_{c(p)})$. In section 4 we present a simpler and therefore better approach.
It remains to show how to extract the points of R_p. This can be done by holding the points in the sweep-line structure in form of *quadrant priority search trees* (QPSTs); this data structure is presented in [GH93a]. The QPST can be built such that the operation *YMinInQuadrant(v)* returns the y-maximal point contained in $\{(x,y)|\, x \leq v.x, y < v.y\}$ or the y-minimal point in $\{(x,y)|\, x \leq v.x, y > v.y\}$. Repeatedly perform this operation on the sweep-line structure's components with p_1 and p_2 as query points, insert the result points into $T_{c(p)}$ if necessary and delete them from the concerned components. At termination when the result points are outside of the rectangles of interest all points in R_p have been examined and correctly deleted from the sweep-line structure's components. A detailed analysis shows that the algorithm PSANFN requires linear space and solves the ANFN problem for a point set with a total of n points in optimal time $O(n \log n)$.

3 Correctness

In this section we prove the correctness of the algorithm PSANFN. We have to show that for all points $p \in S$ the algorithm computes a nearest foreign neighbor correctly. W.l.o.g. we restrict ourselves to the first sweep from left to right and the case that a nearest foreign neighbor of p is contained in the quadrant $QL(p)$.
Split the quadrant into part $QL_a(p)$ which contains all points of $QL(p)$ that are "above" p and part $QL_b(p)$ which contains all points of $QL(p)$ that are "below" p. As described in the previous section the sweep-line structure consists of two components which share common color sets T_i. These components correspond to the parts $QL_a(p)$ and $QL_b(p)$ in the following way: We show that if a nearest foreign neighbor q of p is contained in $QL_a(p)$ then the first sweep-line component will contribute the point q to the color set $T_{c(p)}$, i.e. during the sweep q is deleted from the first sweep-line component and is inserted into $T_{c(p)}$. Before we prove this we give two lemmas; the proofs are omitted and can be found in the full paper [GH93c].
Lemma 3.1 *Let $r, q \in \mathbb{R}^2$ such that $q \in QL_a(r)$. Then the half-line h starting at point $(r.x, r.y + r.x - q.x)$ with slope $+1$ is part of the L^∞-bisector $B_\infty(q,r)$ for $x \geq r.x$. For $1 \leq t < \infty$ the part of $B_t(q,r)$ for $x \geq r.x$ extends above h.*
Lemma 3.2 *Let $p, q, r \in \mathbb{R}^2$ such that $q \in QL_a(p) \cap QL_a(r)$ and $r.x \leq p.x$. Then for all $1 \leq t \leq \infty$ the following implications hold:*

$$r.x \leq p.x \Rightarrow d_t(p,r) \leq d_t(p,q) \qquad r.x < p.x \Rightarrow d_t(p,r) < d_t(p,q)$$

Fix an arbitrary point $p \in S$. W.l.o.g. we assume that a nearest foreign neighbor q of p is contained in $QL_a(p)$. If $d_t(p,q) = d_t(p,\tilde{p})$ a nearest foreign neighbor of p has already been found during the preprocessing step and nothing remains to be shown. We therefore assume that $d_t(p,q) < d_t(p,\tilde{p})$. The idea of the proof is to show that q will be inserted into $T_{c(p)}$ either before or when processing the insertion event of p; this sufficies since then a nearest foreign neighbor of p is found during the final phase of the algorithm. If q has not yet removed from the first component of the sweep-line structure when the insertion event for p is processed then q is inserted into $T_{c(p)}$. If q has already been removed from the first component of the sweep-line structure when the insertion event of p is processed then Theorem 3.3 guarantees that a nearest foreign neighor of p is found during the final phase of PSANFN.
Theorem 3.3 *If q has been removed from the first component of the sweep-line structure before the insertion event of p is processed then q has already been inserted into $T_{c(p)}$.*
Proof: The assumption that q has already been removed from the first component of the sweep-line structure implies that there exists a point $r \in S$ which has been processed before p, i.e. for which $r.x \leq p.x$, such that $q \in QL_a(r)$ and q has been removed from the first component of the sweep-line structure when processing r. If $c(p) = c(r)$ then q has been inserted into $T_{c(p)}$. We may therefore assume that $c(p) \neq c(r)$.
The points p, q, r fulfill the conditions of Lemma 3.2 which implies that $d_t(p,r) \leq d_t(p,q)$. If $p.x = r.x$ then $d_t(p,r) = d_\infty(p,r) \geq (p,\tilde{p})$ and $d(p,q) < d_\infty(p,\tilde{p})$ by assumption which implies $d_t(p,r) > d(p,q)$. This leads to a contradiction since either $|p.y - q.y| \geq |p.y - r.y|$ if $r.y > p.y$ or $|p.x - q.x| \geq |p.y - r.y|$ if $r.y < p.y$. Hence we may assume $p.x > r.x$. In this case Lemma 3.2 shows that $d_t(p,r) < d_t(p,q)$ which is a contradiction to the choice of q. This completes the proof. □

4 Two-color nearest foreign neighbors

In this section we show how to avoid point location when searching for a nearest neighbor of p in $T_{c(p)}$. The idea is that point location with each point of a set in a Voronoi diagram of an other set of points can be understood as an ANFN problem for two different colors where one is only interested in nearest foreign neighbors for the points in the query set. In the following we consider the ANFN problem for the case of two different colors and we therefore

set $k = 2$, i.e. $S = S_1 \cup S_2$. Throughout this section let d_t denote the Minkowski-distance for an arbitrary but fixed $1 \leq t \leq \infty$. For $1 < t < \infty$ the bisector $B_t(p,q)$ of two points $p, q \in \mathbb{R}^2$ can be interpreted as continuous function which are either non-increasing or non-decreasing. This does not hold for $B_1(p,q)$ if the rectangle determined by p and q is a square. In this case the bisector $B_1(p,q)$ consists of two regions and a line segment ([Lee80]). The same is true for the L^∞-metric if p and q have equal y-coordinates. We solve this problem as it is shown in [Lee80].
In [HNS92] Hinrichs et al. present a plane-sweep algorithm for the Euclidean all-nearest-neighbors problem. Although some of their arguments do not apply to other metrics than the Euclidean metric the algorithm can be generalized to work with arbitrary L^t-metrics. As mentioned in [HNS92] the algorithm can be easily modified to solve the ANFN problem for two colors by a split of the y-table which is also true for arbitrary L^t-metrics. We only describe some parts of the generalized version of [HNS92], details about the algorithm and its implementation can be found in [GH93c, HNS92]. Consider a triple (p,q,r) of active points that are consecutive with respect to $<_y$, i.e. $p <_y q <_y r$ and no other point lies between them. If the triple is collinear, i.e. the points p, q, r lie on a line, then the bisectors $B_t(p,q)$ and $B_t(q,r)$ do not intersect as it is shown in Lemma 4.1. Hence in this case no deactivation event for q has to be generated. The proofs of Lemma 4.1, 4.2, and 4.3 are omitted and can be found in [GH93c].

Lemma 4.1 *If the triple (p,q,r) is collinear then the bisectors $B_t(p,q)$ and $B_t(q,r)$ do not intersect.*

The triple (p,q,r) forms a right turn iff r is to the right of the oriented line $\overline{p,q}$.

Lemma 4.2 *Let (p,q,r) be a triple of points that form a right turn. Then for any intersection point v of the bisectors $B_t(p,q)$ and $B_t(q,r)$ we have $v.x \geq \min\{p.x, q.x, r.x\}$.*

Lemma 4.2 implies that if a triple (p,q,r) of points in the y-table forms a right turn a deactivation event for q might have to be generated. It is easy to verify that this is not necessarily the case.
By a mirroring argument Lemma 4.2 also implies that for a triple (p,q,r) of points in the y-table which form a left turn the bisectors $B_t(p,q)$ and $B_t(q,r)$ do not intersect at or to the right of $\max\{p.x, q.x, r.x\}$. Hence in this case no deactivation event for q has to be generated.
Processing of the y-table as in [HNS92] remains correct which follows from Lemma 4.3.

Lemma 4.3 *Given a set S of points sorted according to $<_y$ and the set of bisectors of all consecutive triples (p,q,r). Fix an arbitrary triple (p,q,r). Let v be an intersection point of $B_t(p,q)$ and $B_t(q,r)$ then for any point u with $u.x > v.x$ we have $d_t(u,q) > \min_{p \in S} d_t(p,u)$*

A detailed analysis shows that this generalized algorithm requires linear space and solves the ANN problem for a point set with a total of n points in optimal time $O(n \log n)$.

References

[AERT89] A.Aggarwal, H.Edelsbrunner, P.Raghavan and P.Tiwari: Optimal time bounds for some proximity problems in the plane, *Information Processing Letters* 42, 55-60 (1992).

[BGH93] F. Bartling, Th. Graf, K. Hinrichs: A plane-sweep algorithm for finding a closest pair among convex planar objects, *Preprints Angewandte Mathematik und Informatik*, Universität Münster, Bericht Nr. 1/92 - I

[For87] S. Fortune: A Sweep line Algorithm for Voronoi Diagrams, *Algorithmica* 2, 153-174 (1987).

[FKP92] P.-O. Fjällström. J.Katajainen, J. Petersson: Algorithms for the all-nearest-neighbors problem, *Report 92/2*, Dept. of Computer Science, University of Copenhagen, Denmark, 1992.

[GH93a] Th. Graf, K. Hinrichs: Algorithms on colored point sets, *9th European Workshop on Computational Geometry CG '93*, Informatik-Berichte Fern-Uni Hagen Nr. 140 3/1993, 78-81, (1993).

[GH93b] Th. Graf, K. Hinrichs: A plane-sweep algorithm for finding nearest neighbors among convex planar objects, *Proceedings of the 3rd Workshop on Algorithms and Data Structures WADS '93, LNCS 709, Springer-Verlag*, (1993).

[GH93c] Th. Graf, K. Hinrichs: Simple search for nearest foreign neighbors in arbitrary L^t-metrics, *to appear in Preprints Angewandte Mathematik und Informatik*, Universität Münster, (1993).

[HNS88] K. Hinrichs, J. Nievergelt, P. Schorn: Plane-sweep solves the closest pair problem elegantly, *Information Processing Letters* 26, 255-261 (1988).

[HNS92] K.Hinrichs, J.Nievergelt, P.Schorn: An all-round sweep algorithm for 2-dimensional nearest-neighbor problems, *Acta Informatica*, 29(4), 383-394 (1992).

[Lee80] D. T. Lee: Two-Dimensional Voronoi Diagrams in the L^p-metric, *Journal of the ACM* 27(4), 604-618 (1980).

[LW80] D.T. Lee and C.K. Wong: Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications, *SIAM Journal on Computing* 9(1), 200-211 (1980).

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

[Sha75] M.Shamos, D.Hoey: Closest-point problems, *16th Annual IEEE Symposium on Foundation of Computer Science*, 151-162 (1975).

[Vai89] P.Vaidya: An $O(n \log n)$ algorithm for the all-nearest-neighbours-problem, *Discrete & Computational Geometry* 4, 101-115 (1989).

N-dimensional separation theorems in Digital Topology

R. Ayala* E. Domínguez** A.R. Francés**
A. Quintero* J. Rubio**

When a digital picture D appears on a computer screen, one usually wants to decompose D into its connected components, to find the holes in D, to detect the boundaries of D, etc,... That is, one wants to know as many "topological" properties of D as possible.

In order to state precisely topological properties of a digital picture in a coherent (and useful) way, we need to define an abstract structure on the screen which provides the good definitions and properties. In order words, we need a well founded Digital Topology.

At the present time the best known approach to Digital Topology is due to Rosenfeld [7]. His approach endowes the set of pixels of a planar screen with a natural adjacency relation which yields a graph E_8 in such a way that several topological properties of the digital picture can be translated into combinatorial properties of E_8.

In spite of its important practical applications, this model of Digital Topology is not completely satisfactory from a theoretical point of view. For instance, it is necessary to consider two notions of connection in order to prove a digital version of the Jordan Curve Theorem (a simple closed curve divides the plane into two connected components). In addition, those difficulties oblige to find specific proofs for the digital versions of well known topological results. Moreover, the weak relationship with ordinary topology makes difficult to state important topological notions (e.g. digital manifold).

Therefore, it seems interesting to develop other structures which ease the above troubles.

In this paper we give a fairly general formal approach to Digital Topology. This alternative approach was outlined in [2].

*Dpt. de Algebra, Computación, Geometría y Topología. Facultad de Matemáticas. Universidad de Sevilla. Apto. 1160. E-41080 - Sevilla. Spain.

**Dpt. de Ingeniería Eléctrica e Informática. Facultad de Ciencias. Universidad de Zaragoza. E-50009 - Zaragoza. Spain.

A *screen model* is a pair $(\mathcal{D}_K, K)$ where K is a purely n-dimensional locally finite polyhedral complex together a set $\mathcal{D}_K$ consisting of one interior point $b(\sigma^n)$ for each n-cell $\sigma^n \in K$. We recall that a polyhedral complex is a set of polytopes (i.e. convex cells) with the usual properties of a simplicial set.

In a naive way, the points of $\mathcal{D}_K$ correspond to the pixels of the screen, and a n-cell of K is the bright halo which appears when the corresponding pixel is lighted. So, we can naively regard $\mathcal{D}_K$ as the screen of the $(\mathcal{D}_K, K)$ screen model (possibly an infinite screen!).

The adjacency of the n-cells in K defines a graph $\mathcal{L}_K$ with vertices in $\mathcal{D}_K$ which is called the *logical level* of the screen model. Notice that when K is the natural polyhedral complex on $\mathbb{R}^2$ defined by the lattice $\mathbb{R} \times \mathbb{Z} \cup \mathbb{Z} \times \mathbb{R}$, the graph $\mathcal{L}_K$ is the graph E_8 used by Rosenfeld in [7].

Furthermore, the polyhedral structure of K allows us to consider much more elements than the pixels in the screen $\mathcal{D}_K$. More explicitly, we can form a directed graph $\mathcal{C}_K$ (the *conceptual level*) associated to the screen model as follows. The set of vertices of $\mathcal{C}_K$ consists of $\mathcal{D}_K$ together with one vertex for each cell of K which is intersection of two or more n-cells in K. The directed edges of $\mathcal{C}_K$ are pairs of cell (v, w) with v a face of w.

If $\mathcal{O}(\mathcal{D}_K)$, $\mathcal{O}(\mathcal{L}_K)$, and $\mathcal{O}(\mathcal{C}_K)$ denote the lattices of all subsets of $\mathcal{D}_K$ and full subgraphs of $\mathcal{L}_K$ and $\mathcal{C}_K$ respectively, we can define the following natural transformations

$$(1)\quad \mathcal{O}(\mathcal{D}_K) \xrightarrow{i} \mathcal{O}(\mathcal{L}_K) \underset{p}{\overset{p^*}{\rightleftarrows}} \mathcal{O}(\mathcal{C}_K) \xrightarrow{k} \mathcal{O}(K)$$

where i is induced by the identification of $\mathcal{D}_K$ with set of vertices of $\mathcal{L}_K$. For $G \in \mathcal{O}(\mathcal{L}_K)$, $p(G)$ is the subgraph generated by the vertices in G, together with the vertices in $\mathcal{C}_K$ which are the intersections of two or more n-cells corresponding to those vertices. The map p^* carries the directed graph H to the graph generated by the vertices in H corresponding to the n-cells in K. Finally, if $\mathcal{O}(K)$ is the lattice of subcomplexes of K the map k is induced by the canonical embedding of $\mathcal{C}_K$ in the barycentric subdivision sdK of K.

The diagram (1) is called the *digital space* of the screen model $(\mathcal{D}_K, K)$. A *digital object* O of the screen model is a subset $O \subseteq \mathcal{D}_K$. The *TOP-components* of O in $\mathcal{L}_K$ are the graphs $p^*(C)$ where C is a connected component of $p(i(O))$. The *DIG-components* of O are the set $i^{-1}(A)$ where A is a TOP-component. Notice that a TOP-component needs not to agree with a connected component of $i(O)$ in $\mathcal{L}_K$.

The *continuous analogue* of a digital object O is the order complex $\mathcal{A}_O = \Delta(p(i(O)))$ of the directed graph $p(i(O))$ considered as partially ordered set

(see [1]). That is, $< x_0, x_1, \ldots, x_r >$ is a r-simplex in $\mathcal{A}_O$ if $x_0 \leq x_1 \leq \ldots \leq x_r$ is a directed path in $p(i(O))$. Notice that when $K$is the polyhedral complex associated to the planar lattice $\mathbb{R} \times \mathbb{Z} \cup \mathbb{Z} \times \mathbb{R}$, the continuous analogue $\mathcal{A}_O$ agrees with the polyhedron $C(O)$ in [3].

It can be easily checked that the set of DIG-components of O are in 1-1 correspondence with the set of connected components of $\mathcal{A}_O$. In fact a DIG-component A of O is the set of n-cells in K corresponding to the vertices in $p(i(O)) \cap C$ where C is a connected component of $\mathcal{A}_O$. Therefore $\sigma^n \in A$ if and only if $b(\sigma^n) \in C$.

We say that a digital object O in the screen model is a *digital n-manifold (with boundary)* if its continuous analogue $\mathcal{A}_O$ is a combinatorial n-manifold (with boundary). When $\mathcal{A}_O$ only is a topological manifold, O is called a *weak digital n-manifold.*

In particular, O is a digital closed 1-manifold if and only if O is a digital Jordan Curve. That is, O is a sequence of n-cells $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ such that σ_i is adjacent to σ_j if and only if $|i - j| \leq 1$. In general, a digital n-sphere (n-ball) is any digital object O such that $\mathcal{A}_O$ is a combinatorial n-sphere (n-ball resp.).

Within this theoretical framework we can easily give the following general Jordan Separation Theorem (cf. [4] in dimension 3)

Jordan Theorem. Let K be a polyhedral complex on $\mathbb{R}^n$, then for any (n-1)-sphere O in the screen model $(\mathcal{D}_K, K)$ the complement of O has two DIG-components (one of them bounded A_b and the other unbounded A_∞).

In addition our framework can be used to solve problems which have not been considered yet. We finish this paper with a general digital Schöenflies Theorem which in its simplest form states that any closed curve bounds a digital disk. Obviously, this result is connected with the problem of recognizing holes in a digital picture.

Schöenflies Theorem. With the notation of the above theorem, the union $O \cup A_b$ is a weak digital n-ball. When $n \leq 3$ $O \cup A_b$ is in fact an n-digital ball.

Proof Let C_b (C_∞) be the bounded (unbounded) connected component of $K - \mathcal{A}_O$. It will suffice to show that $\mathcal{A}_{O \cup A_b} = \mathcal{A}_O \cup C_b$, since $\overline{C}_b = \mathcal{A}_O \cup C_b \subseteq sdK$ is a subcomplex which triangulates a topological n-ball by Schöenflies's Theorem (see [6], and [8]). For $n \leq 3$, $\overline{C}_b$ is in fact a combinatorial ball by [5].

Let $\gamma = (b(\sigma_1), b(\sigma_2), \ldots, b(\sigma_m))$ be any simplex in $\overline{C}_b$. Since K is an n-manifold without boundary, each σ_i is defined as an intersection of

n-simplices in K. Clearly $\mathcal{A}_O \subseteq \mathcal{A}_{O \cup A_b}$, so we can assume that $b(\sigma_i) \in C_b$ for some $1 \leq i \leq m$. If τ is an n-simplex with $\sigma_i \subseteq \tau$, and $\tau \in A_\infty$ then $b(\tau) \in C_\infty$. Thus the segment $\overline{b(\sigma_i)b(\tau)}$ is a 1-simplex in sdK which yields a contradiction since $\overline{C}_b$ is a subcomplex of sdK. Therefore, the simplices which contain σ_i are in $O \cup A_b$, and by definition of continuous analogue $\gamma \in \mathcal{A}_{O \cup A_b}$.

Conversely, given the simplex $\gamma = (b(\sigma_1), \ldots b(\sigma_m)) \in \mathcal{A}_{O \cup A_b}$, we know that σ_i is intersection of n-simplices $\tau \in O \cup A_b = \mathcal{D}_K - A_\infty$. Thus $b(\tau) \in \overline{C}_b$. We claim $b(\sigma_i) \in \overline{C}_b$ for $1 \leq i \leq m$. Otherwise, assume $b(\sigma_i) \in C_\infty$. If some $\tau \notin O$ then $b(\tau) \in C_b$ and we get the same contradiction as above. So, each τ necessarily belongs to O and we get the new contradiction $b(\sigma_i) \in \mathcal{A}_O \cap C_\infty$

Finally, if $x \in \overset{\circ}{\gamma} \cap C_\infty$ with $\overset{\circ}{\gamma}$ is the interior of γ, we have $\gamma \in \overline{C}_\infty = C_\infty \cup \mathcal{A}_O$ since $\overline{C}_\infty$ is a subcomplex. Thus , $b(\sigma_i) \in \mathcal{A}_O = \overline{C}_b \cap \overline{C}_\infty$ and $\gamma \in \mathcal{A}_O$ by definition of continuous analogue. So, $x \in \mathcal{A}_O \cap C_\infty$, a contradiction. We have proven $\overset{\circ}{\gamma} \subseteq \overline{C}_b$ and thus $\gamma \subseteq \overline{C}_b$.

Similarly to PL Topology the following problem still remains in Digital Topology

Schöenflies Problem. Is $O \cup A_b$ a digital n-ball ($n \geq 4$)?.

The difficulty is 4-dimensional: if true for $n = 4$ then the answer is yes for all n (see [8]).

REFERENCES

[1] M.M. Bayer. *Barycentric subdivisions.* Pacific J. Math. 135(1988) 1-16.

[2] E. Domínguez, A.R. Francés, A. Márquez. *A Framework for Digital Topology.* Int. Conf. on Systems, Man, and Cybernetics. Proc. 1993 IEEE, 93CH 3242-5, vol. 2. 65-70.

[3] T.Y. Kong, R. Litherland, A.Rosenfeld. *Problems in the Topology of Binary Digital Images.* Open problems in Topology. J. van Mill, G.M. Reed, eds. North-Holland, 1990.

[4] R. Kopperman, P . Meyer, R.G. Wilson. *A Jordan surface theorem for three-dimensional digital spaces.* Discrete Comp. Geom. 6(1991) 155-161.

[5] E.E. Moise. *Geometric Topology in Dimension 2 and 3.* GTM 47. Springer 1977.

[6] M.H.A. Newman. *On the subdivision of euclidean n-space by topological (n-1)-spheres.* Proc. Roy. Soc. 257(1960) 9-12.

[7] A. Rosenfeld. *Digital Topology.* Amer. Math. Monthly 86(1979), 621-630.

