14th Annual Fall Workshop on Computational Geometry

with a Focus on Open Problems

November 19–20, 2004

MIT, Cambridge, Massachusetts, USA

Abstracts

Table of Contents

Foreword

This booklet contains abstracts from talks presented at the *14th Annual Fall Workshop on Computational Geometry*, held on November 19–20, 2004 at MIT in Cambridge, Massachusetts, USA.

In addition to 4 invited talks, we had a record number of 31 contributed talks selected among a record number of 45 submissions.

Sponsors

- National Science Foundation
- Computer Science and Artificial Intelligence Laboratory
- Massachusetts Institute of Technology

Program and Organazing Committee

- Erik Demaine (chair, Massachusetts Institute of Technology)
- Martin Demaine (Massachusetts Institute of Technology)
- Piotr Indyk (Massachusetts Institute of Technology)
- Joseph S. B. Mitchell (Stony Brook University)
- Joseph O'Rourke (Smith College)
- Diane Souvaine (Tufts University)
- Ileana Streinu (Smith College)

History

This series of Fall Workshops on Computational Geometry was originally founded under the sponsorship of the Mathematical Sciences Institute (MSI) at Stony Brook (with funding from the U. S. Army Research Office) and held there from 1991 through 1995. It continued during 1996–1999 under the sponsorship of the Center for Geometric Computing, a collaborative center of Brown, Duke, and Johns Hopkins Universities, also funded by the U.S. Army Research Office. The workshop returned to Stony Brook for its tenth year, and then moved to Polytechnic University, Brooklyn, NY for its eleventh. The twelfth workshop (2002) was part of the Special Focus on Computational Geometry and Applications at DIMACS, while the thirteenth (2003) was part of the Special Semester on Computational Geometry at the Mathematical Sciences Research Institute, Berkeley. In 2004, we are proud to host the Fall Workshop on Computational Geometry at MIT, bringing the workshop to the Boston area for the first time and returning to the original format.

Topologically Sweeping the Complete Graph in Optimal Time

Eynat Rafalin * Diane Souvaine*

1 Introduction

We present a novel, simple and easily implementable[1] approach to sweep a complete graph of N vertices and k intersection points and report all intersections in optimal $O(k) = O(N^4)$ time and $O(N^2)$ space. Our method borrows the concept of horizon trees from the topological sweep method [6] and uses ideas from [9] to handle degeneracies. The novelty of the approach is the use of a *moving wall* that separates the graph into two regions at all times: the region in front of the wall that has known structure, and the region behind the wall that may contain intersections generated by edges that the sweep process has not yet predicted. This method has applications in computing the simplicial depth median of a point set in $\mathbb{R}^2$ [1]. Continuing research concentrates on modifying the topological sweep to work for arbitrary graphs. For some sparse graphs, where the total size of the cuts is limited, the proposed algorithm may be particularly effective.

A graph poses challenges for a sweep-line algorithm, that are not present in line arrangements, and that most existing sweep techniques do not handle. The structure holding the sweep-line status needs to be dynamic, as vertices and edges are constantly inserted and deleted. The event points now have two types and include both intersection points and graph vertices. Finally, *short* edges not yet encountered by the sweep line may create intersections that should be processed before intersections created by *long* edges that have already been detected (see Figure). Processing in the wrong order may introduce intersections not in the graph or ignore others, causing errors.

Existing algorithms for segment intersection detection do not work well for the complete graph. *Plane sweep* [3, 4] adds a $\log N$ factor to the optimal time complexity (yielding $O(N^4 \log N)$ time and $O(N^2)$ space). The optimal deterministic algorithm for the intersection reporting problem [5, 2] using $O(n \log n + k)$ time and $O(n)$ space, is dominated by $O(k) = O(N^4)$ when applied to a complete graph and is highly complex to implement. *Topological sweep* [6, 7, 9] offers a $\log n$ improvement factor over the vertical line sweep on an arrangement of n infinite lines, but does not handle finite line segments.

*Department of Computer Science, Tufts University, Medford, MA 02155. {erafalin, dls}@cs.tufts.edu

[1] On C++ code. Experimental results verify the time and space complexities.

2 Algorithm Overview

Let G be a planar embedding of a complete graph on N vertices. Assume vertices are in general position. The complete graph contains exactly $n = \frac{N(N-1)}{2}$ edges and k, the number of segment intersections, is $O(N^4)$. The number of edges cut by any sweep line is $O(N^2)$, but can be $\Theta(N^2)$ with $N/2$ vertices on each side of the line and $(\frac{N}{2})^2$ edges crossing it.

We sweep G from left to right to report all intersection points using a topological line (*cut*): a monotonic line in y-direction, intersecting each of the n edges *at most* once. The sequence of *active segments*, one per edge intersected by the topological line forms the *cut*. A segment of an edge is delimited by two adjacent intersection points or by the rightmost/leftmost intersection point and a vertex.

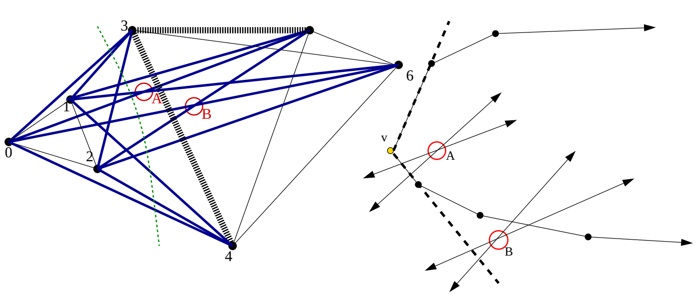
A sweep begins with the leftmost cut, which intersects no edges of the graph, and proceeds to the right in a series of elementary steps until it becomes the rightmost cut. An elementary step comprises the sweeping of the topological line past a vertex of the graph whose incoming edges are all currently intersected by the cut *or* past a *ready* intersection of edges that are consecutive in the current cut. There always exists a *ready* intersection or a *ready* vertex, whose incoming edges already lie on the cut, unless the cut is *rightmost* [10].

To maintain optimal time complexity linear in the size of the arrangement and space complexity linear in the maximal size of the cut, we build upon the concept of *horizon trees* [6]. Our *horizon graphs* are often not trees but *horizon forests*. The upper (resp. lower) *horizon forest* of the cut UHF (resp. LHF) is formed by extending the cut edges to the right. When two edges intersect, only the one of lower (resp. higher) slope continues to the right. Given LHF and UHF, the intersection of the right delimiters of the two forests produces the cut, which is computed from the updated LHF and UHF in constant time per active edge. A set of segments along the cut that contain the same intersection point as their right endpoint, generates a *ready intersection*, see [9, 6].

Intersection event points: The active segments switch position. To update UHF (resp. LHF) after processing the intersection point of segments $s_i, \ldots s_{i+k}$, for each segments s of the intersection apart from the first (last) one, traverse the *bay* formed by the segments above (below) s, until reaching the segment that intersects the extension of s (see [10, 6]).

Vertex event points: The sweep processes the vertices of the graph only when no intersection points are ready and in a left-to-right order, precisely when all the vertices' incoming edges are active: In a vertex event point all of the incoming active edges for vertex v are deleted from the set of active segments and all its outgoing edges are inserted. To update the horizon forests delete in-edges of v and insert its out-edges. Then update the horizon forest, walking in counterclockwise order around the bay formed by the previous edges to find the intersection point with an active edge. The edges emanating from the new vertex to the right, added to the set of active edges, may cut some of the existing active segments, and change the horizon forests and the cut. To update the horizon forests, test every existing forest edge for intersection with the new segment, in time linear in the size of the cut.