[8] C. Rourke, B. Sanderson. *Introduction to Piecewise Linear Topology.* Springer, 1972.

An $O(n)$-Time Rectangle Placement Algorithm (Extended Abstract)

Patrick Healy
Department of Computer Science and Information Systems
University of Limerick
Limerick
Ireland

31 January 1994

Abstract

In this paper we consider the problem of determining whether a given rectangle can be placed in a given layout without overlapping any rectangle that is already placed. We show that it is possible to find a location (if one exists) in time linear in the number of rectangles already placed. This extends previous work that considered the case of finding a suitable location for a rectangle when the layout had been constructed according to the bottom-left placement heuristic.

1 Introduction

The *rectangle layout* problem occurs in a variety of settings. In addition to applications in the glass and lumber industries [4] where it is called *stock cutting*, it has applications in resource allocation problems [6] as well as in VLSI layout [7]. One version of the problem is the following: Given a *stock sheet* $S = (W, H)$ of dimensions $W \times H$ and a set of n rectangles, $R = \{r_j : r_j = (w_j, h_j), 1 \leq j \leq n\}$ (each rectangle, r_j, having width w_j and height h_j), find a placement of rectangles from R in S that minimizes the unoccupied area of S.

The *bottom-left* placement strategy has been shown to be an effective heuristic for this problem although its worst-case performance is poorer than other heuristics [1]. The bottom-left (BL) heuristic attempts to place a rectangle as low as possible in the stock sheet and in the event of ties as far to the left as possible. It has been shown to yield reasonably good results when the rectangles are initially sorted by decreasing width and then placed[1] in that order.

Chazelle [2] presents an algorithm that, given a list of rectangles, places each one in turn according to the BL-heuristic. The running time of the algorithm is $O(n^2)$, with $O(n)$ time required for each placement. A key requirement of his algorithm is that the layout have the bottom-left property at each stage. While this is reasonable in the case of an on-line algorithm, an off-line algorithm may want to deform the layout in some way and place a rectangle in a layout that does not have this property [5]. Below we present an algorithm that finds and reports the set of feasible locations for a given rectangle in an arbitrary layout with the only restriction being that the layout be topologically equivalent to the unit circle. That is, every rectangle is connected to every other rectangle by some set of "bridging" rectangles.

1.1 Terminology

[1] A rectangle is placed only if there is a feasible location at which to place it.

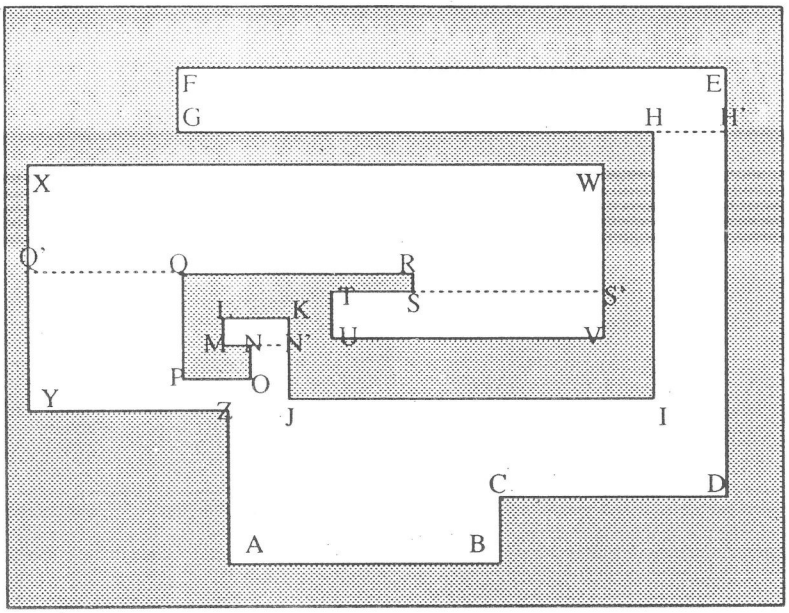


Figure 1: A layout with a hole containing a right notch and three left notches.

Figure 1 illustrates a partial layout. The darkened area represents already placed rectangles and the blank area in the center is referred to as a *hole*. The hole is defined in terms of horizontal and vertical *intervals* which separate the layout from the hole. Thus an interval has a rectangle on one side and the hole on the other. Staying with Chazelle's notation, the term *notch* refers to three consecutive intervals, the first and third of which are parallel and perpendicular to the second and the internal angle they make with the second is reflex. If the second interval of the three intervals of a notch has a rectangle to the left then we refer to it as a left notch. Top, right and bottom notches are defined likewise.

Since a BL-layout has the property that every rectangle is bottom and left justified, a BL-layout cannot have a top (respectively, right) notch for, informally, if it did the notch could be slid down (to the left) until it was supported by some other rectangle.

We call a hole *nice* if it doesn't contain any left or right notches. A nice hole also has the property that it has exactly one *left interval* and exactly one *right interval*. A left or right interval is a vertical interval on the hole boundary with horizontal intervals on either side of it and the external angles are both reflex. A left (right) interval is the leftmost (rightmost) interval in a hole. We call the sequence of intervals above the left and right intervals the *roof* and those below, the *floor*. That is, if we traverse the hole boundary clockwise starting from the left interval, the roof of a subhole comprises all those intervals traversed until the right interval is encountered; similarly for the floor. The roof (floor) of a nice hole has the important property that when traversed in a clockwise (anti-clockwise) order the x-coordinates of the intervals' endpoints increase monotonically.

We represent a hole internally by a circular linked list of intervals as they appear when traversing the perimeter of the hole. So that we may traverse it either clockwise or anti-clockwise quickly we make the list doubly linked.

2 The Algorithm

Given a rectangle, $r = (w, h)$, we firstly find $\mathcal{L}$, the locus of left endpoints of a horizontal line segment of length w so that the line segment fits entirely within the hole, H. From this set we need only find a location

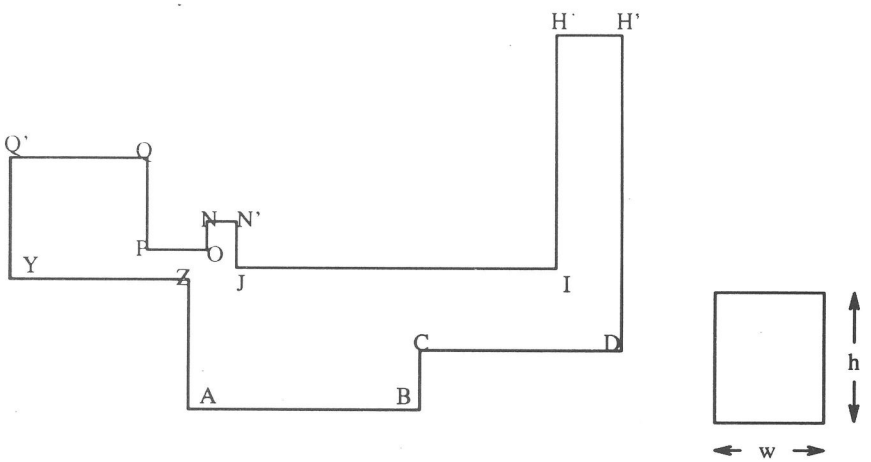


Figure 2: A subhole derived from partitioning the hole above with an unplaced rectangle.

for a vertical line segment of length h. Such a point will be a feasible location for the placement of r in H. The algorithm is easily modifiable to return a location other than the bottom-left location.

In order to find $\mathcal{L}$ we must firstly partition the hole H in to subholes if it contains left or right notches. Figure 1 illustrates a hole that contains a right notch (intervals OP, PQ, QR) and three left notches (intervals GH, HI, IJ; MN, NO, OP and QR, RS, ST). By traversing the hole boundary and by observing the orientation and direction of the hole intervals we can detect a notch. By convention we start all of our boundary traversals from the bottom-most horizontal interval of the hole.

When we detect a left (right) notch we project a horizontal line rightwards (leftwards) from the end of the vertical interval that it shares with the third interval of the notch until it reaches the hole perimeter. We call this endpoint the *notch point* and the new point created by its projection the *projection* of the notch. We call the line that connects the projection to the notch the *partition* since it will partition the hole into two subholes. In figure 1 the Q' is the projection of the right notch and H', N' and S' are the projections of the left notches.

Lemma 1: All notch projections may be found in $O(n)$ time.

We can use the result in [3] to find the projections in linear time.

Theorem 1: A hole may be partitioned into nice subholes in $O(n)$ time.

Constructive Proof: With the notch points and their projections known, we can partition the hole into nice subholes as follows. Proceed around the hole in clockwise order. When a notch is encountered, split the interval containing its projection into two. In figure 1 we would split the interval XY into XQ' and $M'Y$. Now by rearranging the "next" and "prev" pointers of the intervals on the interval list and introducing the interval QQ' we can break the hole's interval list into two separate circular lists. We call the interval so introduced a *partition interval*. A similar traversal of each of these subholes in the clockwise direction will detect any remaining notches and partition them accordingly.

After the hole has been partitioned we consider each partitioned subhole in turn, finding the locus of left endpoints for a line segment of length w. Figure 2 illustrates a subhole derived from the partitioning of the hole in figure 1. Also shown is a rectangle of dimensions $w \times h$ which we wish to place.

The locus of left endpoints can be found by "tracing" a line segment of width w around the perimeter, looking ahead w when necessary. This can be done in $O(n)$ time and is shown shaded in figure 3. As before

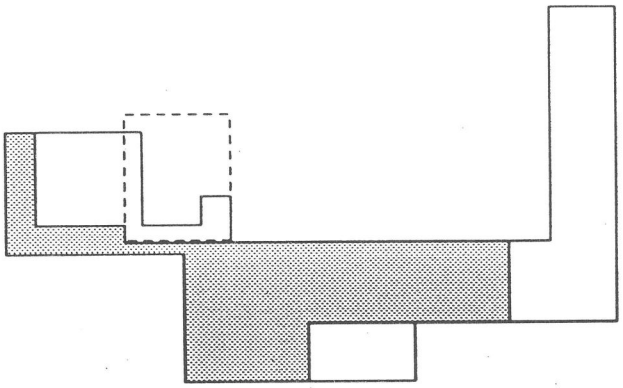


Figure 3: The locus of left endpoints of a line segment of length w.

the locus is represented by a doubly linked list of intervals on the locus boundary. We maintain a two-way link between the locus interval and the interval on the subhole boundary that contains it. (If the subhole interval is also a partition interval then it may have links to two such locus intervals.) The purpose of these pointers is to allow us to coalesce the locii into maximally connected regions, which we describe below. Note that the locus for a given subhole may comprise two or more disjoint regions.

Theorem 2: The maximally connected locus of feasible points for a horizontal line segment of length w can be found in $O(n)$ time.

Constructive Proof: With each subhole's locus determined we coalesce them with their neighboring subholes to form maximally connected regions. This is achievable by traversing the boundary of each locus and checking if the interval is associated with a partition interval. If it is and there is a locus interval on the other side of the partition then the two regions can be coalesced into one larger region. This entire process can be performed by traversing each locus interval once.

With this set of maximally connected regions we have determined every feasible location for placing a horizontal line segment of length w. If there is a location in this set that will also accommodate a vertical line segment of length h then we are done. Such a location can be found by rotating the regions we have found by 90° and now finding a location for a horizontal line segment of length h. This proves the following theorem.

Theorem 3: The set of all feasible locations for placing a rectangle of dimensions $w \times h$ may be determined in $O(n)$ time.

Once a location for a rectangle has been found, it is straightforward to place the rectangle in that hole in $O(n)$ time.

3 Justifications

- We can show that the number of intervals in a layout is linear in the number of placed rectangles;
- Since there can be only $O(n)$ notches in total the number of partition intervals is also linear in the number of rectangles;
- Now there cannot be more intervals defining the locus of left endpoints than there were intervals to begin with. Therefore the locus of left endpoints is linear in the number of rectangles in the layout;

- Since the second half of the algorithm replicates the steps above, the final set of feasible locations is $O(n)$ in the number of rectangles.

References

[1] B. S. Baker, E. G. Coffman, Jr, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM J. Comput.*, 9(4):846–855, November 1980.

[2] Bernard Chazelle. The bottom-left bin-packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, C-32(8), August 1983.

[3] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

[4] Nicos Christofides and Charles Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, Jan-Feb 1977.

[5] Patrick Healy. *Sacrificing: An Augmentation Of Local Search*. PhD thesis, University of Massachusetts, Amherst, May 1991.

[6] J. D. Ullman. Complexity of sequencing problems. In E. G. Coffman, editor, *Computer and Job-Shop Scheduling Theory*. John Wliey and Sons, New York, 1975.

[7] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. In *23rd Design Automation Conference*, pages 101–105, 1986.

Intersection of Straight Lines with Algebraic Surfaces in NC-SAVE*

Mario Dias†
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
39071 Santander, Spain

ABSTRACT

We present an implementation of a method for intersecting straight lines with algebraic surfaces in the NC-SAVE system on the Sequent Symmetry. NC-SAVE is a modeler for the simulation and verification of NC-machining using the *dexel* representation. This modeler is also used for the encoding and visualization of algebraic surfaces and provides many facilities, like zooming, varying the light sources, having several windows with different view points, hidden surface removal and shading.

NC-SAVE

NC-SAVE (Simulation and Verification Environment for NC-machining) represents the workpiece and the worktools in 3-dimensional space. It shows how the individual processing steps generate the final workpiece from the raw stock. It enables an interactive control of the simulation by a multi-window technique, free choice of the view points and light sources, zoom functions,

*This work was supported by the Austrian Ministery of Science and Research in the frame of the project CEI/PACT.

†Currently at RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria.

discrete or continuous image sequences, etc. Furthermore, it provides technological information in a separate window like, e.g., average and maximal loss (removal) of the material, present cutting speed and tool coordinates. Among the many advantages of this modeler, it should be mentioned that the 3D-animation is very realistic.

DEXEL MODELLING

A possibility to design a model that allows the efficient execution of boolean operations is to spatially subdivide an object into many primitive elements such that the boolean operations can be performed very quickly on these primitives, e.g. octree-encoding. This technique uses the cube as the only primitive. One disadvantage of octree-encoding is that it requires a large amount of storage for non-planar and non-orthogonal objects.

The *dexel model* is a modelling scheme that overcomes this negative aspect of octree-encoding and besides it is of non hierarchical type and it is regular in screen dimensions. In 1986, van Hook [3] sketched this model for the first time (see also [2]). The name "dexel" is an abbreviation of "depth element". This method decomposes an object in the following way: A set of paralell and equidistant rays are intersected with the object to be encoded. Two points defining a line segment that lies fully inside the object make up a dexel. All dexels belonging to one ray are sorted and concatenated to a dexel list. All these dexel lists are organized in a dexel matrix. Generally the dexel list is a list of intervals. No problems occur if the object to be encoded is a surface. In this case the intervals have length zero and therefore collapse to a single point. Since the object is encoded normal to the view plane, all visible parts are accessible immediately.

The main advantages of using this model for the representation of algebraic surfaces are the following:

- The level of precision can be predefined.
- Once all objects are kept spatially presorted according to a specific direction, the visible surface can be displayed very quickly with automatic hidden surface removal. Furthermore, the dexel structure allows realistic 3D shaded images.

- Boolean operations can be performed very efficiently, because all these operations can be reduced to the corresponding operations on dexels. Therefore this method allows an easy visualization of the intersection of two surfaces and the intersection of a surface and a plane respectively.

Since the dexel model is very efficient for the generation of realistic images it is suitable for the visualization of surfaces.

CONCLUSION

Our aim is the implementation of a method in order to encode a surface by the dexel model which is done by intersecting a set of parallel and equidistant rays with the algebraic surface. This encoding for surfaces of degree 3 was already motivated in [1].

BIBLIOGRAPHY

[1] Rudolf Eisenstöck, *Visualization of Algebraic Surfaces Using the Dexel Model*, Diploma thesis, RISC-Linz, J. Kepler University, A-4040 Linz, Austria (1992).

[2] Herwig Mayr, Johann Heinzelreiter, *A Solid Modeler for Dynamic Objects Using the Dexel Representation*, Technical Report, RISC-Linz, J. Kepler University, A-4040 Linz, Austria (1990).

[3] Tim Van Hook, *Real-Time Shaded NC-Milling Display*, In David C. Evans and Russell J. Athay, editors, Computer Graphics, volume 20, pages 15-20 ACM/SIGGRAPH (1986).

On the Number of Permutations of Point Sets Generated by Plane Sweeps and Sphere Sweeps

Peter-Michael Schmidt*

December 1993

(Extended Abstract)

The sweep method is a well–known technique in Computational and Combinatorial Geometry. In most cases the d–dimensional space $\mathbb{R}^d$ is swept by a hyperplane which makes a translational movement. Many problems in Computational Geometry can be easily and elegantly solved by means of *plane sweeps* (cf. [BeOt79], [BiNe83], [Fort87]). The sweep stops at certain event points in order to process local informations. That means, that the sweep induces an order in the set M of all event points. For the design of numerically robust algorithms it may be useful to know all orders of M generated by sweeps.

In case of $d = 2$ the sweep of a topological line is also an useful tool for solving certain problems (cf. [EdSo88], [EdGu89]). Conceivably, the Euclidean plane is swept by other curves as for instance circles. In the d–dimensional space we call such a sweep a *sphere sweep*. We assume that the center of the sweep sphere never changes and the radius grows from 0 to ∞.

The talk presents upper bounds to the number of all orders of finite point sets M of $\mathbb{R}^d$ generated by translational plane sweeps (abbreviated by P–sweep) or sphere sweeps (S–sweep). The orders of M are described by permutations of the index set $\{1,\ldots,n\}$.

1 Generating Permutations by Sweeps

Let $M = \{p_1,\ldots,p_n\}$ be a n–point set of $\mathbb{R}^d$ with $(n \geq d \geq 1)$. We assume $\mathbb{R}^d$ to be provided with its natural basis. $\langle\cdot,\cdot\rangle$ denotes the scalar product in $\mathbb{R}^d$ and $\delta(\cdot,\cdot)$ is the Euclidean distance. To define an P–sweep we call a nonconstant linear function $\sigma_P(x) = \langle\tau, x\rangle$ $(x \in \mathbb{R}^d)$ a sweep function. The sweep plane is given by $\sigma_P^{-1}(\tau)$. Running τ through $(-\infty,\infty)$ the hyperplane $\sigma_P^{-1}(\tau)$ makes a translational movement in the direction τ through $\mathbb{R}^d$. An S–sweep is analogously defined: Let $c \in \mathbb{R}^d$ be the common center of all sweep spheres. The sweep function is $\sigma_S(x) = \delta(c,x)$. Running τ through $[0,\infty)$ the sphere $\sigma_S^{-1}(\tau)$ sweeps $\mathbb{R}^d$. Since some of the following definitions and considerations apply to both, P–sweeps and S–sweeps, we will use the symbol K for both kinds of considered sweeps.

*Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik, Mathematisches Institut, Universitätshochhaus, 17. OG, Leutragraben 1, D–07743 Jena, Germany, schmidt@mathematik.uni-jena.dbp.de

If σ_K is a sweep function then the permutation π of $\{1,\ldots,n\}$ is generated by an K–sweep iff $\sigma_K\left(p_{\pi(1)}\right) < \sigma_K\left(p_{\pi(2)}\right) < \cdots < \sigma_K\left(p_{\pi(n)}\right)$. The set of all permutations generated by K–sweeps is denoted by $\Pi_K(M)$. We are interested in the calculation of the following two combinatorial functions:

$$f_K(n,d) := \max\{\text{card } \Pi_K(M) : M \subset \mathbb{R}^d \wedge \text{card}(M) = n\} \text{ for } n \geq d \geq 1 \text{ and } K \in \{P,S\}$$

2 The assigned arrangements

To solve these two counting problems we assign to the point set M an arrangement of hyperplanes[1]. For any 2–set $\{p_i,p_j\} \subseteq M$ we define the hyperplanes $h_{ij} := \{x \in \mathbb{R}^d : \langle p_i - p_j, x\rangle = 0\}$ and $b_{ij} := \{x \in \mathbb{R}^d : \delta(p_i,x) = \delta(p_j,x)\}$. Obviously, h_{ij} is the set of all directions r of P–sweeps with $\sigma_P(p_i) = \sigma_P(p_j)$ and b_{ij} contains all centers c of S–sweeps with $\sigma_S(p_i) = \sigma_S(p_j)$. The arrangement of all h_{ij} $(1 \leq i < j \leq n)$ is denoted by $\mathcal{A}_P(M)$ and $\mathcal{A}_S(M)$ is the arrangement of all b_{ij}. Figure 1 shows $\mathcal{A}_P(M)$ for $M = \{p_1,\ldots,p_4\} \subset \mathbb{R}^3$. For the sake of clearness the points p_i are represented by the set of vertices of a simplex. This arrangement has an interesting structure: Any 3 points of M determine

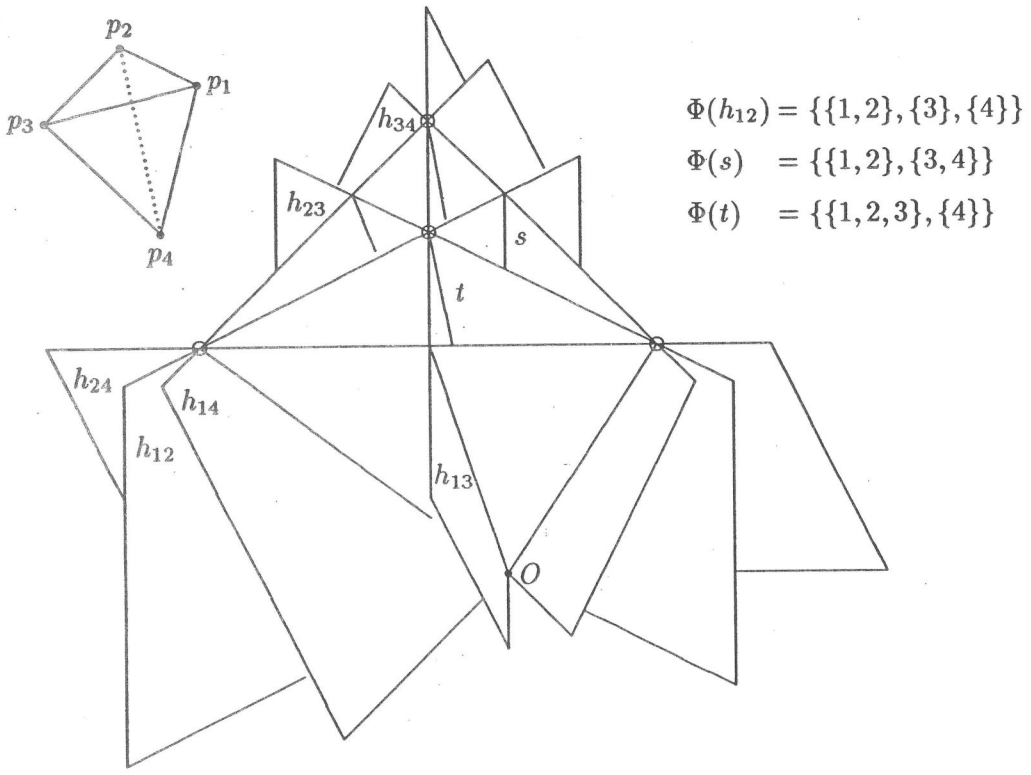


Figure 1: A 4–point set $M \subset \mathbb{R}^3$ and the assigned arrangement $\mathcal{A}_P(M)$

a hyperplane in $\mathbb{R}^3$. The perpendicular through the origin O is a 1–dimensional flat[2] of

[1] An d–arrangement is a finite set of hyperplanes of $\mathbb{R}^d$ together with the induced partition of $\mathbb{R}^d$ into k–dimensional cells (k–cells) for $0 \leq k \leq d$.