A moving wall: The set of active edges does not span the whole arrangement. Consequently, the sweep can encounter intersection points created by active edges before identifying intermediary edges that block them. For example, Figure (a) contains a graph of 7 vertices. The dotted sweep line produces the bold active segments. Intersection B of $\overline{06}$ and $\overline{25}$ is ready to be processed, as it is the right endpoint of the active segments associated with these edges. B, however, cannot be processed yet, as $\overline{06}$ and $\overline{25}$ intersect $\overline{34}$ before point B. Furthermore, if intersection B is processed, $\overline{06}$ and $\overline{25}$ will switch position in the cut, causing $\overline{34}$ to be inserted incorrectly. *Intersection points that are to the right of any edge that is not yet active cannot be* **ready**. Alternatively, consider intersection A. Given the dotted sweep line, both this intersection and vertex 3 are ready to be processed (in vertical sweep vertex 3 would be processed prior to A). However, the order of the cut edges currently places the edge $\overline{16}$ over edge $\overline{05}$. If vertex 3 is processed at this time, the update of the horizon forests and the cut will be incorrect. *All intersection points that are to the left of all segments emanating from vertex v must be processed before v is processed.*



Define a *moving wall* of a position of the sweep line as the semi-infinite lines corresponding with the two extreme edges emanating to the right from the next sweep vertex v_x. The moving walls for all vertices can be computed in $O(N \log N)$ time using [8]. At any time, the sweep line has to be forced to the current position of the moving wall (by processing a ready vertex only if it is to the left of the line defining the wall). Provably, a ready intersection always exists unless the sweep line is aligned with the moving wall or has reached the rightmost cut. To align the sweep line with the cut ensure that no intersection that is inside the wall is ready and that ready intersections that affect the wall are swept before the next vertex is processed.

There may be intersections inside the moving wall associated with v_x that can still be *legally* swept (e.g. intersection B in Figure (b)). The sweep line is *forced* only to the wall formed by the two extreme segments emanating to the right from v_x and their extension to infinity instead of including all *legal* ready intersections, so intersections that are ready but inside the moving wall are discarded (Figure (b)). These intersections are rediscovered when v_x is swept, by checking every pair of adjacent active segments in the cut for ready intersections. This step is performed once for each vertex, with a cost linear in the size of the active set.

Acknowledgment The authors wish to thank Prof. Ileana Streinu, Michael A. Burr and Ryan Coleman.

References

[1] G. Aloupis, S. Langerman, M. Soss, and G. Toussaint. Algorithms for bivariate medians and a Fermat-Torricelli problem for lines. *Comp. Geom. Theory and Appl.*, 26(1):69–79, 2003.

[2] I. J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[3] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, 1979.

[4] K. Q. Brown. Comments on "Algorithms for reporting and counting geometric intersections". *IEEE Trans. Comput.*, C-30:147–148, 1981.

[5] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[6] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, 38:165–194, 1989.

[7] H. Edelsbrunner and D. L. Souvaine. Computing median-of-squares regression lines and guided topological sweep. *J. Amer. Statist. Assoc.*, 85:115–119, 1990.

[8] F. P. Preparata. An optimal real-time algorithm for planar convex hulls. *Comm. ACM*, 22(7):402–405, 1979.

[9] E. Rafalin, D. Souvaine, and I. Streinu. Topological sweep in degenerate cases. In *Proc. 4th ALENEX*, volume 2409 of *LNCS*, pages 577–588. Springer-Verlag, 2002.

[10] E. Rafalin and D. L. Souvaine. Topologically sweeping the complete graph in optimal time and space. Tech. Report 2003-05, Tufts University.

Dynamic Update of Half-space Depth Contours

M. Burr* E. Rafalin* D. L. Souvaine*

Data depth is an approach to statistical analysis based on the geometry of the data. *Half-space depth*[1] has been studied most frequently by computational geometers. The *half-space depth* of a point x relative to a set of points $\mathcal{S} = \{X_1, ..., X_n\}$ in $\mathbb{R}^d$ is the minimum number of points of $\mathcal{S}$ lying in any closed half-space determined by a line through x [2, 11][2]. *Depth contours*, enclosing regions with increasing depth, help to visualize, quantify and compare data sets. Prior work investigated combinatorial properties and algorithms for computation of depth contours for *static* data sets. We present a *dynamic algorithm* for computing the two-dimensional *rank-based* half-space depth contours of a set of n points in $O(n \log n)$ time per operation and in $O(n^2)$ overall space, an improvement over the static version of $O(n^2)$ time per operation. The same algorithm can compute the half-space depth of a single point relative to a data set dynamically in $O(\log n)$ time and $O(n)$ space. The algorithm does not compute the entire set of contours explicitly but maintains the order (ranking) of points according to their half-space depth. A constant number of contours (e.g. $10\%, \cdots 100\%$) can be constructed in $O(n)$ time from the sorted list of the data points, ranked by depth. Our algorithm uses *generalized dynamic segment trees* to update the depth of every data point and is based on key characterizations of the potential changes in the depth contours upon insertions or deletions[3]. We only consider data sets in general position.