[2] The flats of an arrangement are the nonempty intersections of its hyperplanes.

$\mathcal{A}_P(M)$ (for instance, p_1, p_2, p_3 generate t). These flats are circled in Figure 1. Such a flat is contained in 3 hyperplanes. The arrangement $\mathcal{A}_S(M)$ for a planar 4–point set M is shown in Figure 2. This arrangement has an analogous property: The centers of the circle through 3 points of M are contained in 3 lines. In Figure 2 these points are denoted by $m_{i,j,k}$.

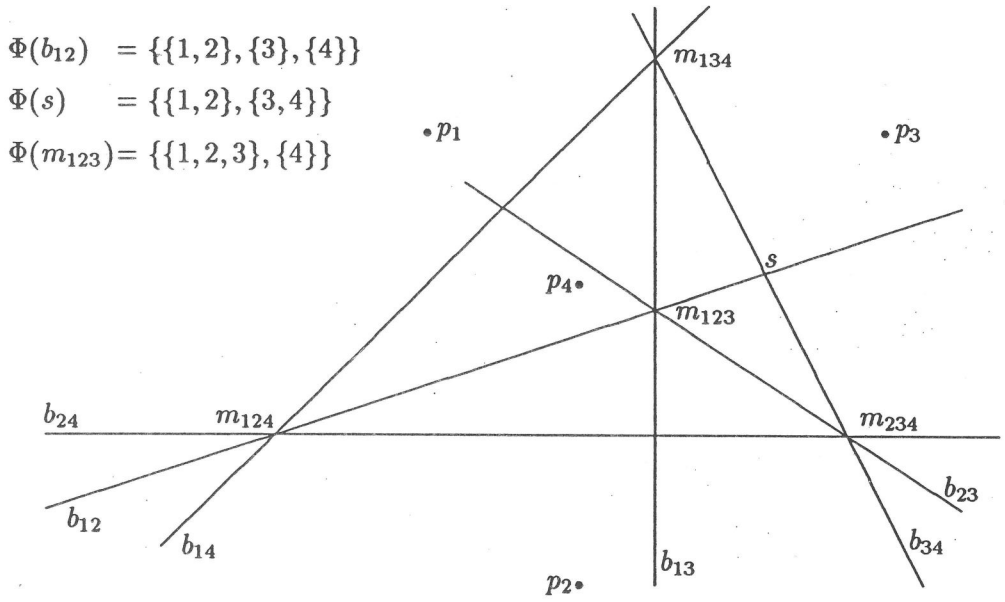


Figure 2: A planar 4–point set M and the arrangement $\mathcal{A}_S(M)$

Choosing r and c in distinct d–cells of $\mathcal{A}_P(M)$ and $\mathcal{A}_S(M)$, respectively, the corresponding K–sweeps generate also distinct permutations. Therefore the cardinality of $\Pi_K(M)$ is the number of all d–cells of $\mathcal{A}_K(M)$. The maximum numbers of d–cells are achieved by point sets M in general position. The notion of general position needed for the P–sweep is given in [GoPo86]. In case of S–sweep this notion must be similarly modified. From now on M is assumed to be in general position.

3 The induced partitions

Each K–sweep with a sweep function σ_K induces a partition $\{N_1, \ldots, N_{n-r}\}$ of $\{1, \ldots, n\}$ $(0 \leq r < n)$ by means of the equivalence relation $i \approx j :\Leftrightarrow \sigma_K(p_i) = \sigma_K(p_j)$ for $i, j \in \{1, \ldots, n\}$. Each partition generated by a sweep of the kind $K \in \{P, S\}$ belongs to

$$\begin{aligned} Z_P(n,d) &= \{\{N_1, \ldots, N_{n-r}\} : 0 \leq r < d \wedge \text{card}(N_i) \leq d \text{ for } i = 1, \ldots, n-r\} \\ Z_S(n,d) &= Z_P(n, d+1) \end{aligned}$$

and conversely, each partition of $Z_K(n,d)$ is generable by an K–sweep.

How it is shown in the figures the flats of $\mathcal{A}_K(M)$ correspond to partitions of $Z_K(n,d)$. This gives the rise to the following definition: A central d–dimensional arrangement $\mathcal{A}$ is

called an (n,d,P)–*arrangement* iff there is a bijection Φ between the set of all flats of $\mathcal{A}$ with dimension > 0 and $Z_P(n,d)$ satisfying

$$F_1 \subset F_2 \text{ iff } \Phi(F_2) \text{ is a refiniment of } \Phi(F_1) \text{ for all flats } F_1, F_2 \text{ of } \mathcal{A}. \qquad (\star)$$

An d–arrangement $\mathcal{A}$ is an (n,d,S)–*arrangement* iff there exists a bijection Φ between the set of all flats of $\mathcal{A}$ and $Z_S(n,d)$ with the property $(\star)$.

Let F be a flat of an (n,d,K)–arrangement. If the partition $\Phi(F)$ consists of $(n-r)$ blocks then $\dim(F) = d - r$ for $0 \leq r < d$ in case of $K = P$ and for $0 \leq r < d+1$ in case of $K = S$.

The following geometric fact leads to a recurrence for the $f_K(n,d)$: The arrangement $\{H_o \cap H : H \in \mathcal{A}\}$ is an $(n,d-1,K)$–arrangement if $\mathcal{A}$ is an (n,d,K)–arrangement and the intersection hyperplane H_o is in general position to all flats of $\mathcal{A}$.

4 Calculating the number of d–cells

The number $f_K(n,d)$ is equal to the number of d–cells of an arrangement $\mathcal{A}_K(M)$ which is an (n,d,K)–arrangement provided that M is in general position. Surprisingly, this number is uniquely determined by the parameter n,d and, of course, by K.

Theorem: The number of d–cells of an (n,d,K)–arrangement and the maximum number of permutations of an n–point set of $\mathbb{R}^d$ generated by K–sweeps ($K \in \{P,S\}$) is given by the recurrence

(a_P) $f_P(n,2) = n \cdot (n-1)$ for $n \geq 2$,

(a_S) $f_S(n,1) = \binom{n}{2} + 1$ for $n \geq 2$,

(b) $f_K(n,n) = f_K(n,n-1)$ for $n \geq 2$,

(c) $f_K(n+1,d) = n \cdot f_K(n,d-1) + f_K(n,d)$ for $n \geq d > 2$,

which yields $f_P(n,d) \in \Theta(n^{2(d-1)})$ and $f_S(n,d) \in \Theta(n^{2d})$ provided that d is fixed.

References

[BeOt79] Bentley, J.L., Ottmann, T.A.: *Algorithms for reporting and counting geometric intersections.* IEEE Transactions on Computers C-28 (1979), 643-647.

[BiNe83] Bieri, H., Nef, W.: *A sweep-plane algorithm for computing the volume of polyhedra represented in Boolean form.* Linear Algebra Appl. 52/53 (1983), 69-97.

[EdSo88] Edelsbrunner, H., Souvaine, D.L.: *Computing media-of-squares regression lines and guided toplogical sweep.* Techn. Report UIUCDCS-R-88-1438, Univ. of Illinois at Urbana-Champaign, Department of Computer Science 1988.

[EdGu89] Edelsbrunner, H., Guibas, L.J.: *Topological sweeping an arrangement.* Journal of Computer and System Sciences 38 (1989), 165–194.

[Fort87] Fortune, S.: *A sweepline algorithm for Voronoi diagrams.* Algorithmica 2 (1987), 153–174.

[GoPo86] Goodman, J.E., Pollack, R.: *Upper bounds for configurations and polytopes in* $\mathbb{R}^d$. Discrete Computational Geometry 1 (1986), 219–227.

Lower Bounds for Arithmetic Networks II: Sum of Betti Numbers

EXTENDED ABSTRACT

J.L. Montaña [2,3] *J.E. Morais* [1,3] *Luis M. Pardo* [1,3]

Abstract.- We show lower bounds for the parallel complexity of membership problems in semialgebraic sets. Our lower bounds are obtained from the Euler characteristics and the sum of Betti numbers. We remark that these lower bounds are polynomial in the sequential lower bounds obtained by Andrew C.C. Yao.

0. Introduction.

One of the classical complexity problems in computational geometry is the point location problem in a real euclidean space. For instance, for a non–negative integer $n \in \mathbb{N}$, let K_n be a simplicial complex in $\mathbb{R}^n$. Thus, the problem is stated as: given a point $\underline{x} \in \mathbb{R}^n$ decide whether $\underline{x}$ belongs to K_n. The usual model of computation accepts arithmetic operations, $\{+,-,\times,/\}$, and sign tests, $\{>0, =0, <0\}$, as elementary operations (cf. [16]). Thus, these problems can always be understood as membership problems in semialgebraic subsets $W_n \subseteq \mathbb{R}^n$.

Usually, the sequential model is described by real RAMS (cf. [16]) and real Turing machines (cf. [4], [6]). The non–uniform sequential model (required for an analysis of lower bounds) is given by sequences of algebraic computation trees. To every node of the tree we assign an alementary operation. Since the works of Dobkin and Lipton, [7], Steele and Yao, [13], and M. Ben-Or, [B-O], the analysis of lower bounds for sequential complexity has been oriented to relate the topology of the semialgebraic sets W_n to the complexity of the problem (cf. [12], [6]). The lower bounds there obtained were of the kind $\Omega(log_2\beta_0(W_n))$, where $\beta_0(W_n)$ is the number of connected components of W_n.

One of the most challenging problems posed by these authors was to relate higher homology groups to the complexity of these problems. Thus, in [3] and [14], the authors stated that a modified Euler characteristics can be used for that purpose. Lately, in [2] the authors showed that the sum of Betti numbers can be used for the restricted model of linear trees. Finally, A.C.C. Yao has shown in [15] that if the W_n are compact sets, the sum of Betti numbers can be used to provide lower bounds for sequential complexity.

In these pages we deal with an analysis of the parallel complexity of the same kind of problems. The parallel model, corresponding to that sequential model, is that of parallel real RAM's (pRAM) of parallel real Turing machines. The non–uniform model for the analysis of lower bounds is that of arithmetic networks over the reals (cf. [8] and [11]). This model improves the performance of the sequential one accepting simultaneous operations and sign tests.

In [11] the authors showed that the lower bounds for sequential time (based on connected components) are also lower bounds for the parallel time up to a square root. More concretely, [11] shows that $\Omega\left(\sqrt{\frac{log_2\beta_0}{n}}\right)$ is a lower bound for the parallel time, independently of the number of processors simultaneously working. Thus, it was conjectured that this feature holds whenever topological lower bounds are considered.

---

1 Departamento de Matemáticas, Estadística y Computación. Universidad de Cantabria.

2 Departamento de Matemáticas e Informática. Universidad Pública de Navarra.

3 Partially supported by DGCYT-PB 92-0498-C02-01 and "POSSO", ESPRIT-BRA 6846.

The main goal of the present pages is to state that this conjecture holds for Euler characteristics and the sum of the Betti numbers. Three are the main obstructions we deal with. First, avoid subadditivity since this property does not hold for both invariants. For the sum of the Betti numbers we introduce the Borel-Moore homology for locally closed semialgebraic sets. As for Euler characteristics we consider the invariant introduced by Yao and observe that it is the alternate sum of the Borel-Moore Betti numbers. The second obstruction is combinatorial. This requieres an analysis of the behaviour of the chosen topological invariant while increasing the parallel time. This is mainly inspired in [11]. Finally, to relate the given representation with parallel complexity requires combining the bounds of [10] with the analysis of non–empty cells done in [9].

We remark that our technics in the analysis of the sum of Betti numbers are different to those of [15]. The $i-$th Borel-Moore Betti number and the β_i', used in [15], are not equal in general.

Our main applications are to show an $\Omega(\sqrt{n})$ lower bound for the parallel time of the n–dimensional knapsack problem, an $\Omega(\sqrt{log_2 N})$ lower bound for the parallel time of the membership problem in a polygon with N vertices, and $\Omega(\sqrt{n/k})$ lower bound for the parallel time of the $k-$equal problem.

1. Preliminaries.

DEFINITION 1.1 . A subset $W \subseteq \mathbb{R}^n$ is called a *semi-algebraic cell*, if there exists polynomials $f_1, \ldots, f_s \in \mathbb{R}[x_1, \ldots, x_n]$ such that $W = \{\underline{x} \in \mathbb{R}^n \mid f_1(\underline{x})\epsilon_1 0, \ldots, f_s(\underline{x})\epsilon_s 0\}$ *where* $(\epsilon_1, \ldots, \epsilon_s) \in \{>, =\}^s$. A *semi-algebraic* set is a finite union of semi-algebraic cells.

For a semialgebraic subset W of $\mathbb{R}^n$, let $H_r(W, \mathbb{Z}_2)$ the $r-$th homology group, and $\beta_r(W)$ the rank of this vector space (that is usually called the $r-$th Bettit number of W). Let $H_*(W, \mathbb{Z}_2) = \bigoplus H_r(W, \mathbb{Z}_2)$ the total group of homology of W. Recall that the usual Euler characteristics is given by $\chi(W) = \sum(-1)^r \beta_r(W)$. We define the Borel-Moore homology of locally closed semialgebraic sets as follows (cf. [5], section 11.7).

DEFINITION 1.2 . Let W be a locally closed semi-algebraic set. We define the Borel–Moore homology of W, $H_r^{BM}(W, \mathbb{Z}_2)$, by

$$H_r^{BM}(W, \mathbb{Z}_2) = H_r(W, \mathbb{Z}_2) \quad \text{if W is compact,}$$

$$H_r^{BM}(W, \mathbb{Z}_2) = H_r(\dot{W}, \dot{W} \setminus \eta(W); \mathbb{Z}_2) \quad \text{otherwise}$$

where $(\dot{W}, \eta(W))$ is the Alexandrov semi-algebraic compactification of W (cf. [5], 2.5.9).

By coherence, let us denote $\beta_r^{BM}(W) = rank(H_r^{BM}(W, \mathbb{Z}_2)$ and $\beta^{BM}(W) = \sum \beta_i^{BM}(W)$.

In [14], the author modified the concept of Euler characteristics to have additivity. This modified Euler characteristics can be stated as follows. By Hironaka's triangulation theorem, for every bounded semialgebraic subset W of $\mathbb{R}^n$, there exists a semi-algebraic triangulation $(\mathcal{D}, h)$ of W. Assuming $\mathcal{D} = \{D_\alpha : \alpha \in A\}$ and $W = \bigcup_{\alpha \in B} h(\mathcal{D}_\alpha)$, (where $B \subseteq A$), we define the modified Euler characteristics of W as $\tilde{\chi}(W) = \sum_{\alpha \in B} (-1)^{dim(D_\alpha)}$.

If the semialgebraic set W is not bounded we define $\tilde{\chi}(W)$ to be $\tilde{\chi}(\varphi(W))$,where φ is the inverse of the stereographic projection form $S^n \setminus \{(0, \ldots, 0, 1)\}$ (S^n being the sphere with radius 1 in R^{n+1}). This $\tilde{\chi}$ is invariant under semialgebraic homeomorphism and it is an additive function (cf. [14])

Observe that if W is locally closed, $\tilde{\chi}(W) = \chi(\dot{W}) - 1$. Thus, from the long exact sequence for Borel-Moore homology (cf. [5], Prop. 11.7.15), we can conclude the following identity

$$\tilde{\chi}(W) = \sum_{r \geq 0} (-1)^r \beta_r^{BM}(W)$$

On the other hand, our non–uniform model of computation is given by sequences of arithmetic networks. An arithmetic network (cf. [8]) $\mathcal{N}$ is an arithmetic circuit (or straight line program) augmented with a boolean component (a boolean circuit) and an interface between them. This interface is given by two special gates, *sign gates* and *selection gates*. For a fixed sign condition $\epsilon \in \{>, =, <\}$, the sign gate outputs the following boolean values:

$$sign(f, \epsilon) = \begin{cases} \mathbf{T}, & \text{if } f \epsilon 0 \\ \mathbf{F}, & \text{otherwise.} \end{cases}$$

The selection gates choose a particular input according to a boolean instance. They have associated a function $sel : \mathbb{R} \times \mathbb{R} \times \{\mathbf{T}, \mathbf{F}\} \to \mathbb{R}$ defined by

$$sel(g, h, \mathbf{B}) = \begin{cases} g, & \text{if } \mathbf{B}=\mathbf{T} \\ h, & \text{otherwise.} \end{cases}$$

The concepts of depth and size of an arithmetic network are the natural extensions of the corresponding ones for boolean circuits and straigth line programs. The depth is the parallel time and the size is the numbers of processors required. We will denote by $D(\mathcal{N})$ the depth of an arithmetic network $\mathcal{N}$. Observe that selection gates may increase the number of polynomials involved in the computation up to a number which is doubly exponential in the depth

$$2^{2^{D(\mathcal{N})}}$$

An example of this feature can be seen in [11].

In this paper we will deal with *acceptor networks*, that is, networks with no arithmetic output nodes and only one boolean output node. So, an acceptor network decides the membership problem for a semi-algebraic set $\mathcal{W}(\mathcal{N})$ of $\mathbb{R}^n$.

2. Stating the main results.

Let $\mathcal{F} := \{f_1, \ldots, f_s\}$ a finite collection of polynomials and D the sum of their degrees. An $\mathcal{F}-$ cell is a semialgebraic cell $W \subseteq \mathbb{R}^n$ such that there is a sequence of sign conditions $\epsilon_1, \ldots, \epsilon_s \in \{>, =, <\}$ verifying

$$W = \{\underline{x} \in \mathbb{R}^n : f_1(\underline{x})\epsilon_1 0, \ldots, f_s(\underline{x})\epsilon_s 0\}$$

Thus, we have the following proposition.

PROPOSITION 2.1. . *With these conventions, we have:*

i) The sum of all the ranks of all the total Borel-Moore homology groups of the $\mathcal{F}-$ cells is of order $(1+D)^{O(n)}$.

ii) For every locally closed semialgebraic set W that can be given as a finite union of $\mathcal{F}-$ cells, $\beta^{BM}(W)$ is of order $(1+D)^{O(n)}$.

iii) The same holds for the modified Euler characteristics.

This proposition leads to the main lower bound technique by using an induction on the depth of the network.

THEOREM 2.2. . *Let W be a locally closed semialgebraic subset of $\mathbb{R}^n$. Let $\mathcal{N}$ be an acceptor network with inputs from $\mathbb{R}^n$ such that $W(\mathcal{N}) = W$. Then,*

$$D(\mathcal{N}) = \Omega\left(\sqrt{\frac{\log_2 \beta^{BM}((W)}{n}}\right)$$

COROLLARY 2.3. . *The same holds replacing* $\beta^{BM}(W)$ *by* $|\tilde{\chi}(W)|$ *above.*

The main applications are obtained by computing the respective invariants. Thus, consider the following examples.

KNAPSACK PROBLEM.- *Given* $(x_1,\ldots,x_n) \in \mathbb{R}^n$, *decide whether there is* $S \subseteq \{1,\ldots,n\}$ *such that*

$$\sum_{i\in S} x_i = 1.$$

$k-$ EQUAL PROBLEM.- *Given* $(x_1,\ldots,x_n) \in \mathbb{R}^n$, *decide whether there are* k *equal coordinates.*

The previous methods provide the following parallel lower bounds:

COROLLARY 2.4. . *With the previous convention we have:*

i) The parallel time required to solve knapsack problem is bigger than $\Omega(\sqrt{n})$.

ii) The parallel time required to solve the $k-$*equal problem is bigger than* $\Omega(\sqrt{log_2(n/k)})$.

References.

[1] M. Ben-Or. "Lower bound for algebraic computation trees", *A.C.M.* 15^th *Symposium on Theory of Computation.*, (1983) 80-86

[2] A. Björner and L.Lovász. " Linear decision trees, subspaces arrangements and Möbius function", *Preprint* (1992) .

[3] A.Björner, L.Lovász and A.Yao."Linear decision trees: volume estimates and topological bounds". In *Proc. A.C.M.* 24^th *Symposium on Theory of Computing,* , (1992) 170-177.

[4] L. Blum, M. Shub and S. Smale; "On a theory of computation and complexity over the real numbers: *NP*-completeness, recursive functions and universal machines". *Bulletin of the A. M. S.*, vol.**21**, n. 1 ,(1989) 1-46.

[5] J .Bochnack, M. Coste, M-F. Roy. "Géométrie algébrique réelle". Ergebnisse der Math., 3.Folge, Band **12**, Springer–Verlag, Berlin, Heidelberg, New York, (1987).

[6] F.Cucker, J.L. Montaña, L.M. Pardo."Time Bounded Computations over the reals". Int. J. of ALGEBRA AND COMPUTATION, Vol.**2**, No. 4, (1992) 395-408.

[7] D.P. Dobkin, R.J. Lipton."On the complexity of computations under barying sets of primitives". *J. of Computer System Scineces* **18**, (1979), 86-91.

[8] J.Von zur Gathen."Parallel arithmetic computations:a survey". In *Mathematical Foundations on computer Science,* 13^th *Proc.*,(1986) .

[9] J. Heintz."Definability and fast quantifier elimination over algebraically closed fields". Theor. Comp. Sci. **24**, (1983) 239-278.

[10] J.Milnor. "On the Betti numbers of real varieties". *Proc. Amer. Math. Soc.*, **15** (1964) 275-280.

[11] J.L.Montaña and L.M.Pardo."Lower bounds for Arithmetic Networks". *AAECC* **4** (1993) 1-24.

[12] J.L. Montaña, Luis M. Pardo y T. Recio." The Non-Scalar Model of Complexity in Computational Semialgebraic Geometry". In *Proc. MEGA'90, Progress in Mathematics* **94**, *Birkhäuser,* (1991) 346-362.

[13] M.Steele and A.Yao."Lower bounds for Algebraic Decision Trees". *J. Algorithms* **3** (1982) 1-8.

[14] A. Yao."Algebraic Decision trees and Euler characteristics". *Proceedings of* 33^rd *Annual IEEE on Foundations of Computer Science*, (1992) 268-277.

[15] A. C.C. Yao. "Decision trees and Betti Numbers". *To appear in Proc. FOCS'94* (1994).

[16] F.F. Yao." Computational Geometry". In *Handbook of Theoretical Computer Science.* J. van Leeuwen, ed., Elsevier, Amsterdam (1990) 343–391.

Splines and Pierce-Birkhoff Conjecture: the evaluation of continuous and piecewise polynomial functions

Laureano González-Vega[1,2]
Departamento de Matemáticas
Universidad de Cantabria
Santander 39071, Spain

Henri Lombardi[2]
Laboratoire de Mathematiques
Université de Franche-Comté
25030 Besançon, France

Introduction.

One of the powerful methods in Computer Aided Design to deal with curves in $\mathbb{R}^3$ is based in the use of spline curves. A spline curve s is a continuous map of a collection of intervals $\{[\alpha_i, \alpha_{i+1}] : i = 0, \ldots, m-1\}$ into $\mathbb{R}^3$, where each interval is mapped onto a polynomial curve segment. So, if $s: [\alpha_0, \alpha_m] \longrightarrow \mathbb{R}^3$ is a spline curve and $s = (s_1, s_2, s_3)$, then every s_k is an univariate continuous and piecewise polynomial function: i. e. there exist polynomials $P_1, \ldots, P_m$ in $\mathbb{R}[x]$ such that:

$$s_k(\omega) = P_j(\omega) \qquad \text{if} \quad \omega \in [\alpha_{j-1}, \alpha_j]$$

The extremes of the different intervals associated to s are called knots (see [5] for more details about spline curves). An important problem that appears dealing with spline curves is that usually the α_k's are real algebraic numbers and the P_j's are polynomials with rational coefficients which implies that the evaluation of the spline curve s is a hard problem, even in a rational point inside the interval $[\alpha_0, \alpha_m]$.

The main goal achieved in this paper is to show that there exists an algorithm providing a closed "rational" expression for every s_i that avoids these evaluation problems: we shall compute, beginning with the P_j's and the α_k's, a finite set of polynomials $F_{uv} \in \mathbb{Q}[x]$ such that:

$$\omega \in [\alpha_0, \alpha_m] \implies s_i(\omega) = \sup_u \inf_v F_{uv}(\omega)$$

i.e. we do not need to compare ω with any α_k in order to compute the value of $s_i(\omega)$.

The problem, so explained, is a particular case of a well-known open problem in Real Algebraic Geometry, the Pierce-Birkhoff Conjecture, which asks if every continuous and piecewise polynomial function over an ordered field can be represented by means of a sup-inf expression over a finite set of polynomials. This conjecture has been proved in the affirmative sense only for $n = 1$ and $n = 2$ (see [6] or [4]) and remains still open for $n \geq 3$.

Section II presents a new proof for the case $n = 1$ that simplifies the existing ones (see [6] or [4]) in the sense that, in general, our recursion process does not need to be applied over an exponential number of different cases. Moreover it remains still unsolved that our algorithm provides a polynomial time solution for the considered problem (see section III for a more detailed discussion about this question). This problem provides a new example for the optimal way of using Computer Algebra to deal with "real" or "practical" problems. Firstly, given the spline curve s, we use our algorithm to construct the sup-inf functions associated to s and after we apply this expression as a numerical subroutine in order to trace the spline curve s considered.

The algorithm to be presented in section II will use the fact that real algebraic numbers can be manipulated without loss of precision. This is achieved considering the Computer Algebra methods to deal with real algebraic numbers either using isolating intervals (see [2]) or Thom's coding (see [3], [1] or [7]).

I. Two rings of functions: rational semipolynomials and rational continuous piecewise polynomial functions.

We introduce in this section two kind of functions that will be our main study object: the $\mathbb{Q}$-semipolynomials (composition of polynomials and absolute value) and the continuous piecewise polynomial functions over $\mathbb{Q}$.

Definition I.1

The set of continuous piecewise polynomial functions over $\mathbb{Q}$, $\mathcal{PWP}$, is the set of functions $h: \mathbb{R} \longrightarrow \mathbb{R}$ such that there exist $P_1, \ldots, P_m$ in $\mathbb{Q}[x]$ and $-\infty = \alpha_1 < \alpha_1 < \ldots < \alpha_{m-1} < \alpha_m = \infty$ in $\mathbb{R}$ verifying for every $\omega \in \mathbb{R}$: $\quad h(\omega) = P_j(\omega) \qquad \text{if} \quad \omega \in [\alpha_{j-1}, \alpha_j]$.

1 Partially supported by CICyT PB 92/0498/C02/01 (Geometría Real y Algoritmos).

2 Partially supported by Esprit/Bra 6846 (Posso).

Definition I.2
The set of $\mathbb{Q}$-semipolynomials, $\mathcal{SP}$, is the set of functions $h:\mathbb{R} \longrightarrow \mathbb{R}$ such that there exists a finite set of polynomials f_{ij} in $\mathrm{K}[x]$ verifying:

$$\alpha \in \mathbf{R} \quad \Longrightarrow \quad h(\alpha) = \sup_i \inf_j f_{ij}(\alpha)$$

Clearly we have that $\mathcal{SP} \subseteq \mathcal{PWP}$ and the problem of proving the equality (for n arbitrary and considering an arbitrary ordered field $\mathbf{K}$ instead of $\mathbb{Q}$ and an arbitrary real closed field $\mathbf{R}$ containing $\mathbf{K}$ instead of $\mathbb{R}$) is the Pierce-Birkhoff conjecture (see [4] or [6]). Next proposition summarizes two standard properties of $\mathcal{SP}$ which will be very useful in the next section. Their proof can be found in [4] or [6].

Proposition I.3
$\mathcal{SP}$ is a commutative ring with unit equal to the set of functions $h:\mathbb{R} \to \mathbb{R}$ that can be described as a finite composition of polynomial functions with coefficients in $\mathbb{Q}$ and the function absolute value $|\bullet|$.

Given $P \in \mathbb{Q}[x]$, the two elements of $\mathcal{SP}$ defined by:

$$[P]^+ = \sup\{P, 0\} = (P + |P|)/2 \qquad [P]^- = \inf\{P, 0\} = (P - |P|)/2$$

will play an important role in the next sections.

II. The algorithm.

Let P be a polynomial in $\mathbb{Q}[x]$ with degree n, $\alpha_1, \ldots, \alpha_r$ the real roots of P, $\alpha_0 = -\infty$ and $\alpha_{r+1} = +\infty$. For every i in $\{0, \ldots, r+1\}$ we define the element of $\mathcal{PWP}$ given by:

$$\mathrm{C}_i(P) = \begin{cases} P(\omega) & \text{if } \omega \geq \alpha_i \\ 0 & \text{if } \omega \leq \alpha_i \end{cases}$$

Our first aim will be to show that every $\mathrm{C}_i(P)$ belongs to $\mathcal{SP}$ and to produce, in an algorithmic way, its representation as element of $\mathcal{SP}$. To achieve this goal we derive an expression of every $\mathrm{C}_i(P)$ in terms of $\mathrm{C}_{i-1}(P)$ and in terms of the C's functions associated to other polynomials of degree strictly smaller than n. Firstly we only need to study the case where P is squarefree because:

$$\widetilde{P} = P/\gcd(P, P^{(1)}) \quad \Longrightarrow \quad \mathrm{C}_i(P) = \gcd(P, P^{(1)})\mathrm{C}_i(\widetilde{P})$$

Also we can assume that $i \in \{1, \ldots, r-1\}$ because:

$$\mathrm{C}_0(P) = P \qquad \mathrm{C}_{r+1}(P) = 0 \qquad \mathrm{C}_r(P) = \begin{cases} [\mathrm{C}_{r-1}(P)]^+ & \text{if } \operatorname{lcof}(P) > 0 \\ [\mathrm{C}_{r-1}(P)]^- & \text{if } \operatorname{lcof}(P) < 0 \end{cases}$$

In the first step it is computed the quotient and remainder of the euclidean division of P with all its derivatives:

$$P(x) = P^{(u)}(x)Q_u(x) + R_u(x) \qquad \deg(R_u) < n - u \qquad u \in \{1, \ldots, n-1\}$$

Next it is computed the first index u (beginning in $n-1$ and decreasing one by one) such that the polynomial $P^{(u)}(x)$ has a real root, β_i^u, in the interval $[\alpha_i, \alpha_{i+1})$ (the smallest one in such interval). The existence of such index is assured by Rolle's Theorem. If γ_i^u is the first real root of $R_u(x)$ bigger or equal than β_i^u (if such root does not exist then $\gamma_i^u = +\infty$) then we define the element of $\mathcal{SP}$ given by:

$$E_i = Q_u \mathrm{C}_j(P^{(u)}) + \mathrm{C}_k(R_u)$$

where j and k are the indices of position for β_i^u and γ_i^u as roots of $P^{(u)}$ and R_u.

The definition of the C's functions allows to state that:

$$E_i(\omega) = \begin{cases} 0 & \text{if } \omega \leq \beta_i^u \\ P(\omega) - R_u(\omega) & \text{if } \beta_i^u \leq \omega \leq \gamma_i^u \\ P(\omega) & \text{if } \gamma_i^u \leq \omega \end{cases}$$

Depending on the sign of $P(\beta_i^u)$ we get the following cases for the situation between $P(x)$ and $E_i(x)$:

a-. if $P(\beta_i^u) = 0$ then $\alpha_i = \beta_i^u = \gamma_i^u$ which gives directly:

$$\mathrm{C}_i(P)(\omega) = E_i(\omega) = \begin{cases} P(\omega) & \text{if } \omega \geq \alpha_i \\ 0 & \text{if } \omega \leq \alpha_i \end{cases}$$

b-. if $P(\beta_i^u) \neq 0$ then defining

$$H_i(x) = \begin{cases} \sup\{P(x), E_i(x)\} & \text{if } P(\beta_i^u) > 0 \\ \inf\{P(x), E_i(x)\} & \text{if } P(\beta_i^u) < 0 \end{cases}$$

it is obtained the following equality:

$$H_i(\omega) = \begin{cases} [P(\omega)]^{\mathrm{sign}(P(\beta_i^u))} & \text{if } \omega \leq \alpha_i \\ P(\omega) & \text{if } \omega \geq \alpha_i \end{cases}$$

In case (a) we have $\mathrm{C}_i(P) = E_i$ as we wanted and it is only needed to study the case (b). If $P(\beta_i^u) > 0$ then we define the element of $\mathcal{SP}$:

$$T_i(x) = \inf\{[\mathrm{C}_{i-1}(P)(x)]^+, H_i(x)\}$$

which is equal to $\mathrm{C}_i(P)$:

- if $\omega \leq \alpha_{i-1}$ then $H_i(\omega) = [P(\omega)]^+$, $\mathrm{C}_{i-1}(P)(\omega) = 0$ and $T_i(\omega) = \inf\{0, [P(\omega)]^+\} = 0$,
- if $\alpha_{i-1} \leq \omega \leq \alpha_i$ then $P(\omega)$ is negative (P is square-free and positive in the next interval), $H_i(\omega) = 0$, $[\mathrm{C}_{i-1}(P)(\omega)]^+ = [P(\omega)]^+ = 0$ and $T_i(\omega) = \inf\{0, 0\} = 0$,
- if $\alpha_i \leq \omega$ then $H_i(\omega) = P(\omega)$, $\mathrm{C}_{i-1}(P)(\omega) = P(\omega)$ and $T_i(\omega) = \inf\{[P(\omega)]^+, P(\omega)\} = P(\omega)$.

If $P(\beta_i^u) < 0$ then defining:

$$T_i(x) = \sup\{[\mathrm{C}_{i-1}(P)(x)]^-, H_i(x)\}$$

it is obtained also that T_i agrees with $\mathrm{C}_i(P)$. Next proposition summarizes what we have proved.

Proposition II.1
Let Q be a polynomial in $\mathbb{Q}[x]$ of degree n with r different roots in $\mathbb{R}$ and $P \in \mathbb{Q}[x]$ its squarefree part. Then for every $i \in \{1, \ldots, r-1\}$:

$$\mathrm{C}_i(Q) = \begin{cases} \gcd(P, P^{(1)})E_i & \text{if } P(\beta_i^u) = 0 \\ \gcd(P, P^{(1)})\inf\{[\mathrm{C}_{i-1}(P)]^+, \sup\{P, E_i\}\} & \text{if } P(\beta_i^u) > 0 \\ \gcd(P, P^{(1)})\sup\{[\mathrm{C}_{i-1}(P)]^-, \inf\{P, E_i\}\} & \text{if } P(\beta_i^u) < 0 \end{cases}$$

where:

$$E_i(x) = Q_u(x)\mathrm{C}_j(P^{(u)}) + \mathrm{C}_k(R_u)$$

with $P^{(u)}$ the derivative of smallest degree with a root $\beta_i^u \in [\alpha_i, \alpha_{i+1})$ (the smallest one in such interval), Q_u and R_u quotient and remainder in the division of P and $P^{(u)}$, γ_i^u the smallest real root of R_u bigger or equal than β_i^u and j and k the indices of position for β_i^u and γ_i^u as roots of $P^{(u)}$ and R_u.

Theorem II.2
Let Φ be an element of $\mathcal{PWP}$. Then there exists $\Psi \in \mathcal{SP}$ such that for every $\omega \in \mathbb{R}$: $\Phi(\omega) = \Psi(\omega)$

Proof:
Let $P_1, \ldots, P_m$ be the polynomials defining Φ and $\alpha_1 < \ldots < \alpha_m$ the corresponding knots. For every $i \in \{1, \ldots, m-1\}$ we define the polynomial in $\mathbb{Q}[x]$ given by:

$$\Delta_i(x) = P_{i+1}(x) - P_i(x)$$

The continuity of Φ implies that every α_i is a root of $\Delta_i(x)$ and let $s(i)$ the position index of α_i as root of Φ. In these conditions we define the element of $\mathcal{SP}$ given by:

$$\Psi = P_1 + \sum_{i=1}^{m-1} \mathrm{C}_{s(i)}(\Delta_i)$$

The definition of the C's functions provides directly the equality between Ψ and Φ. ∎

It is important to remark here that the proof of the last theorem provides the algorithm to compute, for every $\Phi \in \mathcal{PWP}$, the corresponding $\Psi \in \mathcal{SP}$ such that $\Phi = \Psi$. Such algorithm consists simply of a finite number of calls to the construction of the C's functions. The recursion phase in the algorithm computing such functions finishes when we arrive to a degree 0 polynomial but it is worthly to remark here that such phase can be ended when the polynomial we are dealing with has only one real root. The reason is very simple, if P is a polynomial in such conditions and $\widetilde{P}$ is the squarefree part of P then we have:

$$\mathrm{C}_0(P) = P \qquad \mathrm{C}_1(P) = \begin{cases} \gcd(P, P^{(1)})[\widetilde{P}]^+ & \text{if lcof}\,(\widetilde{P}) > 0 \\ \gcd(P, P^{(1)})[\widetilde{P}]^- & \text{if lcof}\,(\widetilde{P}) < 0 \end{cases} \qquad \mathrm{C}_2(P) = 0$$

III. Complexity, examples and conclusions.

The main difference between the method presented here to compute the C's functions and the one used in [6] or [4] is that the formulae there derived for $\mathbf{C}_i(P)$ depends on $\mathbf{C}_{i-1}(P)$, $\mathbf{C}_j(P^{(1)})$ and $\mathbf{C}_k(R[P])$ where:

$$R[P] = P - \frac{x}{\deg(P)} P^{(1)} \qquad \deg(R[P]) \leq \deg(P) - 1$$

This implies that, in order to compute $\mathbf{C}_i(P)$, we need to apply this formulae $O(2^n)$ times (exactly 2^n times if P contains a monomial with every degree smaller than n) with $n = \deg(P)$.

Two are the keys of our algorithm. The first one is the selection of the derivative $P^{(u)}$ with smallest degree having a real root inside the interval $[\alpha_i, \alpha_{i+1})$ what implies that $\mathbf{C}_i(P)$ will depend on $\mathbf{C}_j(P^{(u)})$ and not on $\mathbf{C}_j(P^{(1)})$. The second one is that we replace the $R[\bullet]$ operator by the computation of the remainder between P and $P^{(u)}$ decreasing also the degree with respect to the algorithm showed in [4] or [6], even in the case when $u = 1$. Moreover it remains still unsolved the problem of showing that the complexity of the algorithm C is polynomial. The main problem we have found, dealing with this question, is that the number of recursion calls depends strongly on the distribution of the real roots of P and its derivatives.

Example

In order to show the way of working for the algorithms presented in the previous section, it is given here one detailed example of how to compute the C's functions associated to the polynomial:

$$P = x^3 - 3x + 1$$

To compute the semipolynomials $\{\mathbf{C}_i(P)\}_i$ it is easy to see that P is squarefree and has three real roots $(\alpha_1 < \alpha_2 < \alpha_3)$. Following the steps in the algorithm C we get firstly that $\mathbf{C}_0(P) = P$. To obtain $\mathbf{C}_1(P)$ we observe that the real root of $P^{(2)} = 6x$ is in the interval (α_1, α_2). As P is positive over such real root we get:

$$\mathbf{C}_1(P) = \inf\{[x^3 - 3x + 1]^+, \sup\{[x]^+(x^2 - 3), (x^3 - 3x + 1)\}\}$$

To compute $\mathbf{C}_2(P)$ we observe that the second real root of the polynomial $P^{(1)}$ is in the open interval (α_2, α_3). As P is negative over such real root then:

$$\mathbf{C}_2(P) = \sup\{[\mathbf{C}_1(P)]^-, \inf\{x^3 - 3x + 1, \tfrac{x}{3}\mathbf{C}_2(P^{(1)})\}\}$$

Working in a similar way with $P^{(1)}$ we get:

$$\mathbf{C}_2(P^{(1)}) = 3[\sup\{[x^2 - 1]^-, \inf\{x[x]^-, x^2 - 1\}\}]^+$$

providing as result for $\mathbf{C}_2(P)$ the following semipolynomial:

$$\mathbf{C}_2(P) = \sup\{[\mathbf{C}_1(P)]^-, \inf\{x^3 - 3x + 1, x[\sup\{[x^2 - 1]^-, \inf\{x[x]^-, x^2 - 1\}\}]^+\}\}$$

The computation of the C's functions associated to P is finished defining:

$$\mathbf{C}_3(P) = [\mathbf{C}_2(P)]^+ \qquad \mathbf{C}_4(P) = 0$$

References.

[1] F. Cucker, L. González Vega and F. Roselló: *On algorithms for real algebraic plane curves.* Effective Methods in Algebraic Geometry. Progress in Mathematics 94, 63–88, Birkhauser (1991).

[2] R. Loos and G. E. Collins: *Real zeros of polynomials.* Computer Algebra. Computing Suplementum 4, 83–95, Springer-Verlag (1982).

[3] M. Coste, M.-F. Roy: *Thom's Lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets.* Journal of Symbolic Computation 5, 121–129 (1988).

[4] C. N. Delzell: *On the Pierce-Birkhoff conjecture over ordered fields.* Rocky Mountain Journal of Mathematics 19, 651–668 (1989).

[5] G. Farin: *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press (1988).

[6] L. Mahe: *On the Pierce-Birkhoff conjecture.* Rocky Mountain J. of Mathematics 14, 983–985 (1984).

[7] M.-F. Roy and A. Szpirglas: *Complexity of computation on real algebraic numbers.* Journal of Symbolic Computation 10, 39–51 (1990).

Classifying Hyperplanes in Hypercubes (Extended Abstract)

Oswin Aichholzer[1] Franz Aurenhammer

Institute for Theoretical Computer Science
Graz University of Technology
Klosterwiesgasse 32/2
A-8010 Graz, Austria
e-mail: oaich@igi.tu-graz.ac.at
auren@igi.tu-graz.ac.at

1 Introduction

Among the simplest high-dimensional geometric objects is the d-dimensional hypercube (d-cube) $C^d = [0,1]^d$. Dispite of its simple definition, C^d has been an object of study from various different points of view. The theory of convex polytopes provides classical results concerning sections and projections of hypercubes; see Coxeter [3] and Grünbaum [6]. Purely combinatorial properties of C^d, mainly involving certain subgraphs formed by its edges and vertices (the latter are just the various d-tuples of binary digits) have been investigated extensively in coding theory and in communication theory; see, e.g., [4, 2, 11, 5]. Many easily stated questions concerning the geometry of C^d are still unsettled. A long-standing elementary conjecture on hypercube space fillings (Keller's conjecture) has been recently disproved [7].

In the last years, increased interest in problems involving the placement of hyperplanes in hypercubes can be observed. Motivation stems, among other areas, from coding theory [11] and from the theory of machine learning [1, 9, 10]. Again, several elementary questions turn out to be surprisingly difficult: What is the minimum number of (non-axis-parallel) hyperplanes that cover all the 2^d vertices of C^d ? How many edges of C^d may be cut by a single hyperplane? How many and which types of hyperplanes can be spanned by vertices of C^d ?

The present paper is devoted to the third question. Let us denote with $\mathcal{H}(C^d)$ the set of hyperplanes that can be spanned by (d affinely independent) vertices of C^d. An attempt is made to characterize and classify the hyperplanes in $\mathcal{H}(C^d)$. An obvious criterion is *parallelism*, where we distinguish between k-parallel hyperplanes (being parallel to exactly $k > 0$ coordinate axes) and skew (0-parallel) hyperplanes. A concept that turns out to be important is *symmetry*, which is fulfilled by hyperplanes that halve the set of d-cube vertices they avoid.