1 Preliminaries

The statistics community produced contradictory definitions for depth contours. The two main approaches were termed *cover* and *rank* [9]. The **cover** approach defines the contour of depth k as the boundary of the set of points in $\mathbb{R}^d$ with depth $\geq k$ (for half-space depth $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$). The *cover-based* half-space depth contour is provably the boundary of the intersection of all closed half-planes containing exactly $n - k + 1$ data points whose bounding line passes through two data points. The **rank** approach defines the αth central region as the convex hull containing the most central fraction of α sample points [5]. The α-central rank-based half-space-depth contour is constructed by sorting all points of the original set according to their half-space depth, yielding $\{X_{[1]}, \cdots X_{[n]}\}$, the *ranking order* of the points and taking the convex hull of data points $X_{[1]}, \cdots X_{[\alpha]}$. Both approaches assign the same depth value to points that are members of the data set $\mathcal{S}$ and create depth contours that are nested. The main visual difference: vertices of the *rank* contours are only data points while vertices of the *cover* contours can be any point from the data set.

Algorithms have existed for some time for constructing depth contours in 2 and higher dimensions under either definition. The best 2-D implementation for computing the ranking order of a set of points or all *cover*-based depth contours runs in $\Theta(n^2)$ [6]. Other implementations (e.g. [10, 3]) compute cover-based contours.

Much prior work exists on *dynamic geometric structures* (e.g. [7, 1]). To the best of our knowledge, we present the *first* dynamic algorithm for computation of half-space contours, addressing prior interest [4].

2 The Algorithm

Rank-based contours do not have the appealing properties of the *cover-based* contours: e.g. a unique structure that is relatively easy to update. Our algorithm utilizes the fact that a data point has equal depth values under the cover and rank approaches, and computes the *rank-based* contour by considering the *cover-based* contours. Thus, the analysis of our algorithm for the *rank-based* contours refers to the complexity of *cover-based* contours.

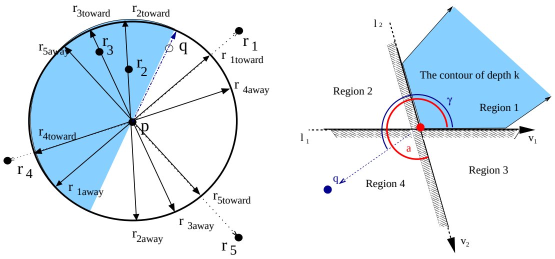
A key idea is, for each data point p, to consider every directed line l passing through p and another data point and the associated closed half-plane H_l to its right. H_l is represented by the unit vector v_{H_l} associated with l, see figure (a). For each point p there is a set of $2(n-1)$ unit vectors, that can be thought of as points on the unit circle, centered at p. Every point $r \in \mathcal{S} \backslash \{p\}$ is assigned to two antipodal vectors, $\hat{r}_{towards}$ and $\hat{r}_{away}$ (where $\hat{r}_{towards}$ is the vector pointing towards r). When a point q is inserted into or deleted from the data set, *exactly the half-planes with associated vectors in the semi-circle counter-clockwise from $\hat{q}_{toward}$ to $\hat{q}_{away}$* have their depth incremented or decremented. These half-planes are updated *simultaneously*, to recompute the depth of p efficiently. To do so we use the concept of *defining lines, half-planes and edges*. If p represents a data point of depth k with respect to set $\mathcal{S}$ of n points, p will appear *exactly* once on the (cover) contour of depth k. If p is on a non-degenerate contour it has exactly two incident edges, the *defining edges of*

*Department of Computer Science, Tufts University, Medford, MA 02155. {mburr,erafalin,dls}@cs.tufts.edu. Partially supported by NSF grant CCF-0431027

[1] Also called *location depth* or *Tukey depth*.

[2] For the reminder of the paper, every half-plane is considered *closed* unless otherwise mentioned.

[3] A detailed analysis can be found in [8] where we also present an $O(n \log^2 n)$ time and over all $O(n^2)$ space algorithm for dynamically computing *cover-based* half-space depth contours.



(a) (b)

p, on the contour of depth k. Every edge on any depth contour is a sub-segment of a line created by joining two data points q_1, q_2 of the set $\mathcal{S}$. The *defining lines* l_1, l_2 for p with respect to $\mathcal{S}$ are the lines containing p's *defining edges*. Each defining line l_i, $i \in \{1, 2\}$, bisects the plane into two closed half-planes containing $k+1$ and $n-k+1$ data points where the half-plane containing $k+1$ points does *not* contain the k-th depth contour. The *defining half-planes* H_{l_1}, H_{l_2} for p with respect to $\mathcal{S}$ are the *closed* half-planes bounded by the defining lines of p which contain $k+1$ data points

When point q is inserted into $\mathcal{S}$ the cover-based depth of a point $p \in \mathcal{S}$ remains unchanged if q is inside p's depth contour and can increase only if q is in the region outside p's depth contour. The update of every data point p, when point q is inserted to or deleted from $\mathcal{S}$, depends on the location of q relative to the *defining lines* for p[4]. *Nine cases* **completely** *determine how p's depth changes and how its two defining lines are transformed*[5], see figure (b). These updates are computed in $O(\log n)$ time for each data point p: the number of data points in every half-plane defined by p and each point in $\mathcal{S} \setminus p$ is recomputed; the defining lines of p with respect to the new data set $\mathcal{S} \cup q$ or $\mathcal{S} \setminus q$ are found; the *number of data points in the defining half-plane* is the *updated depth* of p; knowing all half-planes containing exactly $k' + 1$ data points and determined by a line through p and another data point makes it possible to determine p's *new defining half-planes* as well. (Note that at least one of the defining half-planes for every data point remains unchanged after a single insertion or deletion, see [8]).

Data Structures: For efficiency the algorithm uses *new generalized dynamic segment trees*. Each tree represents the half-planes passing through a data point $p \in \mathcal{S}$ and is implemented as an augmented dynamic red-black tree.

To re-sort data points, we use a linked list of buckets for depths, from 0 to $n/2$ (the minimum to maximum possible), to hold all data points. Bucket k holds a linked list of data points of depth k. Upon insertion or deletion every point q that changes its depth is moved from its old bucket to a new bucket. Since the depth of q changes by at most 1, the update takes $O(1)$ time.

3 Open Questions

- The lower bound for computing the half-space depth rank of data points (and thus their rank contours) is $\Omega(n \log n)$, based on reduction to sorting. We are seeking a method to order data points according to their depth in $o(n^2)$.
- To the best of our knowledge, no dynamic algorithm for computing depth contours according to other depth measures exists. We are working on a dynamic scheme to compute regression depth contours (envelopes of the arrangement of lines).
- Most real life experiments are high-dimensional. Since existing static algorithm for computing depth contours for most data depth measures are exponential in dimension, *dynamic approximation algorithms* for depth contours of *multivariate data* are needed.

Acknowledgement: The authors wish to thank S. Venkatasubramanian, S. Krishnan and R. Liu.

References

[1] Y. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proc. of the IEEE*, 80(9):1412–1434, 1992.

[2] J. Hodges. A bivariate sign test. *The Annals of Mathematical Statistics*, 26:523–527, 1955.

[3] S. Krishnan, N. H. Mustafa, and S. Venkatasubramanian. Hardware-assisted computation of depth contours. In *13th ACM-SIAM SODA*, 2002.

[4] R. Liu. Private communications, May 2003. Department of Statistics, Rutgers University.