We show that k-parallel hyperplanes in $\mathcal{H}(C^d)$ correspond to skew ones in $\mathcal{H}(C^{d-k})$, and that skew and non-symmetric hyperplanes in $\mathcal{H}(C^d)$ correspond to skew and symmetric ones in $\mathcal{H}(C^{d+1})$, in a unique manner. This tells us that the skew and symmetric type is the only really interesting one. The enumeration of all hyperplanes of this type, however, is complicated by the following unpleasant phenomenon: In each further dimension new kinds of hyperplanes appear that seem to be not accessible by using the results for lower

[1]Research supported by the Jubiläumsfond der Österreichischen Nationalbank and the Austrian Ministery of Science.

dimensions. In fact, a theorem by Naumann [8] gives some evidence for the intrinsic complexity of $\mathcal{H}(C^d)$: Every $(d-1)$-polytope is obtainable as the intersection of a $(d-1)$-dimensional hyperplane and a hypercube of sufficiently high dimension. We were able to enumerate experimentally the sets up to $\mathcal{H}(C^8)$, but did not succeed in developing an enumeration scheme for $\mathcal{H}(C^d)$ for general d. An upper bound on the size of the normal vector of hyperpanes in $\mathcal{H}(C^d)$ is offered. Generally, we see our contributions as the first steps towards a systematic study of the set $\mathcal{H}(C^d)$.
The situation gets strikingly simpler if we restrict attention to hyperplanes in $\mathcal{H}(C^d)$ that do not (properly) cut d-cube edges. Those might be called *hull-honest* hyperplanes as their intersection figure with C^d coincides with the convex hull of the vertices of C^d they contain. We give an easy-to-use characterization of hull-honest hyperplanes that leads us to their exact total number, in general dimensions.

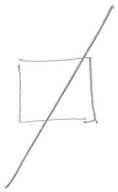
2 Overview of Results

Our set of interest, $\mathcal{H}(C^d)$, can be partitioned into classes by considering parallelism of its hyperplanes to coordinate axes. Let us call a hyperplane $H \in \mathcal{H}(C^d)$ *k-parallel* $(0 \leq k \leq d-1)$ if H is parallel to exactly k axes of $\mathbf{R}^d$, that is, to exactly k spanning edges of C^d. 0-parallel hyperplanes are called *skew*. For $d > 0$ and $0 \leq k < d$, let $e_k(d)$ denote the number of k-parallel hyperplanes in $\mathcal{H}(C^d)$. It is possible to relate e_k to e_0 (the number of skew hyperplanes) in a simple way.

(1) For $d > 0$ and $0 \leq k < d$ we have $e_k(d) = \binom{d}{k} e_0(d-k)$.

Note that we can thus restrict attention to skew hyperplanes, as each non-skew hyperplane has its skew analog in some dimension lower.
Next we study in which ways a hyperplane $H \in \mathcal{H}(C^d)$ may partition the vertex set V^d of C^d. To this end, we call H a (j,p,k)-*plane* if it contains $p \geq d$ vertices and splits the rest into subsets of cardinalities j and k where $j \leq k$. A (j,p,k)-plane is called *symmetric* if $j = k$, i.e., if the vertices of the d-cube which are avoided by the hyperplane are partitioned into equal-sized subsets. One might conjecture that a valid triple (j,p,k) uniquely determines a hyperplane in $\mathcal{H}(C^d)$, up to hypercube symmetries. This is false already in dimension 5: For instance, the two hyperplanes $H = ((-3,1,1,1,1),0)$ and $H' = ((-1,-3,1,2,2),0)$ in $\mathcal{H}(C^5)$ are both (11,5,16)-planes. However, H cuts 10 edges of C^5 in their interior while H' does this 13 times. Dimension 5 is the first where this phenomenon occurs which turns out to be frequent in higher dimensions. This already reveals part of the difficulty of classifying the hyperplanes in $\mathcal{H}(C^d)$. An important property is the following:

(2) There exists a one-to-one correspondence between skew non-symmetric hyperplanes with p vertices in $\mathcal{H}(C^d)$ and skew symmetric hyperplanes with $2p$ vertices in $\mathcal{H}(C^{d+1})$.



The merit of (2) is that attention can be restricted to skew symmetric hyperplanes for further classification. Note also that the number of vertices covered by a symmetric hyperplane $H \in \mathcal{H}(C^d)$ is at least $2(d-1)$.
A completly different way to classify the hyperplanes in $\mathcal{H}(C^d)$ is by considering their

It can be shown that the absolute value of the sum of v's entries has the same bound as a single entry.

To get an idea of the behaviour of the numbers and types of hyperplanes in $\mathcal{H}(C^d)$ for dimensions $d \geq 4$ we did some numerical investigations. We computed all these hyperplanes in dimensions up to 8. Their numbers turned out to increase superexponentially with d (roughly 2^{d^2}), as one would expect from the trivial upper bound $\binom{2^d}{d}$. (Each d-tuple chosen from the 2^d vertices of C^d potentially gives rise to a hyperplane.) An exhaustive enumeration for dimensions higher than 8 currently seems to be out of reach. For $\mathcal{H}(C^8)$ our program, though exploiting theoretical results reported in the present paper, needed about 12 days on a DEC 5000/240, and for $\mathcal{H}(C^9)$ we estimate a running time of about 35 years.

The experimental results, on the one hand, led us to observe structural properties of $\mathcal{H}(C^d)$ which could be made rigerous, but on the other showed us the intrinsic difficulty of classifying d-cube hyperplanes. In each further dimension we encountered new types of hyperplanes which could not be brought into connection with hyperplanes in lower dimensions. Although we only needed to deal with skew symmetric hyperplanes this did not really help, since their growth is superexponential, too, and they prevail over all other kinds of hyperplanes. For instance, in $\mathcal{H}(C^8)$ about 99.3 percent of all hyperplanes are skew.

Most hyperplanes are spanned by very few vertices. In $\mathcal{H}(C^8)$ about 30 percent cover only the minimum of 8 vertices. A correlation between the number of vertices covered by a hyperplane, its degree of parallelism, and the size of its normal vector can be observed: The smaller the normal vector and the higher the degree of parallelism, the more vertices may be covered. Saks [10] showed that the maximum number of vertices covered by a skew hyperplane is $\binom{d}{\lfloor \frac{d}{2} \rfloor}$. Fact (5) on skew hull-honest hyperplanes tells us that this bound is attainable. It is not difficult to deduce that a k-parallel hyperplane can cover not more than $\binom{d-k}{\lfloor \frac{d-k}{2} \rfloor}$ vertices. Note that $(d-1)$-parallel hyperplanes cover exactly half of the vertices of C^d, the maximum possible.

Dimension d	Number of hyperplanes in $\mathcal{H}(C^d)$	Number of skew hyperplanes $e_0(d)$	Number of different types of skew hyperplanes	Number of skew hull-honest hyperplanes $h_0(d)$
1	2	2	1	2
2	6	2	1	2
3	20	8	1	8
4	140	88	3	24
5	3254	2704	8	64
6	252434	234688	35	160
7	71343208	69640192	219	384
8	86246755608	85682904704	1293	896
9	?	?	?	2048
10	?	?	?	4608

Table of numbers of different kinds of hyperplanes in the d-cube.

References

[1] Noga Alon and Zoltán Füredi, Covering the Cube by Affine Hyperplanes, to appear

surface of intersection with C^d. To this end, $H \in \mathcal{H}(C^d)$ is called a *hull-honest* hyperplane if the convex hull of $H \cap V^d$ coincides with the polytope $H \cap C^d$. Expressed in different terms, H cuts edges of the d-cube only at their vertices but not in their interior. In fact, in dimensions two (square) and three (cube) each spanned line (plane) is hull-honest. But starting with dimension four, spanned hyperplanes can properly intersect edges of the d-cube, and the convex hull of all the vertices lying on the hyperplane is no longer the surface of intersection of hyperplane and d-cube. Hull-honest hyperplanes seem to be the only type in $\mathcal{H}(C^d)$ which has been studied previously, motivated by the intersection polytopes they generate [3]. We now give an easy-to-use criterion to characterize hull-honest hyperplanes, and we also give the exact number of such hyperplanes, in general dimensions.

(3) Let $H \in \mathcal{H}(C^d)$ be a hyperplane through the origin (which is no loss of generality) and let $v = (v_1, \ldots, v_d)$ be the (shortest integer) normal vector of H. Then H is hull-honest iff $v_i \in \{-1, 0, 1\}$ for $i = 1, \ldots, d$.

(4) For $d \geq 2$ and $1 \leq p \leq d-1$ there always exists a skew hull-honest hyperplane $H = (v, 0) \in \mathcal{H}(C^d)$ with exactly p arbitrarily chosen entries of v being $+1$.

(3) and (4) implies that, for a given number of positive entries in the normal vector, there is a unique skew and hull-honest hyperplane in $\mathcal{H}(C^d)$, up to hypercube symmetries. (In fact, the corresponding intersection figures turn out to be well-known polytopes [3].) In particular, there is a unique symmetric, skew, and hull-honest hyperplane in every even dimension and no such one in odd dimensions. More detailly, we have:

(5) Let $H = (v, 0) \in \mathcal{H}(C^d)$, $d \geq 2$, be skew and hull-honest and let p, $1 \leq p \leq d-1$, entries of v be $+1$. There are $\binom{d}{p}$ such hyperplanes H and each of them contains exactly $\binom{d}{p}$ vertices of C^d.

(6) The number of skew hull-honest hyperplanes in $\mathcal{H}(C^d)$ is given by

$$h_0(d) = \begin{cases} 2 & d = 1, \\ (d-1)2^{d-1} & d \geq 2. \end{cases}$$

(7) The total number of hull-honest hyperplanes in $\mathcal{H}(C^d)$ is given by

$$h(d) = \frac{3^d(2d-3)}{6} + 2d + \frac{1}{2}.$$

We also derived an upper bound on the size of the normal vector v of an arbitrary hyperplane in $\mathcal{H}(C^d)$. Beside the theoretical interest, this bound proved useful in the experimental enumeration of $\mathcal{H}(C^d)$. We assumed v to be an integer vector and as short as possible. By symmetry of the d-cube, we can restrict attention to hyperplanes which pass through the origin.

(8) Let $H = (v, 0) \in \mathcal{H}(C^d)$. The entries v_i of v are bounded by

$$|v_i| \leq 2\left(\frac{d}{4}\right)^{\frac{d}{2}} \text{ for } i = 1, \ldots, d.$$

in European Journal of Combinatorics

[2] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw Hill, New York, 1968

[3] Harald S. M. Coxeter, *Regular Polytopes*, Dover Publications, New York, 1963/73

[4] E.N. Gilbert, Gray Codes and Paths on the n-Cube, Bell Systems Tech. J. 37, pp 1-12, 1958

[5] Ron L. Graham and H.O. Pollak, On the Addressing Problem for Loop Switching, Bell Systems Tech. J. 50, pp 2495-2519, 1971

[6] Branko Grünbaum, *Convex Polytopes*, Interscience, New York.

[7] Jeff Lagarias and Peter Shor, manuscript, AT&T Bell Labs, Murray Hill, NJ

[8] H. Naumann, Beliebige konvexe Polytope als Schnitte und Projektionen höherdimensionaler Würfel, Simplices und Masspolytope, Mathematische Zeitschrift 65, pp 91-103, 1956.

[9] Patrick E. O'Neil, Hyperplane Cuts of an n-Cube, Discrete Mathematics 1, pp 193-195, 1971/72

[10] Michael E. Saks, Slicing the Hypercube, manuscript, Dept. of Math., Rutgers Univ., New Brunswick, NJ

[11] N.J.A. Sloane, A Short Course on Error Correcting Codes, CISM Courses and Lectures 188, Springer, Wien, New York, 1975

Monotone Boolean Formulæ for Isothetic Polyhedra

Robert Juan-Arinyo
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Av. Diagonal 647, 8a, E–08028 Barcelona

Abstract

We consider the problem of converting boundary representations of isothetic polyhedra into constructive solid geometry (CSG) representations. The CSG representations are based on the halfspaces supporting the faces of the polyhedron.

Introduction

Peterson [4] considered the problem of obtaining a CSG representation for non necessarily convex polyhedral solids such as extrusions and pyramids. These solids are simple in the sense that they can be described by a two-dimensional base plus a parameter giving the third dimension. Peterson proved that every polyhedron belonging to these two families admits a representation by a boolean formula based on the halfspaces supporting its faces. This boolean formula exhibits two important features: no term is complemented (it is monotone) and each supporting halfspace appears in the formula once and only once.

Dobkin *et al.* [1] proved that not all polyhedra have a CSG representation of the style given above. Dobkin *et al.* called this kind of boolean formula a *Peterson-style formula*. In this work two questions were left open: 1) Can the interior of a polyhedron with n faces be represented by a formula using $O(n)$ literals? 2) Is it possible to characterize polyhedra, other than extrusions and pyramids, that can be represented by a *Peterson-style formula*?

Juan-Arinyo considered the second open question in [2]. There, the *convex hull decomposable (CHD)* isothetic polyhedra were defined and it was proved that every polyhedron in this family has a Peterson-style formula. It is easy to see that the algorithm given in [2] is a simplification of the Alternating Sum of Volumes (ASV) Decomposition algorithm reported in [5] and that it trivially applies to every *convex hull decomposable* polyhedron. The main problem with both ASV and CHD decompositions is that they do not apply to *cyclic* polyhedra, [5, 2, 3].

In this work we extend the polyhedral domain for which a Peterson-style formula exists with those cyclic isothetic polyhedra such that for each cyclic deficiency it is possible to find out at least either a convex or a concave path of *pseudo maximal* edges that splits the cyclic deficiency set.

Extending the domain

A polyhedron is said to be *isothetic* if all of its faces are perpendicular to each other. The number of faces converging to a isothetic vertex can be either three, four or six. Henceforth we shall deal with isothetic polyhedra.

Any given isothetic polyhedron P always has at least six faces on its convex hull, $CH(P)$, and $CH(P)$ always splits the set of faces of P in several subsets. One set is made of the faces such that their supporting halfplanes belong to $CH(P)$. Faces in this set are not necessarily connected. Each of the other sets contains a maximal connected set of faces such that their supporting halfplanes do not belong to $CH(P)$. Each of these sets is called a *deficiency set* of P. Let $\{h_i, 1 \leq i \leq 6\}$ denote the supporting halfspaces in $CH(P)$ and let D_j denote the deficiencies of P. Furthermore, let $R(D_j)$ be the generalized halfspace defined in $\mathbf{E}^3$ by deficiency D_j. Then

$$\bigcap_i h_i \bigcap_j R(D_j)$$

is a boolean formula that properly represents polyhedron P. Obviously, if, for every j, $R(D_j)$ has a Peterson-style formula, then the formula above for P is also a Peterson-style formula.

If deficiency D_j is not cyclic it has a Peterson-style formula [2]. Hence, what we have to do is to build the Peterson-style formula for a cyclic deficiency D_j.

To extend the polyhedral domain for which a Peterson-style formula exists we first define the *pseudo maximality* property of edges in deficiency

D. Then we define an *edge path*, γ, on D as a set of ordered edges of D such that: 1) The first vertex of the first edge and the last vertex of the last edge are on $CH(D)$; 2) No vertex in the set of edges is shared by more than two edges; and 3) All the edges in the set are either convex or concave. If all the edges in γ are convex, the path is said to be convex and the path is concave when all the edges in it are concave. Now it is trivial to prove the following lemma.

Lemma. Let D be a deficiency set of faces and γ a pseudo maximal path on D. Let D_1 and D_2 be the two subsets of faces induced by γ in D and let $R(D_1)$ and $R(D_2)$ be the regions they define in $\mathbf{E}^3$. Then, the region $R(D)$ defined in $\mathbf{E}^3$ by D is properly represented by the CSG fromula $R(D_1) \cap R(D_2)$ when γ is convex and by $R(D_1) \cup R(D_2)$ when γ is concave.

This lemma allows us to apply the algorithm reported in [2] to each subset in D induced by the pseudo maximal path γ, and to compute a Peterson-style formula for deficiency D.

Extension of the domain depends strongly on the definition of *pseudo maximality*. We have tried two different definitions, each of them having pros and cons.

We will prove, by giving an example, that a Peterson-style formula does not exist for every isothetic cyclic polyhedron and we will give a rational that leads to a sufficient condition for the non-existence of this kind of formula.

References

[1] D. Dobkin and L. Guibas and J. Hersberger and J. Snoeyink. An Efficient Algorithm for Finding the CSG Representation of a Symple Polygon. *ACM Computer Graphics*, Vol. 22, No. 4, pages 31–40. August 1988.

[2] R. Juan. On Boundary to CSG and Extended Octrees to CSG conversions. In W. Strasser and H.-P. Seidel, editor, *Theory and Practice of Geometric Modeling*, pages 349–367, Springer-Verlag, 1989.

[3] Y.S. Kim and D.J. Wilde. Local Cause of Non-convergence in a Convex Decomposition Using Convex Hulls. In *The 1989 ASME Design Technical Conferences. Advances in Design Automation*, Montreal, Quebec, Canada, 1989.

[4] D.P. Peterson. Halfspace Representations of Extrusions, Solids of Revolution and Pyramids. Technical Report SAND84-0572, Sandia National Labs., Albuquerque, New Mexico, 1984.

[5] T. Woo. Feature Extraction by Volume Decomposition. In *Conference CAD/CAM Technology in Mechanical Engineering*, Cambridge, Mass., 1982.

From Spider Robots to Half-disk Robots

J-D. Boissonnat* O. Devillers* S. Lazard*

We study the problem of computing the set $\mathcal{F}$ of accessible and stable placements of a spider robot. The body of this robot is a single point and the legs are attached to the body. The maximal extension of each leg is R (accessibility constraint) and the body of the robot must lie inside the convex hull of the footholds (stability constraint). The environment consists of a set of n points $\{s_1, \ldots, s_n\}$ in the plane representing the authorized footholds. We present an efficient algorithm to compute $\mathcal{F}$, based on the following property: The spider robot at point P can set its legs in a stable way if and only if there doesn't exist any half disk of radius R centered at P containing no foothold.

Thus, we turn the computation of $\mathcal{F}$ into the computation of the free space $\mathcal{L}$ (defined in $\mathbb{R}^2 \times S^1$) of a half-disk robot moving under translation and rotation among n obstacles (the footholds). $\mathcal{F}$ is equal to $\mathcal{C}(p_{//\theta}(\mathcal{L}))$ where $p_{//\theta}$ is the orthogonal projection onto $\mathbb{R}^2$ defined in $\mathbb{R}^2 \times S^1$ and $\mathcal{C}$ denotes the complementary set in $\mathbb{R}^2$. We can express $\mathcal{L}$ in terms of Minkowski sums and we find that $\mathcal{L}$ is the complementary of the union of the n helicoidal volumes $\mathcal{H}_i$ (in $\mathbb{R}^2 \times S^1$). The intersection of $\mathcal{H}_i$ with the plane $\theta = \theta_0$ is a half disk of radius R centered at s_i with orientation θ_0 (see Figure 1). The following argument will permit us to avoid the computation of the union of volumes $\mathcal{H}_i$.

A point of the boundary $\delta(\mathcal{F})$ of $\mathcal{F}$ corresponds to a placement of the robot in a limit position either due to the stability constraint or the accessibility constraint. So $\delta(\mathcal{F})$ is composed of arcs of circles C_i (of radius R, centered at s_i) and staright edges which are portions of line segments $[s_i, s_j]$. We can calculate the circular arcs of $\delta(\mathcal{F})$ by computing for each i_0 the union $\mathcal{U}_i$ of the $\mathcal{H}_i \cap \mathcal{C}_{i_0}$ where $\mathcal{C}_{i_0}$ is the cylinder $C_{i_0} \times S^1$ of $\mathbb{R}^2 \times S^1$. We reduce the computation of each $\mathcal{U}_i$ to the construction of some lower and upper enveloppes of algebraic functions of a special type. We can compute each $\mathcal{U}_i$ in $O(k_{i_0} \log k_{i_0})$ time where k_{i_0} is the number of $\mathcal{H}_i$ intersecting $\mathcal{C}_{i_0}$ [2, 3]. Hence, we can compute all the circular arcs of $\delta(\mathcal{F})$ in $O(|\mathcal{A}| \log n)$ time, where $\mathcal{A}$ is the arrangement of the circles C_i of radius R centered at the s_i. Moreover we can compute within the same time bound the straight edges of $\delta(\mathcal{F})$.

The algorithm that we present computes $\mathcal{F}$ in $O(|\mathcal{A}| \log n)$ time and $O(|\mathcal{A}|\alpha(n))$ space where α is the pseudo-inverse of the Ackerman function. The geometric complexities of $\mathcal{A}$ and $\mathcal{F}$ are $O(n^2)$ and the bound is reached in the worst case. Compared to the algorithm described in [1], our algorithm computes $\mathcal{F}$ in a completely different way. The time complexity is the same but our method is easier to implement and can be extended to the case where the footholds are no longer points but polygons with the same complexities if n is the total number of polygons edges.

*INRIA, BP 93, 06902 Sophia Antipolis Cedex, France. E-mail : Sylvain.Lazard@inria.fr

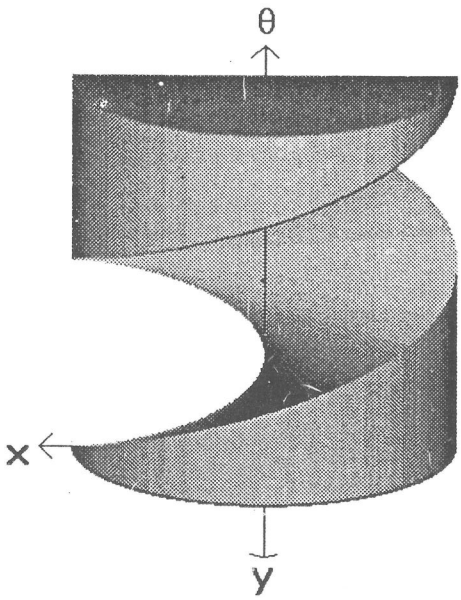


Figure 1: Helicoidal volume $\mathcal{H}_i$

1. J.-D. Boissonnat, O. Devillers, L. Donati, and F. P. Preparata. Stable placements of spider robots. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 242–250, 1992.

2. M. Dickerson and R. L. Drysdale. Fixed radius search problems for points and segments. *Inform. Process. Lett.*, 35:269–273, 1990.

3. J. Hershberger. Finding the upper envelope of n line segments in $o(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

On the Tolerance of some Geometric Structures *

(Extended abstract)

Manuel Abellanas †
Francisco Gómez ‡
Ferran Hurtado §
Pedro Ramos ¶

December 3, 1993

Let $S = \{p_1, \ldots, p_n\}$ be a set of points in the plane and let us consider:

a) The Euclidean Minimum Spanning Tree of S, $EMST(S)$;

b) The number h of points in the convex hull of S;

c) The property P_k: the first k points of S can be enclosed in a circle with the remaining $n - k$ points outside it;

d) A line l that bisects S.

These examples correspond respectively to a combinatorial structure, a parameter, a property and a geometric object *topologically* associated to the set of points: if S is in *general position* (the meaning of general position depends on the structure or property under consideration), we can move the points arbitrarily inside some neighbourhood (perhaps very small) and be certain that $EMST(S)$ and h do not change, P_k is still true and l is still a bisector.

The *tolerance* of $EMST(S)$ is defined as the supreme of $r > 0$ such that if each point p_i is moved arbitrarily but not more than r then the combinatorial structure $EMST(S)$ does not change. The definitions are analogous in situations b), c) and d) and have always the same "conservative" sense.