[5] R. Liu, J. Parelius, and K. Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *The Annals of Statistics*, 27:783–858, 1999.

[6] K. Miller, S. Ramaswami, P. Rousseeuw, T. Sellarés, D. Souvaine, I. Streinu, and A. Struyf. Fast implementation of depth contours using topological sweep. In *Proc. 12th SIAM-ACM SODA*, pages 690–699, 2001.

[7] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. System Sci.*, 23(2):166–204, 1981.

[8] E. Rafalin, M. Burr, and D. L. Souvaine. Dynamic update of half-space depth contours. *to appear.*

[9] E. Rafalin and D. L. Souvaine. Data depth contours - a computational geometry perspective. Tech. Report 2004-01, Tufts University, CS Department, May 2004.

[10] I. Ruts and P. J. Rousseeuw. Computing depth contours of bivariate point clouds. *Comp. Stat. and Data Analysis*, 23:153–168, 1996.

[11] J. W. Tukey. Mathematics and the picturing of data. In *Proc. of the Int. Cong. of Math., Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que., 1975.

[4] The data point to be inserted or deleted lies in one of four regions or one of the defining lines, yielding nine cases.

[5] For example, if q is inserted into exactly one defining half-plane H_{l_2}, then p's depth is unchanged, but H_{l_2} is no longer a defining half-plane. It can be shown that the vector for p's new defining half-plane H_{m_2} is the first vector found by traversing the vectors starting from $v_{H_{l_2}}$ towards $v_{H_{l_1}}$ whose associated half-plane contains $k+1$ data points.

Computing a Point of High Simplicial Depth in R^2

Jason Burrowes-Jones*
Electrical Engineering
Howard University

William Steiger
Computer Science
Rutgers University

November 3, 2004

Abstract

Given a set $S = \{P_1, \ldots, P_n\}$ of n points in R^2, the *simplicial depth* $\sigma(Q)$ of a point $Q \in R^2$ is the number of open triangles $\Delta P_i P_j P_k$ that contain Q, $1 \le i < j < k \le n$. A point is Q is "deep" if $\sigma(Q) \ge \max_i \sigma(P_i)$. We give a simple, easily implementable $O(n(\log n)^2)$ deterministic algorithm to compute a deep point and we can also guarantee that Q has depth at least cn^3 for a constant $c > 0$.

1 Introduction and Summary

In 1974 John Tukey [11] proposed the now familiar notion of halfspace depth, generalizing from one dimension the idea of measuring depth by ranks. Since then several other mulitvariate depth measures have been proposed, e.g., hyperplane depth [10], Oja depth [9] (and also see [5]. Here we address the notion of simplicial depth proposed by Regina Liu in 1990 [8]. Given a set $S = \{P_1, \ldots, P_n\}$ of n *data points* in general position in R^d the *simplicial depth* of a point $Q \in R^d$ is defined to be

$$\sigma(Q) = |\{(i_1, \ldots, i_{d+1}), 1 \le i_1 < \cdots < i_{d+1} \le n \,:\, Q \in \Delta P_{i_1} P_{i_2} \cdots P_{i_{d+1}}\}|, \tag{1}$$

the number of open simplices whose vertices are points in S and which contain Q. This measure is affine invariant and robust over samples from a probability distribution.

Even in R^2 simplicial depth offers interesting computational and combinatorial challenges. The simplicial depth of a point Q can be computed in $\Theta(n \log n)$: once the radial ordering of the P_i about Q is known, $\sigma(Q)$ can be obtained in $O(n)$ further steps [5]; the lower bound is due to Aloupis et.al. [1]. By constructing the arrangement of the lines dual to the P_i, we obtain for each P_i, the radial order of the other points around it, and therefore can compute the simplicial depth of *all* points in S in time $O(n^2)$.

We call a point $Q \in R^2$ "deep for S" if $\sigma(Q) \ge \max_i \sigma(P_i)$, and by the previous observation, a deep point could be found in time $O(n^2)$. A point $P^* \in R^2$ (not necessarily in S) of maximal simplicial depth is called a *simplicial median*.