The tolerance is naturally related to the accuracy of the input of data since, if the tolerance is big, errors comparatively small in the input will be irrelevant. On the contrary,

*Partially supported by CICYT grant TIC 93-0747-C02
†Facultad de Informática (U.P. de Madrid) mavellanas@fi.upm.es
‡E.U. de Informática (U.P. de Madrid) fmartin@eui.upm.es
§Facultad de Informática (U.P. de Catalunya) hurtado@ma2.upc.es
¶Dpto. Matemática Aplicada (U.P. de Madrid) pramos@fi.upm.es

if the situation is like in Figure 1 (where the tolerance is small), even tiny errors in data can produce different results. However, the tolerance should not be confused with the concept of *algorithmic robustness* which studies how small roundoff errors can accumulate during different steps of an algorithm and produce a false final result. This is the approach in [4] where Guibas, Salesin and Stolfi define a concept very similar to tolerance but from the point of view of algorithmic robustness. The same can be said about the papers [5] by Guibas, Salesin and Stolfi and [6] by Milenkovic and Li where the authors propose some algorithms to compute an approximate convex hull taking into account roundoff errors. The main difference is that the tolerance measures the possible changes of a combinatorial structure *exactly* associated with a set of points.

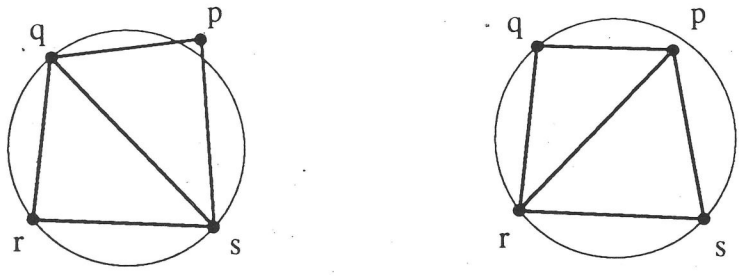


Fig. 1. A small perturbation of S can change DT(S).

The tolerance can also be interpreted as a measure of how far is a configuration of points from being degenerate with respect to some geometric or combinatorial structure since degenerate configurations have tolerance equal to zero (an arbitrarily small movement of the points can change the structure).

The concept of tolerance, because so natural, has already appeared implicitely in different settings. For example in [2] the authors consider polygons subjected to an assumption that we can describe now as a fixed lower bound for the tolerance of the simplicity of the polygon. In graph layout, it is sometimes interesting to redraw a graph with slight changes to make the picture clearer while preserving the *mental map* of the diagram; a possible way to do it is to preserve some geometric graph, as the Delaunay triangulation, of the points that correspond to the nodes [3], [6], [7]; so the tolerance would give here bounds for safe perturbations of the nodes.

Another framework closely related with tolerance is dynamic maintenance of geometric or combinatorial structures of moving points. If we have an upper bound of the velocity of the points, the tolerance gives an interval of time in which the structure does not change and then the passage between the initial and final configurations can be discretized.

If we divide the tolerance by a measure of the size of the set, say the diameter, we obtain a measure independent of the size that could be interpreted as a *quality coefficient*. Thereupon, we can ask ourselves, for a given structure, in what configuration of points the maximum value is reached.

If two or more structures give similar information about a configuration of points, their tolerance can be a reasonable criterion for a choice. A natural question is whether there exist any structure that has always a tolerance bigger than the others.

Finally, we can observe that in the examples a),b) and c) the geometrical object under consideration is uniquely defined, but when we have other possibilities as in d), where we consider a line that bisects a set of points it would be reasonable to try to select the object with maximum tolerance. In fact the concept of tolerance was introduced in [1] where the tolerance of the simplicity of a polygon is computed and the following problem is proposed: how should a set of points be polygonized in order that the resulting polygon has maximum tolerance? (Figure 2).

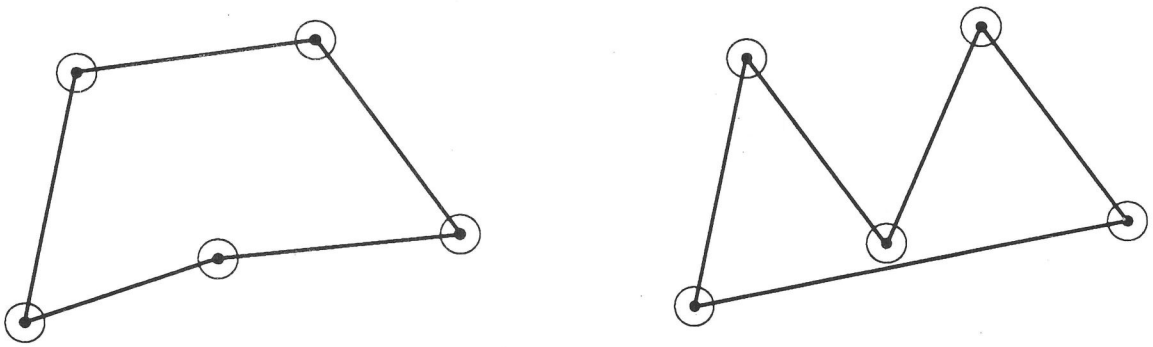


Fig. 2. Two different polygonizations of the same set with very different tolerance of their simplicity.

We have mentioned several directions of research that are suggested by the concept of tolerance, but certainly the basic question is how to compute the tolerance of a given structure of a set. Our objective is to exemplify this question with some important examples. The other aspects we mentioned earlier are developped in [9].

1 References

[1] M.Abellanas, J.García, G. Hernández, F.Hurtado, O.Serra, J.Urrutia: *Updating Polygonizations*, Computer Graphics Forum, vol.12, no.3 (1993), pp.143-152.

[2] E.M.Arkin,P.Belleville,J.S.B.Mitchell,D.Mount,K.Romanik, S.Salzberg,D.Souvaine: *Testing Simple Polygons* Proceedings 5th Canadian Conference on Computational Geometry (1993) pp. 387-392.

[3] P.Eades, W.Lai, K.Misue, K.Sugiyama: *Preserving the Mental Map of a Diagram* Research Report IIAS-RR-91-16E, International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories Ltd., 17-25, Shinkamata 1-chome, Ohta-ku, Tokyo 144, Japan, August 1991.

[4] L.Guibas, D.Salesin, J.Stolfi: *Constructing Strongly Convex Approximate Hulls with Inaccurate Primitives* Proceedings 1st Annual SIGAL International Symposium on Algorithms, LNCS vol 450, pp. 261-270.

[5] L.Guibas, D.Salesin, J.Stolfi: *Epsilon-Geometry: Building Robust Algorithms for Imprecise Computation* Proceedings 5th ACM Annual Symposium on Computational Geometry (1989) pp.208-217.

[6] K.A.Lyons: *Cluster Busting in Anchored Graph Drawing* Ph.D. Thesis (in preparation).

[7] K.A.Lyons, H.Meijer, D.Rappaport: *Properties of the Voronoi Diagram Cluster Buster* Proceedings 1993 CAS Conference (CASCON'93) Vol. II, A.Gawman, W.M. Getleman, E.Kidd, P.Larson, J.Slonim, Eds, IBM Canada Ltd. Laboratory Center for Advanced Studies and National Research Council Canada, Toronto, Ontario, Canada. October 24-28 1993, pp. 1148-1163.

[8] V.Milenkovic, Z.Li: *Constructing Strongly Convex Hulls Using Exact or Rouded Arithmetic* Proceedings 6th ACM Annual Symposium on Computational Geometry (1990) pp.235-243.

[9] P.Ramos: *Tolerancia de Estructuras Geométricas y Combinatorias* Ph.D. Thesis (in spanish), in preparation.

Controlling Guards

Gregorio Hernández-Peñalver
Facultad de Informática. Universidad Politécnica de Madrid
Campus de Montegancedo. Boadilla del Monte. 28660 Madrid (Spain)
e-mail: gregorio@fi.upm.es

Extended Abstract

1 Introduction

The original Art Gallery Problem raised by V. Klee asks how many guards are sufficient to watch every point inside a n-sided simple polygon. Chvatal showed that $\lfloor \frac{n}{3} \rfloor$ guards are always sufficient and sometimes necessary [1]. Many variations of this theorem have been explored and many results have been obtained, [2,4]. In 1987, Shermer introduces the concept of hidden guard set [5]. A hidden guard set is a guard set of points such that no two guards in the set are visible to each other. This corresponds to "independent dominating sets" in $PVG(P)$. Shermer showed that not every polygon admits a hidden guard set on its vertices and that, given a polygon P, the problem of determining whether such a hidden guard set exists for P is NP-complete.

In this paper we analyze another variation: every guard must be watched by another guard. We consider two kinds of guards on the vertices of a polygon: **watched guards**, such that each one is watched, at least, by another guard, and **connected guards**, such that $VG(S)$, the visibility graph of the set of guards, is connected. We prove that $g^W(n)$, the minimum number of watched guards necessary to watch any polygon of n vertices, is $\lfloor \frac{2n}{5} \rfloor$. If we consider connected guards, then $g^C(n)$, the minimum number of connected guards, is $\lfloor \frac{n}{2} \rfloor - 1$, being $\frac{n}{2} - 2$ when the polygon is an ortogonal polygon. We proof that all these bounds are tight.

2 The lower bounds

Figure 1 shows an example of a simple polygon that needs two watched guards for each five triangles in a triangulation of P. Therefore

$$g^W(n) \geq \lceil \frac{2t}{5} \rceil = \lfloor \frac{2n}{5} \rfloor$$

If we consider connected guards, the snake polygon shown in figure 2 requires $\lfloor \frac{n}{2} \rfloor - 1$ connected guards. For ortogonal polygons the lower bound is $\frac{n}{2} - 2$. Looking at the most simple periodic staircase (fig. 3), we check that $\frac{n}{2} - 2$ connected guards are necessary and sufficient to watch the polygon.

3 Upper bound for watched guards

The proof is by induction and it follows the main outlines of O'Rourke's proof for mobile guards [3]. Let P a polygon and T a triangulation graph for P. O'Rourke uses the identity between the number of combinatorial and geometric mobile guards necessary and sufficient to dominate and cover triangulation graphs and polygons, respectively.

Now the combinatorial counterpart of the watched guards are the vertex guards in T such that any two of them are linked by an arc of T. It is clear that if a triangulation graph of a polygon can be dominated by k combinatorial guards, then the polygon can be covered, i. e. watched, by k geometric watched guards placed at vertices. This implies that a proof of the sufficiency of $\lfloor \frac{2n}{5} \rfloor$ combinatorial watched guards in a triangulation graph establishes the sufficiency of the same number of geometric watched guards in a polygon with $n > 4$ vertices.

Following O'Rourke's proof we must consider the edge contraction of T and utilize the following lemmas:

Lemma 1(O'Rourke)
Let T a triangulation graph of a polygon P, and T' the graph resulting from an edge contraction of T. Then T' is a triangulation graph of some polygon P'.

Lemma 2
Suppose that $f(n)$ combinatorial watched guards are always sufficient to dominate any n-node triangulation graph. Then if T is an arbitrary triangulation graph of a polygon P with one vertex guard, Q, placed at any one of its n nodes, then an additional $f(n-1)$ watched guards are sufficient to dominate T. (But, perhaps the guard Q remains without any guard watching to it).

Then we establish the sufficiency of the bound $\lfloor \frac{2n}{5} \rfloor$ for small triangulation graphs and the existence of a diagonal that will allow us to get the induction step.

Lemma 3
(a) Every triangulation graph of a pentagon can be dominated by two combinatorial watched guards with one at any selected node.
(b) Every triangulation graph of a hexagon can be dominated by two combinatorial watched guards with one at one vertex of any selected edge.
(c) Every triangulation graph of a n-polygon with $7 \geq n \geq 11$ can be dominated by $\lfloor \frac{2n}{5} \rfloor$.

Lemma 4
If T is any triangulation graph of a polygon P, with $n \geq 12$ vertices then there exists a diagonal that cuts off exactly $6, 7, 8, 9$ or 10 edges.

With the preceding lemmas available, the induction proof is a simple enumeration of cases.

Theorem
Every triangulation graph T of a polygon with $n \geq 5$ vertices can be dominated by $\lfloor \frac{2n}{5} \rfloor$ combinatorial watched guards.

Proof.
Lemma 3 establishes the truth of the theorem for $5 \leq n \leq 11$. Assume now, that $n \geq 12$ and the induction hypothesis. Lemma 4 establishes that there is a diagonal that partitions T into two graphs T_1 and T_2, where T_1 contains k boundary edges with $6 \leq k \leq 10$. We must consider each value of k separately. Here we include only one case.

Case $k = 9$. (See figure 4)
The presence of any of the diagonals $(0,8),(0,7),(0,6),(1,9),(2,9),(3,9)$ would violate the minimality of k. So that the triangle L in T_1 that is bounded by d is either $(0,4,9)$ or $(0,5,9)$, which are equivalent cases. Suppose L is $(0,5,9)$. Form the graph T_0 by adjoining the polygon 056789 to T_2. The polygon 012345 has six edges, and so by Lemma 3(b), it can be dominated with two combinatorial watched guards, one of them placed at node 0 or 5. We choose node 5. T_0 has $n-4$ edges and the guard at node 5 permits the remainder of T_0 to be dominated by $f(n-4-1) = f(n-5)$ combinatorial watched guards, where $f(m)$ is the number of combinatorial watched guards that are always sufficient to dominate a triangulating graph of m nodes. By the induction hypothesis, $f(m) = \lfloor \frac{2m}{5} \rfloor$.

Therefore, $\lfloor \frac{2(n-5)}{5} \rfloor = \lfloor \frac{2n}{5} \rfloor - 2$ combinatorial watched guards suffice to dominate T_0. Together with the two allocated in the polygon 012345, we conclude that T is dominated by $\lfloor \frac{2n}{5} \rfloor$ combinatorial watched guards.

4 Upper bounds for connected guards

Let P a simple polygon with n vertices. Once P is trangulated, it is easy to see that we can place adequately guards at vertices of the polygon in such form that every guard watchs two triangles and $VG(S)$, the visibility graph of the set of guards, is connected. Therefore $g^C(n) = \lfloor \frac{n-2}{2} \rfloor$

In the ortogonal case, we consider a convex quadrilateralization of the polygon. Let be S the set of guards built placing a guard at every edge that share two convex quadrilaterals. It is easy to check that $VG(S)$ is connected and $card(S) = \frac{n}{2} - 2$. Therefore $g_O^C(n)$, the minimum number of connected guards necessary to watch any ortogonal polygon of n vertices, is $\frac{n}{2} - 2$.

References

[1] V. Chvátal. *A Combinatorial Theorem in Plane Geometry.* J. of Combinatorial Theory Series B 18, pp. 39-41, (1975).

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

[3] J. O'Rourke. *Galleries Need Fewer Mobile Guards: a Variation on Chvátal's Theorem.* Geometriae Dedicata 14, pp. 273-283, (1983).

[4] T. Shermer. *Recent Results in Art Galleries.* Proceedings of the IEEE (G.T. Toussaint, ed.), Sept. 1992.

[5] T.Shermer. *Hiding People in Polygons.* Computing 42, pp. 109-131, (1989).

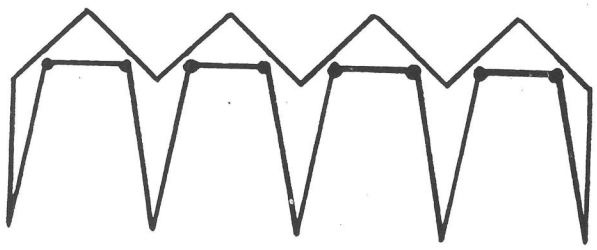


Fig. 1

Fig. 2

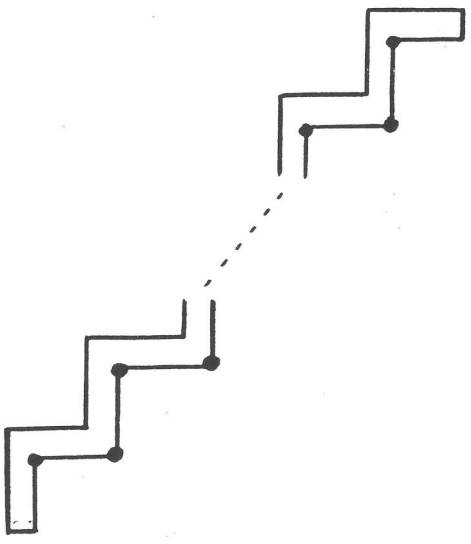


Fig. 3

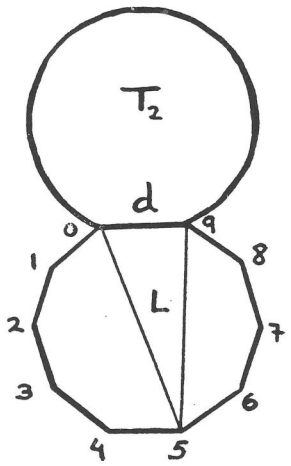


Fig. 4

Computing the Geodesic L_1-Diameter of a Simple Rectilinear Polygon in Parallel

Sven Schuierer

Abstract

Rectilinear paths and rectilinear obstacles play an important role in many applications: VLSI design is one of the most prominent examples. A natural metric to measure the distance between two points in such a setting is the L_1-metric since it corresponds to the length of a shortest rectilinear path. In this talk we address the problem of finding the maximal L_1-distance, i.e., the length of a shortest rectilinear path, between two points inside a polygon, also known as the *diameter problem*. A closely related problem is that of locating the set of points inside a simple rectilinear polygon whose maximal L_1-distance to any other point in the polygon is minimized. This set is known as the *center* of the polygon.

Center and diameter problems have been considered in various other contexts and a number of results have been obtained. Pollack *et al.* [PSR89] give an $O(n \log n)$ algorithm to compute the geodesic center of a polygon if the Euclidean metric is used. It is not known up to date if the time complexity of this algorithm is optimal or whether it can be improved upon. The Euclidean diameter of a simple polygon can be computed in linear time by a result of Hershberger and Suri [HS93]. They employ an ingenious variant of matrix searching to compute the Euclidean diameter. Unfortunately, it is not known how to parallelize matrix searching efficiently. Hence, different techniques have to be used in a parallel setting.

In this talk we consider the L_1-metric inside a simple rectilinear polygon P. We give an $O(\log n)$ time algorithm for a $CREW$-PRAM with $O(n/\log n)$ processors to compute the L_1-diameter $D(P)$ of a triangulated simple rectilinear polygon P as well as a pair of vertices that span $D(P)$. With the help of this vertex pair the center of P can also be computed within the same time bound.

References

[HS93] J.Hershberger and S. Suri. *Matrix Searching with the Shortest Path Metric.* 25th ACM Symposium on Theory of Computing.

[PSR89] R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete and Computational Geometry*, 4(6):611–626, 1989.

Towards a Fast Solution Method for the General Robot Motion Planning Problem using a Manhattan-like Distance Function on a Non-Uniform Grid in Configuration Space.

Carl Van Geem
RISC-Linz
Research Institute for Symbolic Computation
Joh. Kepler University
A-4040 Linz, Austria
Internet: Carl.Van.Geem@risc.uni-linz.ac.at

December 14, 1993

1 Introduction

The generalized Piano Mover's Problem can be extended in several ways, e.g. by taking in account kinematics and dynamic constraints, by allowing obstacles to move slightly, by asking for a path that can be seen as a good pay-off between safest and shortest path, by planning with uncertainty,... All these extensions are usually treated separately from each other in the literature. We will consider a combination of some of these extensions and we will investigate a direction of research that seems to be suitable for such combination. The approach plans in configuration space and is related to the potential field approach. The potential function however shall be replaced by an L^1-distance-like function on a non-homogeneous higher-dimensional grid. This distance function shall be computed with a wavefront expansion algorithm. The approach seems to suit in a natural way to massively parallel implementation.

2 Problem Specification

A formal specification of the three-dimensional generalized piano mover's problem $(O, P, p_{start}, p_{goal})$ can be given as follows:

- the obstacle set O consists of a finite set of (rational) polyhedra $O_1, ..., O_{n_1}$,
- the object P to be moved, consists of a finite set of (rational) polyhedra $P_1, ..., P_{n_2}$, which are freely linked at distinguished linkage vertices $v_1, ..., v_{n_3}$,
- p_{start} and p_{goal} are distinguished initial and final rational positions of P.

Motion planning in its most basic form, can be defined as finding a collision-free path for an object, or a collection of objects, among stationary objects. The objective of Robot Motion Planning is to automatically guide around a robot in an environment filled with obstacles and to choose forces to be applied when assembling objects. This basic form of the problem can be extended, including planning with uncertainty, allowing obstacles to move, taking in account kinematic constraints and dynamics,...

3 Solving the problem ...

In essence there are two computational approaches to solve this problem: the Connectivity Representation approach and the Potential Field approach.

The *Connectivity Representation* approach consists of precomputing the connectivity of the space of all positions that are safe for the robot in the form of a non-directed graph, the so called connectivity graph, and then searching in this graph for a path.

The *Potential Field* approach makes the configuration space (i.e. the set of all possible configurations, whereas a configuration is a specification of the position of the robot relative to some fixed coordinate system) discrete, by turning it into a fine regular grid, and searches this grid for a free path. The search is then guided by a heuristic that always can be seen as a potential field.

4 Potential Field Approach

In the potential field approach, in the configuration space (Cspace) the robot, represented as a point, is attracted from start position to the goal position by an attractive potential field, whereas the obstacles are modelled as repulsive potentials. A path is found by computing the gradients of the total potential and following the negative trend from the well towards the minimum point. Local minima are causing some inefficiency or failure of this approach. One can try to define a function without any local minima (or with very few) or one

can find ways to escape out of the local minima when the robot gets trapped in them. When this approach is used as a heuristic, usually one cannot guarantee that a path will be found always if there exists one (since such a path might have to move in a direction opposite to the one 'downwards').

The potential field approach leads in a natural way to massively parallel implementation, using some kind of a grid to model the Cspace.

4.1 L^1-distance

Another function that can be classified with the potential fields, uses the L^1-distance which can be computed with a so called wavefront technique. In fact, the Cspace is modeled as a uniform grid, where some cells are 'filled' by the obstacle and where the 'empty' cells represent a free position in Cspace. At initialisation time, the 'filled' cells are labeled with '1' and the others with '0'. It is assumed that q_{start} and q_{goal} both are in an 'empty' cell. The navigation function used here is the L^1 distance, also called Manhattan distance, to the cell with goal position in it. This distance function U can easily be computed by a so called 'wavefront expansion' technique. At first, U is set to 0 at q_{goal}. In the next step, in all neighboring empty cells U is set to 1. Then recursively all empty neighbors of cells with $U = d$, except for those which already have a value, will get value $d+1$ for U. When q_{start} is reached, a path can be planned by following the decent of distance.

5 Direction for Further Investigation

On the one hand, the approach with the wavefront algorithm and the Manhattan distance is a complete solution (i.e. always finds a path if there is one and reports failure otherwise) to the Motion Planning Problem and a variant of the potential field approach, but seems to have the drawback of the size of the grid, since it grows exponentially with the dimension of the Cspace.

On the other hand, potential fields seem to be useful for planning in higher-dimensional Cspaces and also seem to react on slightly moving obstacles by a small perturbation which stabilizes after a short time.

If we want to use some grid in order to model a Cspace of dimension m bigger than 2 or 3 (e.g. up to 31), we would like to reduce the size. The 2^m-tree decomposition (for $m = 2$ it is also called a quadtree, for $m = 3$ an octree) is a well-known representation for the Cspace, and has the advantage that it is a finer grid nearby obstacles and a rougher grid where there are no obstacles around. It also needs less cells for representing the Cspace. When a Manhattan-like distance is 'filled in' in the grid-cells of the quadtree, one easily sees that, when the path along the cells is constructed in the straightforward analogous way, this path preferes to develop using the 'big' cells which are further away from the obstacles. One could make the observation that the behaviour of

the Manhattan-like distance near obstacles is similar to the velocity of water particles in a river, where the particles close to the river bank move slower than the particles in the middle of the river.

The path generated by the quadtree and the Manhattan-like distance is still fast, since it has similar properties as the path generated by the wavefront approach described above.

Since the representation is done by a non-uniform grid, the size of the cells along the path gives us some information about the allowed error during the execution of the path. So also uncertainty seems to be incorporated in a natural way in this proposed approach.

A new efficient Method to Represent and Process Proximity and Similarity in Sets of Complex Objects

Hartmut Noltemeier

University of Würzburg,

Deparment of Math. and Computer Science.

97074 Wuerzburg, Germany

A B S T R A C T

This paper centers around the representation of sets (of complex objects) including their similarity and/or proximity properties. The goal is a compact representation as well as the support of various kinds of operations. We shortly review bisector trees, Voronoi trees and introduce monotone bisector trees, showing some major advantages of this class of trees. Variants of this class (MB*T) are efficient tools f.e. for the compact representation and efficient partitioning of complex scenes of geometric objects, and we are showing additionally that by this single data structure a lot of operations can be supported very well. We report on extensive experimental results and point to some topics of futher interest.

Technological Conditions in Geometric Algorithms: A Case Study for the Nesting Problem

Sabine Stifter
RISC-Linz
(Research Institute for Symbolic Computation)
Johannes Kepler University
A-4040 Linz, Austria

Many, basically geometric, problems that arise in applications are not purely geometrical since there are also some technological conditions that have to be considered. These technological conditions are often reflected in strategies a human would follow when solving the task by hand.

A typical example is the "nesting problem". The following is a very informal description of the problem, as it may be stated by an engineer working on flame cutting:

Given: a set of shapes with certain properties,
a set of sheets,
restrictions,
optimality criteria.

Find: an arrangement of the shapes on (a subset of) the sheets, such that the restrictions are satisfied and the arrangement is (close to) optimal with respect to the optimality criteria.

Examples of properties of shapes:

- shapes can be rotated arbitrarily,
- shapes can be reflected on some axes,
- shapes can be rotatated by π only.

Examples of restrictions:

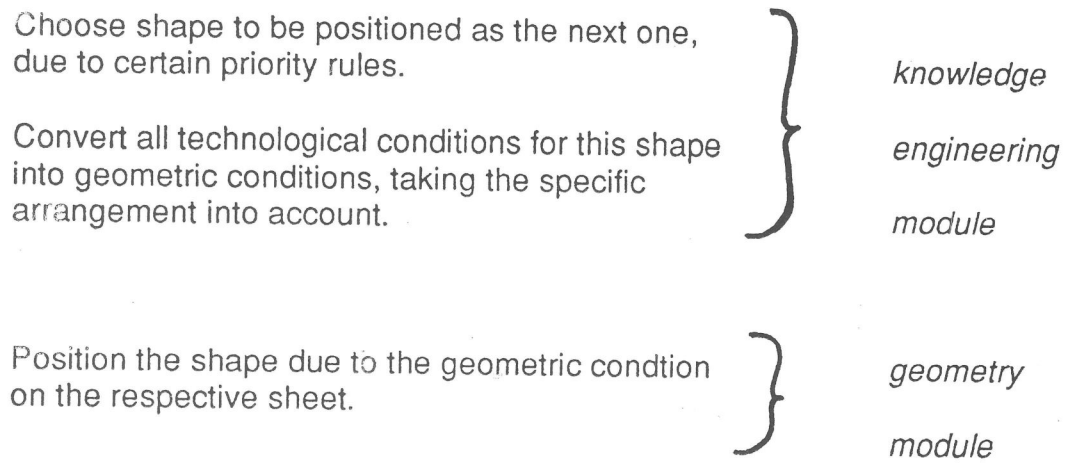
- the distance between two shapes has to be larger than a certain minimal distance,
- each shape needs a region where cutting can start.

Examples of optimality criteria:

- big shapes should be more at the boundary of the sheets,
- waste (area that is not covered by a shape) should be minimal,
- arrangements should (if possible) be identiacal on several sheets,
- not too many small shapes should be close together.

Expressions like "big", "small", "not too many" look purely geometrical. However, "not too many small shapes should be close together" can be specified by "the region should not be heated too much when cutting these shapes". So actually "small" and "too many" are closely related with each other and depend on the actual cutting technique. So "small" is not an intrinsic property of a single shape, but is a property that depends on the position of the shape on the sheet (and on the shapes that are positioned closely to it), and also on the cutting technique used. So in fact it is a "technological condition" that is specified above very informally. How can such technological conditions be handled? One way is to convert them to geometric condtions by using the technological knowledge or information. This conversion can be done by the engineer and be used as an input, i.e. as additional properties of the respective shapes. A second possibility is to use a two fold approach: the first level is some kind of knowledge engineering system for converting the technological information on-line into geometric information for a specific shape and a specific arrangement already reached on the sheet. The second level is a purely geometric algorithm that solves the geometric positioning problem.

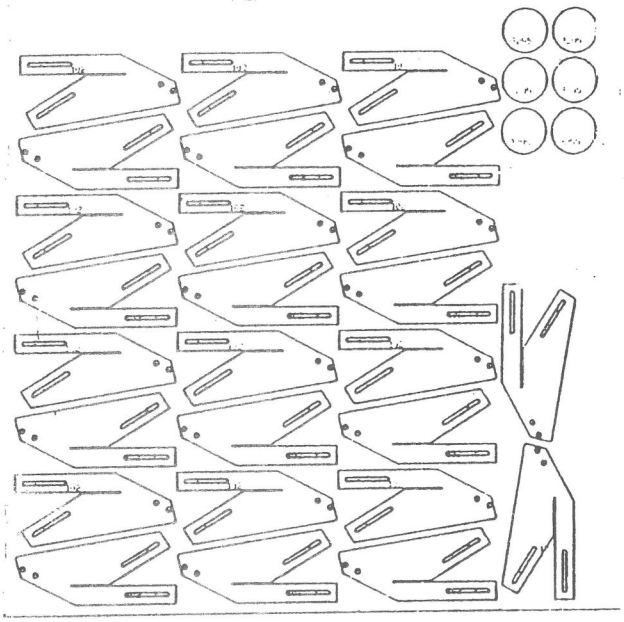
This two fold approach has the following software technological structure:



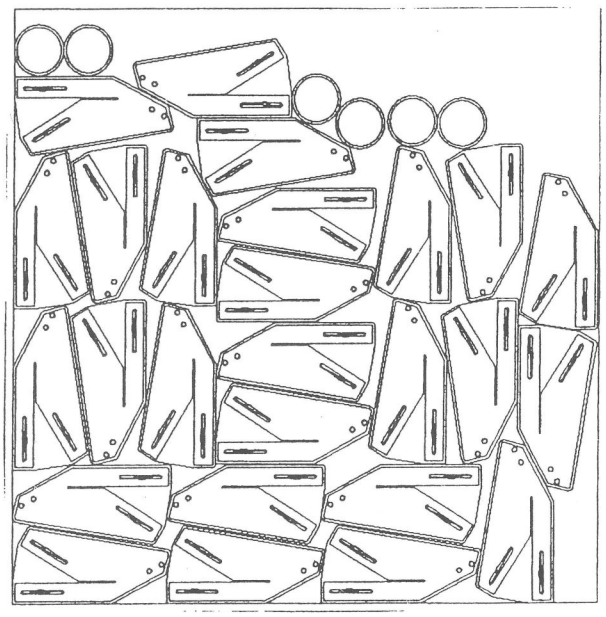
We realized this approach for the nesting problem in cooperation with an industrial partner. The specificl way of incorporating technological information into the geometric algorithm makes the system quite competitive with other systems already on the market.

We developed the geometry module based on a configuration space technique. The configuration space algorithm keeps especially track of all possible "free" position for further shapes. In order to keep computing times small, an approximative grid based approach is followed. So the basic strategy followed in the geometry module is to position each shape "as close as possible" to the arrangement already cosntructed. Where "as close as possible" is another technological condition that has to be translated to geometry.

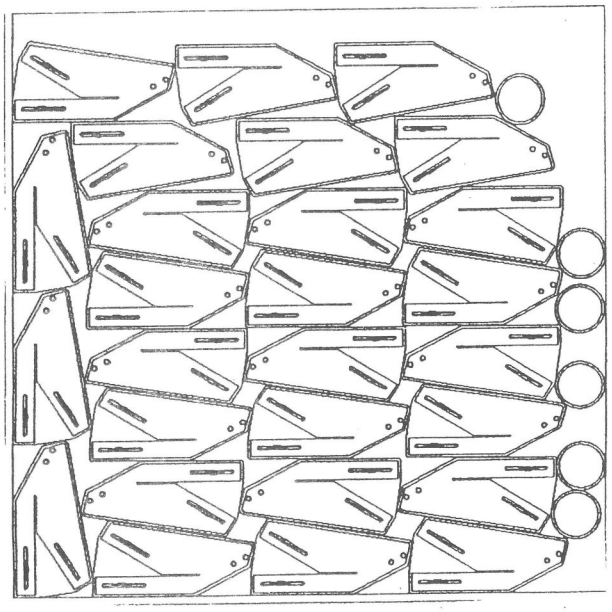
For the realization of the knowledge engineering module, the "translation" of human expertise for interpreting expressions like "big", "small", "close together" and also "as close as possible" into geometric conditions was the most challenging task. The following example shows three different results for three different geometric interpretations of "as close as possible" used by the geometry module for positioning a single shape. The first strategy is to interpret it as "as much to the left as possible", the second strategy is to interpret it as "minimize the area to the left and below", the third strategy considers a relaxed computation of the area.



Strategy 1:
"as much to the left as".



Strategy 2:
"minimize area to the left and below".



Strategy 3:
"minimize the area to the left and below", relaxed area computation.

Matching Shapes with a Reference Point
(Extended Abstract)

Helmut Alt*
Freie Universität Berlin
e-mail: alt@inf.fu-berlin.de

Oswin Aichholzer†
Technische Universität Graz

Günter Rote
Technische Universität Graz

1 Introduction

This paper is motivated by a problem that is typical in application areas such as computer vision or pattern recognition, namely, given two figures A, B, to determine how much they "resemble each other". Here, a "figure" will be a union of finitely many points and line segments in $\mathbf{R^2}$ or triangles in $\mathbf{R^3}$. Note that sets of curves in $\mathbf{R^2}$ and $\mathbf{R^3}$ or surfaces in $\mathbf{R^3}$ can be approximated arbitrarily closely by these objects. As a measure for "resemblance" we will use the *Hausdorff-metric* δ_H, which is a somehow natural distance measure and gives reasonable results in practice.

We assume that A and B are not fixed but can be moved by a *translation*, by a *rigid motion* (translation and rotation) or even transformed by a *similarity* (scaling and rigid motion) in order to match them as well as possible and then determine the minimal Hausdorff-distance. This problem of finding an optimal matching has been considered for the two-dimensional case in several previous articles (e.g. [AST, CGHKKK]). Those algorithms however, use sophisticated and powerful tools like parametric search and therefore do not seem to be applicable in practice.

Here, we follow a different approach which was already used in [ABB]. We do not try to find an *optimal* solution but an *approximation* to the optimal one by simpler algorithms. More precisely, if the optimal matching yields Hausdorff-distance δ our algorithms will find a matching T such that $\delta_H(A, T(B)) \leq a\delta$ for some constant $a > 1$. We call such a solution a *pseudooptimal matching* with *loss factor* a.

2 Reference Point Methods

A *reference point*, is defined for arbitrary dimension d as follows:

*This research was supported by the ESPRIT Basic Research Action Program No. 7141, Project ALCOM II.

†That research was supported by the Jubiläumsfond der Oesterreichischen Nationalbank.

Definition 1 *Let C^d denote the set of compact subsets of $\mathbf{R^d}$, and let $\mathcal{T}$ be a set of transformations on $\mathbf{R^d}$. A mapping $s\colon C^d \to \mathbf{R^d}$ is called a* reference point *with respect to $\mathcal{T}$ iff*

(a) s is equivariant *with respect to $\mathcal{T}$, i.e., for all $A, B \in C^d$ and $T \in \mathcal{T}$ we have*

$$s(T(A)) = T(s(A))$$

and

(b) there exists some constant $c \geq 0$ such that if for all $A, B \in C^d$,

$$\|s(A) - s(B)\| \leq c \cdot \delta_H(A, B).$$

In other words, s is a Lipschitz-continuous mapping between the metric spaces (C^d, δ_H) and $(\mathbf{R^d}, \|\cdot\|)$ with Lipschitz constant c. We call c the quality *of the refence point s.*

Based on the existence of a reference point for $\mathcal{T}$ we obtain the following algorithms for pseudooptimal matchings where $\mathcal{T}$ is the set of translations, rigid motions, and similarity transformations, respectively:

In the case of **translations** just compute $s(A)$ and $s(B)$, translate B by $s(A) - s(B)$, and output this matching as the pseudooptimal solution.

For **rigid motions** match $s(A)$ and $s(B)$ as before. Then rotate the image of B around $s(A)$ until the best matching under these rotations is found.

For **similarity transformations** match $s(A)$ and $s(B)$. Then determine the diameters $d(A)$ and $d(B)$ and scale the image of B by $\alpha := d(A)/d(B)$ around the center $s(A)$. Find an optimal matching under rotations as before.

These algorithms are simpler than the ones for finding the optimal solutions, since after Step 1 the matchings are restricted to ones leaving the reference point invariant. In d dimensions this eliminates d degrees of freedom. The qualities of these algorithms are as follows:

Theorem 2 *Suppose that a reference point of quality c for the sets of transformations $\mathcal{T}$ in the algorithms above exists. In the case of similarity transformations also assume that $s(A)$ always lies within the convex hull $conv(A)$. Then pseudooptimal matchings are found for translations and rigid motions with loss factor $a = c + 1$, for similarity transformations with loss factor $a = c + 3$.*

3 The Steiner Point

It follows from results in [ABB] that for reference point methods we can replace figures by their convex hulls, so we can without loss of generality restrict our attention to *convex* figures.

Our candidate for a reference point is the so-called *Steiner point*, which has been investigated intensively in the field of convex geometry [G].

Definition 3 *We denote by B^d the d-dimensional unit ball and by S^{d-1} its boundary, the $(n-1)$-dimensional unit sphere in $\mathbf{R^d}$.*

Let A be a convex body (convex and compact subset) in $\mathbf{R^d}$. The support function *$h_A\colon \mathbf{R^d} \to \mathbf{R}$ of A is given by $h_A(\boldsymbol{u}) = \max_{\boldsymbol{a}\in A} \langle \boldsymbol{a}, \boldsymbol{u}\rangle$ (see Figure 1).*

The Steiner point *$s(A)$ of A is defined as*

$$s(A) = \frac{d}{Vol(S^{d-1})} \int_{S^{d-1}} h_A(\boldsymbol{u})\boldsymbol{u}\, d\omega(\boldsymbol{u})$$

where $d\omega(\boldsymbol{u})$ is the surface element of S^{d-1}.

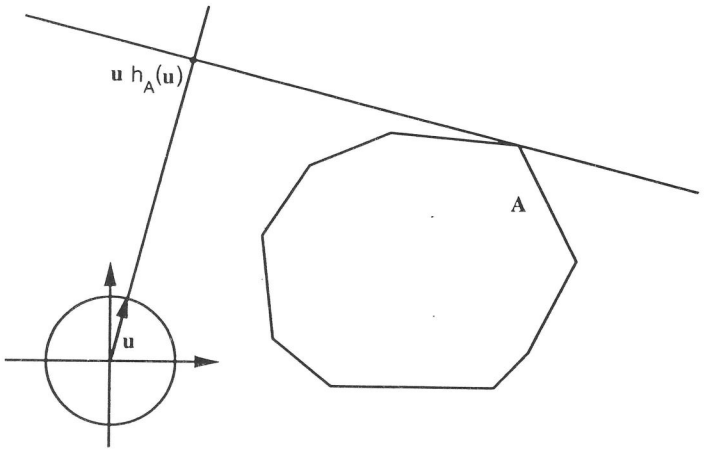


Figure 1: The support function $h_A(\mathbf{u})$ of a convex body A.

For the Steiner point we obtain

Theorem 4 *The Steiner point is a reference point for similarity transformations in arbitrary dimension d. For $d = 2$, its quality is $4/\pi$, for $d = 3$ it is $3/2$, for arbitrary d it is less than $\sqrt{2/\pi} \cdot \sqrt{d+1}$.*

It is well-known [G] that the Steiner point is easy to compute for convex polytopes. In fact, it is the weighted sum of the vertices, where the weight of vertex v is that fraction of the surface of the unit sphere that lies between the unit vectors normal to the hyperplanes meeting at v (the normalized exterior angle at v).

Combining the previous theorems and using suitable matching algorithms (see [ABB]) we get:

Theorem 5 *Let A and B be sets of n and m line segments in $d = 2$ dimensions or n and m triangles in $d = 3$ dimensions. Then pseudooptimal matchings can be found for A and B applying the corresponding algorithms of Section 2 as indicated in the following table, where $H(n, m)$ denotes the time to compute the Hausdorff-distance in 3 dimensions.*

$\mathcal{T}$	running time	loss factor
translations		
$d = 2$	$O((n+m)\log(n+m))$	$4/\pi + 1$
$d = 3$	$O(H(n,m))$	2.5
rigid motions		
$d = 2$	$O(nm\log(nm)\log^*(nm))$	$4/\pi + 1$
$d = 3$	$O((nm)^3 H(n,m))$	2.5
similarities		
$d = 2$	$O(nm\log(nm)\log^*(nm))$	$4/\pi + 3$
$d = 3$	$O((nm)^3 H(n,m))$	4.5

Notice that an upper bound of $O((n^2m + nm^2)log^3(nm))$ for $H(n,m)$ is known [AG].

4 Lower Bounds

For the case of translations we can show that the quality c of any reference point in two (or higher) dimensions cannot be better (i.e., smaller) than $\sqrt{4/3}$.

More general, for translations in d dimensions we obtain a lower bound of $\sqrt{2d/(d+1)}$ on the quality.

Comparing $\sqrt{4/3} \approx 1.155$ with the upper bound of $4/\pi \approx 1.27$ for the Steiner point, we note that the gap is quite small. We are quite sure that the lower bound can be improved, and we believe that the Steiner point is *the* optimal reference point with the best quality guarantee.

References

[ABB] H. Alt, B. Behrends, J. Blömer, Approximate matching of polygonal shapes, *Proceedings 7th Annual Symposium on Computational Geometry*, 1991, pp. 186–193.

[AG] H. Alt, M. Godau, Computing the Hausdorff-distance in Higher Dimensions, to appear.

[AST] P. J. Agarwal, M. Sharir, S. Toledo, Applications of parametric searching in geometric optimization, *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 1992, pp. 72–82.

[CGHKKK] P. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, D. Kravets, Geometric pattern matching under Euclidean motion, *Proc. 5th Canadian Conference on Computational Geometry*, 1993, pp. 151–156.

[G] B. Grünbaum, *Convex Polytopes*, Wiley & Sons, 1967.

Binary Space Partition for Sets Of Cubes *

Mark de Berg Marko M. de Groot

Abstract

In this paper we present a scheme to construct in time $O(n \log^3 n)$ a binary space partition of size $O(n \log^2 n)$ for a set of n non-intersecting axis-parallel cubes in 3-space.

1 Introduction

Problems where the input is a set of objects in the plane or in space are often solved by partitioning the space into subspaces, and then solving the problem on the subspaces recursively. A natural way to perform the partitioning is to make a linear cut of the space, that is, to split the space (and possibly some of the objects) with a hyperplane. The splitting process is repeated for each of the half-spaces with the corresponding sets of (fragments of) objects. This continues until there is at most one fragment of an object left in each of the subspaces. Such a partitioning scheme is called a *binary space partition*, or *bsp* for short. A binary space partition is naturally modeled as a tree structure: a *binary space partition tree*, or *bsp tree*. The nodes of the bsp tree store the splitting hyperplanes; the leaves correspond to the cells (subspaces) in the final partitioning and store (the fragment of) the object that is left in that cell. The formal definition of a bsp tree is as follows:

Definition 1.1 *A* binary space partition tree *(*bsp tree*) for a set S of objects in d-dimensional space is a binary tree $\mathcal{T}$ with the following properties:*

- *If $|S| \leq 1$ then $\mathcal{T}$ is a leaf; the set S is stored explicitly at this leaf.*
- *If $|S| > 1$ then the root ν of $\mathcal{T}$ stores a hyperplane h_ν, together with the set $S(\nu)$ of objects that are contained in h_ν. The left child of ν is the root of a BSP tree $\mathcal{T}^+$ for the set $h_\nu^+ \cap S = \{h_\nu^+ \cap s : s \in S\}$, where h_ν^+ is the region above h_ν. The right child of ν is the root of a BSP tree $\mathcal{T}^-$ for the set $h_\nu^- \cap S$, where h_ν^- is the region below h_ν.*

*Department of Computer Science,Utrecht University,P.O. Box 80.089,3508 TB Utrecht,The Netherlands. Research was supported by the ESPRIT Basic Research Actions of the EC under contract No. 7141 (project ALCOM II: *Algorithms and Complexity*). The first and third author were also supported by the Dutch Organization for Scientific Research (NWO).