A max-depth data point (i.e., in S), though "deep" by definition, may actually have low depth (i.e., 0). However Boros and Füredi [3] showed that there is always a point $Q \in R^2$ that is contained in $2/9$ of the triangles determined by the points in S, but that NO point in R^2 is in $1/4 + \varepsilon$ of them. This means that a simplicial median P^* satisfies $n^3/27 \le \sigma(P^*) \le n^3/24 + O(n^2)$, a result generalized to higher dimension by Bárány [4].

It seems to be hard to find a simplicial median. The $O(n^4)$ time algorithm of Aloupis et.al. [2] is currently best. Therefore the following result may be useful and interesting.

*Research done as a student in the DIMACS Research Experiences for Undergraduates Program at Rutgers

Theorem 1 *Given n points in general position in the plane and $\varepsilon > 0$, let P^* denote a simplicial median and σ^* its depth. Then a deep point Q having depth at least $(1-\varepsilon)\sigma^*$ can be computed in time $O(n(\log n)^2)$.*

It is easy to find a point Q^* of the claimed depth, an "approximate median", and to do so within the claimed time bounds. The difficulty is to also guarantee that it is "deep". We do this with a pruning argument similar to the one used by Langerman and Steiger [7] for the case of hyperplane depth. A main ingredient is the following

Lemma 1 *Given a set S of n points in R^2, in time $O(n \log n)$ a point $Q' \in R^2$ can be found, along with its depth δ', and a "witness halfspace" h that contains at least cn points $P_i \in S$, $c > 0$, with $\delta(P_i) \leq \delta'$.*

Once we find Q', points in $S \cap h$ are pruned, and the Lemma is applied again to the points in $S \backslash (S \cap h)$, giving Q''. We keep the deeper point. After $O(\log n)$ such steps we have Q^+, a deep point for S. If we apply this procedure to $S \cup Q^*$, Q^* an approximate median, the deep point we get would satisfy the claims of the Theorem.

References

[1] G. Aloupis, C. Cortes, M. Soss, and G. Toussaint. Lower Bounds for Computing Statistical Depth. *Computational Statistics and Data Analysis 40*, 223-229 (2002).

[2] G. Aloupis, S. Langerman, M. Soss, and G. Toussaint. Algorithms for Bivariate medians and a Fermat-Torecelli Problem for Lines. *Comp. geom., theory and Application 26*, 69-79 (2003).

[3] E. Boros and Z. Füredi. The Number of triangles Covering the Center of a Set. *Geom. Dedicata 17*, 69-77 (1984).

[4] I. Bárány. A Generalization of Carathéodory's Theorem. *Discrete math. 40*, 141-152 (1982).

[5] J. Gill, W. Steiger, and A. Wigderson. Geometric Medians, *Discrete Math. 108*, 37-51. (1992).

[6] S. Jadhav and A. Mukhopadhyay. Computing a Centerpoint of a Finite Planar Set of Points in Linear Time. *Discrete and Comp. Geom. 12*, 291-312. (1994).

[7] S. Langerman and W. Steiger. Computing a High Depth Point in the Plane. *Developments in Robust Statistics. Proceedings of ICORS 2001*, R. Dutter, P. Filmoser, U. Gather, and P.J. Rousseeuw, eds. 227-233 (2002).

[8] R. Liu. On a Notion of Data Depth Based on Random Simplices. *Annals of Statistics 18*, 405-414 (1990).

[9] H Oja. Descriptive statistics for Multivariate Distributions. *Statistics and Probability Letters 1*, 327-332 (1983).

[10] P. Rousseeuw and M. Hubert. Depth in an Arangement of Hyperplanes. *Discrete and Comp. Geom. 22* 167-176 (1999).

[11] John Tukey. Mathematics and the Picturing of Data. *Proc. International Conf. of Mathematicians (Vancouver, 1974)* , Vol 2, 523-531 (1975).