Binary space partition trees are popular in many application areas. In computer graphics, for example, bsp trees are used for efficient implementations of the painter's algorithm [4]. In this algorithm one tries to "paint" the objects in a back-to-front order onto the screen. Thus objects in the front are painted on top of objects in the back, resulting in a correct view of the scene. Note that a depth order does not always exist, since there can be cyclic overlap among the objects. When the objects are stored in a bsp tree, however, then a back-to-front order for the object fragments in the tree can easily be obtained for any given viewing direction by traversal of the tree. Other uses of bsp trees in computer graphics include shadow generation [2]. In geometric modelling bsp trees have been used for the implementation of set operations on polyhedra [5, 8] and in robotics for (approximate) cell decomposition methods [1].

Obviously, the efficiency of algorithms that are based on binary space partitions depends strongly on the *size* of the bsp that is used, that is, on the number of cells in the final partitioning. Hence, it is important to choose the splitting hyperplanes in such a way that the fragmentation is kept as low as possible.

Several results in this direction have been obtained by Paterson and Yao [6, 7]. They proved that for any set of n line segments[1] in the plane there exists a bsp of size $O(n \log n)$; when the segments are orthogonal then there exists a bsp of $O(n)$ size. In both cases the bsp can be computed in $O(n \log n)$ time. For orthogonal objects the same result (with a slightly better constant for the combinatorial bound) was achieved by d'Amore and Franciosa [3]. Paterson and Yao have conjectured that any set of segments in the plane admits a bsp of linear size, but until now this conjecture is still open. Paterson and Yao also proved bounds on bsp trees in higher dimensions: they have shown that any set of $(d-1)$-simplices in d-space admits a bsp of size $O(n^{d-1})$, and any set of orthogonal rectangles admits a bsp of size $O(n^{d/(d-1)})$. In three-dimensional space they have given lower bound constructions which match their upper bounds, namely $\Omega(n^2)$ for the general case and $\Omega(n\sqrt{n})$ for the axis-parallel case.

Constructing a binary space partition of a set of axis-parallel cubes, i.e. cubes of which the edges are axis-parallel, in 3-space can be done using the scheme presented by Pater-

[1] Here and in the sequel we assume that the input objects are disjoint, since otherwise no bsp exists where each cell contains at most one object.

son and Yao [6], who present a scheme to construct in time $O(n\sqrt{n})$ a binary space partition of size $O(n\sqrt{n})$ for a set of n axis-parallel rectangles in 3-space. Thus the scheme of Paterson and Yao applied to the set of $6n$ facets of n axis-parallel cubes results in a binary space partition of size $O(n\sqrt{n})$, constructed in time $O(n\sqrt{n})$. In this paper we present a scheme to construct in time $O(n \log^3 n)$ a binary space partition of size $O(n \log^2 n)$ for a set of n axis-parallel cubes in 3-space.

2 Cubes in 3-space

We will now address the problem of finding a binary space partition for a set of n axis-parallel cubes in 3-space. Our method consists of two stages. In the first stage we construct an orthogonal subdivision such that all vertices of the cubes are contained in the partition planes chosen. The orthogonal subdivision is a balanced subdivision. In the second stage we further refine this orthogonal subdivision, such that a true bsp of the input cubes results.

The first stage. We recursively divide the space in two using planes that are orthogonal to one of the coordinate axes. Using this type of planes to subdivide the space, we construct only rectangular subspaces, which are called *voxels*. If the edges of a voxel that are parallel to the x_i-axis ($i \in \{1, 2, 3\}$) are longer then the edges parallel to the other two axes, then we call the x_i-axis the *main-axis* of the voxel. Thus the main-axis defines the dimension of the voxel that has the largest extent.

We choose the partition planes to split the interior of a single voxel, denoted by ν, such, that firstly we make all *free-cuts* [6] possible, i.e. partition planes containing a facet of a cube(s) without intersecting any other cube, and secondly, if no free-cuts are left, the partition planes meet the following criteria:

- the plane cuts the voxel orthogonal to the main-axis of the voxel.
- let n_ν denote the number of vertices intersecting ν, then the plane splits the set of n_ν vertices in two subsets such, that the number of vertices within both subsets is at most $n_\nu/2$ and the plane contains at least one vertex.

In case a voxel has more than one unbounded dimension or has two or more dimensions of equal extent (so more than one main-axis exist), we can choose one of those main-axes and split the interior of the voxel using a plane orthogonal to that axis.

Now we have defined the criteria that our partition planes must meet, we can describe a scheme to construct an orthogonal subdivision. We recursively divide the space into two subspaces, thereby dividing the set of vertices and thus the set of cubes within this space into two subsets. The recursive subdivision stops as soon as no vertex is left within the interior of a voxel. Note that the planes used to partition the space may intersect the cubes. After the first stage the following holds.

Lemma 2.1 *For each voxel there are at most four cubes that have an edge intersecting the two opposite sides of the voxel that are orthogonal to the main-axis of the voxel.*

Proof: Place four cubes at each of the four corners of the voxel, such that they have one edge intersecting the two opposite sides of the voxel that are orthogonal to the main-axis of the voxel. Because the cubes have equal extent in all dimensions and this extent at least equals the longest extent of the voxel no more such cubes can be added having a non-empty intersection with the interior of the voxel. □

To count the number of times each cube is cut in the first stage, we first bound the number of times each edge of a cube is cut.

Lemma 2.2 *The orthogonal subdivision applied to a set of n cubes results in $O(n \log n)$ edge fragments.*

Proof: We show that each such edge is only cut by $O(\log n)$ partition planes. As soon as an edge is cut for the first time, both segments will have a single endpoint in their respective subspace. Every successive cut of such segment by a partition plane results in a segment having one endpoint in the interior of its respective subspace and in a segment not having an endpoint in the interior of its subspace.

In the former case we can charge the number of cuts to the endpoint of the segment. The segment is cut at most once by each partition plane. Let $f_e(n)$ denote the number of times an edge is cut by the orthogonal subdivision in a space with n vertices. Knowing that each partition plane cuts the set of vertices in two sets each containing at most half of the vertices, we can bound the number of times the segment is cut by: $f_e(n) = 1 + f_e(n/2)$. Thus each segment is cut $O(\log n)$ times. Because an edge is divided into two such segments by the first partition plane each edge is cut $O(\log n)$ times.

In the latter case the segment intersects two opposite sides of the voxel. The segment will only be cut by partition planes when it intersects the two opposite sides orthogonal to the main-axis of the voxel. Lemma 2.1 states that at most four such segments can exist within each voxel. The recursive subdivision stops as soon as no vertex is left within the interior of a voxel. We represent the subdivision by a bsp-tree. As its leaves we find the voxels not containing a vertex in their interior. One level up from its leaves, however, we find

all the voxels containing at least one vertex in their interior. Because there are $8n$ vertices, our bsp-tree has $O(n)$ leaves. So there are only $O(n)$ segments intersecting two opposite sides orthogonal to the main-axis of a voxel. □

We bounded the number of times a cube is intersected in one of its edges, but a cube can also be cut without any of its edges being cut. So, we still have to count all intersections of the cube with a partition plane, such that the plane intersects the cube *without* intersecting any of its edges. This only can happen within a facet (fragment) of a cube. Such a facet is bounded by four edges, such that at least one of these edges is contained in a cube edge, i.e. an edge of a cube of the input set of cubes, and the other edges are the intersection of the facet of the cube with a partition plane. Note that each facet of which all four edges are intersections of the facet with a partition plane can be removed by a free cut, i.e. a partition plane containing the facet itself.

Lemma 2.3 *The orthogonal subdivision cuts each facet of a cube in at most* $O(\log^2 n)$ *fragments.*

Proof: We show that each facet (fragment) is cut by only $O(\log n)$ partition planes. As soon as a facet is cut into two fragments at least one fragment is created that has a bounding edge that is contained in a cube edge. We can charge the number of cuts to the edge that is contained in a cube edge. A partition plane intersects a facet at most once and does not intersect the bounding edge of the facet that is contained in a cube edge. The facet (fragment) is cut into at most two (new) fragments that have a bounding edge that is contained in a cube edge. Let $f_e(n)$ denote the number of times such new fragment is cut by the recursive subdivision in a space with n vertices. Knowing that each partition plane cuts the set of vertices in two sets each containing at most half of the vertices, we can bound the number of times the fragment is cut by: $f_e(n) = 1 + f_e(n/2)$. Thus each fragment is cut $O(\log n)$ times. Together with the proof of 2.2 this leads to the $O(\log^2 n)$ bound. □

Now we can bound the number of cube fragments generated by the recursive subdivision.

Lemma 2.4 *The partition scheme as described above generates an orthogonal subdivision of size* $O(n \log^2 n)$.

Proof: We proved that each facet is cut only $O(\log^2 n)$ times. Note that each cube has six facets. Because each partition plane intersects a cube if and only if the plane intersects a facet of the cube, each cube is cut into $O(\log^2 n)$ fragments. □

The second stage. Now voxels are left that are empty or that are intersected by cubes such that each cube has no vertex in the interior of the voxel. Observe that each cube intersecting a voxel intersects at least one pair of opposite sides of the voxel. We define two types of intersecting cubes, namely cubes that intersect the pair of opposite sides of a voxel that are orthogonal to the main-axis of that voxel and cubes that intersect one of the other pairs of opposite sides of the voxel. Lemma 2.1 shows that there are four cubes that intersect the two opposite sides of a voxel that are orthogonal to the main-axis of that voxel. These four cubes can be removed using a constant number of partition planes per cube, each plane containing (a fragment of) one side of the cube intersecting the voxel. Observe that each voxel created by these partition planes, has a main-axis that has the same orientation as the voxel they originate from.

Now within each voxel cubes remain that intersect at least one pair of opposite sides of the voxel that are parallel to the main axis of the voxel. We seperate the cubes intersecting one pair of opposite sides from the cubes intersecting the other pair of opposite sides by partition planes orthogonal to the main-axis of the voxel, and containing at least one facet (fragment) of a cube in their interior. Thus, within each voxel all intersecting cubes now at least intersect the same pair of opposite sides of the voxel. By projecting all cubes within a single voxel onto one of the pair of opposite sides of the voxel they intersect, partitioning the space within a voxel now is reduced to partitioning a set of rectangles in the plane. Observe that, because all cubes intersect the side of the voxel on which they are projected and the projection is orthogonal to that side of the voxel, no intersections are generated between the projections of the cubes. According to Paterson and Yao an $O(n)$ size binary space partition exists for a set of n orthogonal line segments, which can be constructed in time $O(n \log n)$ [7]. The partition planes for the set of cubes within the interior of the voxel are found by extending each partition line, that results from this reduced problem, to a partition plane orthogonal to the side of projection and containing the partition line.

Lemma 2.5 *For any set of* n *non-intersecting axis-parallel cubes in* 3*-space there exists an* $O(n \log^2 n)$ *size binary space partition.*

Proof: The proof of lemma 2.2 states that we have $O(n)$ voxels. The four cubes that intersect the two opposite sides of a voxel, denoted by ν, that are orthogonal to the main-axis of that voxel are removed, using a constant number k_ν of partition planes. So, to remove these cubes in total at most $O(n)$ partition planes are needed. Let n_ν denote the number of cubes intersecting ν. In total we generate $\sum_\nu k_\nu n_\nu = O(n \log^2 n)$ cube fragments.

Now we have to seperate within each voxel the cubes intersecting one pair of opposite sides of the voxel from the cubes

intersecting the other pair of opposite sides. Let $n_{\nu,x}$ denote the number of cubes intersecting the one pair and $n_{\nu,y}$ denote the number of cubes intersecting the other pair. Then at most $\sum_\nu n_{\nu,x} + n_{\nu,y} = O(n \log^2 n)$ partition planes orthogonal to the main-axis of the voxel are needed.

To partition the space within each voxel we use the scheme presented by Yao and Paterson [7]. They describe a scheme that constructs a binary space partition of size $O(n)$ for a set of n orthogonal non-intersecting line segments in the plane. Using an orthogonal projection, we reduce the original 3-dimensional problem to the construction of a binary space partition for non-intersecting line segments in the plane. Each projected cube is a rectangle that can be represented by the union of 4 line segments when using the scheme of Yao and Paterson. If we project n_ν cubes within a single voxel ν, then we can build a binary space partition of size $O(n_\nu)$. We extend each partition line found to a partition plane, orthogonal to the side of projection and containing the partition line. Thus, these cubes can be separated using only $O(n_\nu)$ partition planes. Again summing over all voxels generates an $\sum_\nu n_\nu = O(n \log^2 n)$ size subdivision. □

Theorem 2.6 *For any set of n non-intersecting axis-parallel cubes in 3-space there exists an $O(n \log^2 n)$ size binary space partition, that can be found in time $O(n \log^3 n)$.*

Proof: The size of the binary space partition follows from lemma 2.5.

The orthogonal subdivision as referred to by lemma 2.4 can be build in time $O(n \log^2 n)$. The following algorithm is adequate to achieve this bound [7]. We consider the configuration of edges of the set of input cubes. The configuration can be represented by six sorted lists, $E_{i,j}$ where $i \neq j$ and $i, j \in \{1, 2, 3\}$. The list $E_{i,j}$ contains all the edges of the configuration parallel to the x_i-axis, sorted in increasing order of x_j-coordinates. To determine the proper partition plane meeting the criteria previously mentioned, it is sufficient to step through the pair of lists $E_{j,i}$, $E_{k,i}$ in increasing x_i-order until the appropriate point is reached. To construct the representations for the configurations for both halfspaces defined by the partition plane chosen, a lineair pass through the representation of the previous configuration is sufficient. Thus the time for searching a partition plane is lineair in the size of the respective configuration. The total running time of this part of the algorithm is therefore bounded by the sum of the configuration sizes at all the nodes of the balanced partition tree, which is at most $O(\log n) \cdot O(n \log^2 n) = O(n \log^3 n)$.

The further refinement of the subdivision can be constructed in time $O(n \log^3 n)$. Then, projection of the cubes onto the sides of the voxels will take time $\sum_\nu n_\nu = O(n \log^2 n)$ in total. And according to Yao and Paterson [7], finding the partition planes will take time $\sum_\nu n_\nu \log n_\nu = O(n \log^3 n)$.

□

3 Conclusion

In this paper we presented a scheme that builds in time $O(n \log^3 n)$ an $O(n \log^2 n)$ sized binary space partition for a set of n non-intersecting and axis-parallel cubes in 3-space. The corresponding binary space partition tree is partly *balanced*: the first stage in our scheme generates a balanced binary orthogonal subdivision. The remainder of the scheme however does not take any balancing condition into account.

References

[1] C. Ballieux. Motion planning using binary space partition. Report inf/src/93-25, Utrecht University, 1993.

[2] N. Chin and S. Feiner. Near real time shadow generation using bsp trees. In *SIGGRAPH'90*, pages 99–106, 1990.

[3] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 1–5, 1992.

[4] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3):124–133, 1980.

[5] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH'90*, pages 115–124, 1990.

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

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

[8] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proc. SIGGRAPH'87*, pages 153–162, 1987.

List of Authors

Manuel Abellanas 64
Facultad de Informatica
Universidad Politécnica de Madrid
Boadilla Del Monte
E-28660 Madrid
<mavellanas@fi.upm.es>

Oswin Aichholzer 53, 81
Inst. for Theoretical Comp. Sci.
Graz University of Technology
Klosterwiesgaße 32/2
A-8010 Graz
<oaich@igi.tu-graz.ac.at>

Helmut Alt 81
Institut fur Informatik
Freie Universität Berlin
Takustraße 9
D-14195 Berlin
<alt@inf.fu-berlin.de>

Franz Aurenhammer 53
Inst. for Theoretical Comp. Sci.
Graz University of Technology
Klosterwiesgaße 32/2
A-8010 Graz
<aurenham@tcs.fu-berlin.de>

Rafael Ayala 29
Dpt. de Algebra, Comp., Geom. y Top.
Facultad de Matemáticas.
Universidad de Sevilla.
Apto. 1160.
E-41080 – Sevilla.

Ludger Becker 16
Inst. für numer. Math.-Informatik
Westfälische Wilhelms-Universität
Einsteinstraße 62
D-48149 Münster
<beckelu@math.uni-muenster.de>

Mark de Berg 89
Department of Computer Science,
Utrecht University
P.O. Box 80.089,
3508 TB Utrecht,
The Netherlands
<markdb@cs.ruu.nl>

Jean D. Boissonnat 62
INRIA, BP 93
F-06902 Sophia Antipolis Cedex

Lucian Cucu 5
RISC - Linz
A-4040 Linz,
<lcucu@risc.uni-linz.ac.at>

Olivier Devillers 62
INRIA, BP 93,
F-06902 Sophia Antipolis Cedex
<Olivier.Devillers@sophia.inria.fr>

Mario Dias 38
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<mdias@risc.uni-linz.ac.at>

Eladio Domínguez 29
Dpt. de Ing. Eléctrica e Informática.
Facultad de Ciencias.
Universidad de Zaragoza.
E-50009 – Zaragoza.
<ccia@cc.unizar.es>

Mircea Dragan 5
RISC - Linz
A-4040 Linz,
<mdragan@risc.uni-linz.ac.at>

Angel R. Frances 29
Dpt. de Ing. Eléctrica e Informática.
Facultad de Ciencias.
Universidad de Zaragoza.
E-50009 – Zaragoza.
<ccia@cc.unizar.es>

Carl van Geem 73
RISC-Linz,
Joh. Kepler University of Linz,
A-4040 Linz, Austria
<Carl.Van.Geem@risc.uni-linz.ac.at>

Francisco Gómez 64
Dpto. de Matemática Aplicada
E.U. de Informática
Universidad Politécnica de Madrid
Cta. de Valencia Km 7
E-28031 Madrid
<fmartin@eui.upm.es>

Laureano González-Vega 49
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<g_vega@ccucvx.unican.es>

Thorsten Graf 25
Inst. fur numer. Math.-Informatik
Westfälische Wilhelms-Universität
Einsteinstrasse 62
D-48149 Muenster
<graf@math.uni-muenster.de>

Marko de Groot 89
Department of Computer Science,
Utrecht University
P.O. Box 80.089,
3508 TB Utrecht,
The Netherlands
<marko@cs.ruu.nl>

Patrick Healy 33
Department of Computer Science
University of Limerick
Limerick, Ireland.?
<healyp@ul.ie>

Gregorio Hernández-Peñalver 68
Facultad de Informatica. UPM
Campus de Montegancedo
Boadilla del Monte.
E-28660 Madrid
<gregorio@fi.upm.es>

Klaus Hinrichs 16, 25
Inst. für numer. Math.-Informatik
Westfälische Wilhelms-Universität
Einsteinstraße 62
D-48149 Münster
<khh@math.uni-muenster.de>

Ferran Hurtado 64
Dept. Matemàtica Aplicada II
Universidad Politècnica de Catalunya
Pau Gargallo, 5
E-08028 Barcelona
<hurtado@ma2.upc.es>

Tudor Jebelean 5
RISC - Linz
A-4040 Linz,
<tjebelea@risc.uni-linz.ac.at>

Robert Juan-Arinyo 58
Dep. de Lleng. i Sist. Informàtics
Universitat Politècnica de Catalunya
Av. Diagonal 647, 8a
E–08028 Barcelona
<robert@lsi.upc.es>

Rolf Klein 1
FernUniversitat Hagen, ¨
Praktische Informatik VI,
Elberfelder Straße 95,
D-58084 Hagen.
<rolf.klein@fernuni-hagen.de>

Sylvain Lazard 62
INRIA, BP 93
F-06902 Sophia Antipolis Cedex
<Sylvain.Lazard@sophia.inria.fr>

Andrezj Lingas 1
Lund University
Dept. of Computer Science
Box 118
S-221 00 Lund
<andrzej@dna.lth.se>

Henri Lombardi 49
Laboratoire de Mathematiques
Université de Franche-Comté
F-25030 Besançon
<hl@grenet.fr>

Clovis Maliska Jr. 8
SINMEC - Dept. Engenharia Mecânica
Universidade Federal de Santa Catarina
88040.970 Florianópolis, SC
Brazil

Marisa Mazón 12
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<mazon@ccucvx.unican.es>

José Luis Montaña 45
Dep. de Mat., Estad. e Informática
Universidad Pública de Navarra
Pamplona, Spain
<montana1@ccucvx.unican.es>

José Enrique Morais 45
Departamento de Matematicas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander

Viorel Negru 5
RISC - Linz
A-4040 Linz.
<vnegru@risc.uni-linz.ac.at>

Hartmut Noltemeier 77
University of Würzburg
Dep. of Mathematics and Comp. Sci.
Am Hubland
D-97074 Würzburg.
<noltemei@informatik.uni-wuerzburg.dbp.de>

Luis Miguel Pardo 45
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<pardo@ccucvx.unican.es>

Antonio Quintero 29
Dpt. de Algebra, Comp., Geom. y Top.
Facultad de Matemáticas.
Universidad de Sevilla.
Apto. 1160.
E-41080 – Sevilla.
<quintero@algebra.us.es>

Pedro Ramos 64
Dpto. de Matemática Aplicada
E.U. de Informática
Universidad Politécnica de Madrid
Cta. de Valencia Km 7
E-28031 Madrid
<pramos@fi.upm.es>

Tomás Recio 12
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<recio@ccucvx.unican.es>

Waldir L. Roque 8
Instituto de Matemática
Univ. Federal do Rio Grande do Sul
91501-970 Porto Alegre, RS
Brazil
<roque@inf.ufsc.br>

Gunter Rote 81
Graz University of Technology
Institut für Mathematik (501B)
Kopernikusgaße 24
A-8010 Graz

Julio Rubio 29
Dpt. de Ing. Eléctrica e Informática.
Facultad de Ciencias.
Universidad de Zaragoza.
E-50009 – Zaragoza.
<rubio@cc.unizar.es>

Francisco Santos 20
Departamento de Matemáticas
Facultad de Ciencias
Universidad de Cantabria
E-39071 Santander
<santos@ccucvx.unican.es>

Peter-Michael Schmidt 41
Friedrich-Schiller-Universitaet Jena
Fakultät für Mathematik und Informatik
Universitätshochhaus 17.
OG, Leutragraben 1,
D-07743 Jena
<schmidt@minet.uni-jena.de>

Sven Schuierer 72
Institut für Informatik
Universität Freiburg
Rheinstr. 10-12
D-7800 Freiburg
<schuiere@fidji.informatik.uni-freiburg.de>

Sabine Stifter
RISC-Linz
Johannes Kepler Universitat
A-4040 Linz
<stifter@risc.uni-linz.ac.at>

Andreas Voigtmann 1
Inst. für numer. Math.-Informatik
Westfälische Wilhelms-Universität
Einsteinstraße 62
D-48149 Münster
<avoigt@math.uni-muenster.de>