Abstract, 14th Annual Fall Workshop on Computational Geometry, Boston, Nov 2004

Floodlight Illumination of Infinite Wedges

Matthew Cary[†] Atri Rudra[†] Ashish Sabharwal[†] Erik Vee[§]

[†]Computer Science and Engr, Box 352350, University of Washington, Seattle, WA 98195-2350

[§]IBM Almaden Research Center, Department K53/B2, 650 Harry Road, San Jose, CA 95120

{cary,atri,ashish}@cs.washington.edu vee@almaden.ibm.com

The floodlight illumination problem asks whether there exists a one-to-one placement of n floodlights illuminating infinite wedges of angles $\alpha_1, \ldots, \alpha_n$ at n locations $p_1, \ldots, p_n$ in a plane such that a given infinite wedge W of angle θ located at point q is completely illuminated by the floodlights. We prove that this problem is NP-hard, closing an open problem from CCCG 2001 [2]. In fact, we show that the problem is NP-complete even when $\alpha_i = \alpha$ for all $1 \leq i \leq n$ (the *uniform* case) and $\theta = \sum_{i=0}^{n} \alpha_i$ (the *tight* case). We discuss various approximate solutions and show that computing any *finite* approximation is NP-hard while ε*-angle* approximations can be obtained efficiently. Most proofs are omitted in this abstract due to lack of space. Interested readers are referred to [1].

1 Preliminaries

A *generalized wedge* is a wedge with a continuous finite region adjacent to its apex removed. The FLOODLIGHT ILLUMINATION problem on generalized wedges is as follows: given n sites $p_1, \ldots, p_n$, n angles $\alpha_1, \ldots, \alpha_n$, and a generalized wedge W, determine whether there is an assignment of angles to sites along with angle orientations that illuminates W. In the *tight* illumination problem, the sum of floodlight spans $\sum \alpha_i$ equals the wedge angle. A different specialization is the *uniform* problem, where in addition to being tight, $\alpha_i = \alpha_j$ for all i, j.

We now look at the related problem of MONOTONE MATCHING Suppose we are given n lines in the plane, $n+1$ vertical lines defining n finite width vertical slabs, and two points, one on the leftmost vertical line and one on the rightmost. Call this an *arrangement of lines and slabs*, and denote it by (L, S, λ, ρ), where L is the set of lines, $S \equiv \{s_1, \ldots, s_{n+1}\}$ is the set of vertical lines $x = s_i$ forming slabs, and λ and ρ are the two special points on the lines $x = s_1$ and $x = s_{n+1}$, respectively. A *monotone matching* in (L, S, λ, ρ) is a set of n line segments, each a portion of a unique line and spanning a unique slab, such that the following holds: (1) the left end point of the first segment is above λ, (2) the left endpoint of each subsequent segment is above the right endpoint of the segment in the previous slab, and (3) ρ is above the right endpoint of the last segment. In the more general problem of PSEUDOLINE MONOTONE MATCHING, one has to check whether a given arrangement of pseudolines[1] and slabs admits a monotone matching.

The floodlight illumiination problem can be related to the monotone matching problem through duality as described by Steiger and Streinu [3].

As a tool for our main result, we use NP-completeness of the problem of finding whether a given directed graph has a directed disjoint cycle cover.

Theorem 1. DIRECTED DISJOINT CYCLE COVER *is NP-complete, even for graphs with indegree and outdegree each bounded above by 3, as well as for graphs with outdegree exactly 3 and indegree at most 4.*

2 Floodlight Illumination is NP-Hard

To give a flavor of the proof of our main result, we prove the following result in this abstract:

Theorem 2. PSEUDOLINE MONOTONE MATCHING *is NP-complete.*

The most important gadget is the *forcing gadget*, shown in Figure 1. This is a sequence of slabs associated with pseudolines that forces the line used previous to the gadget to end below a chosen point, and the line used after the gadget to start above another chosen point.

Proof of Theorem 2. As a potential matching can easily be verified in polynomial time, this problem is in NP. The proof of NP-hardness is by a reduction from the bounded degree version of DIRECTED DISJOINT CYCLE COVER (see Theorem 1).

[1]A *pseudoline* is a curve in $\mathbb{R}^2$ that intersects any vertical line in exactly one point. A *collection of pseudolines* is a set of pseudolines no two of which intersect more than once.

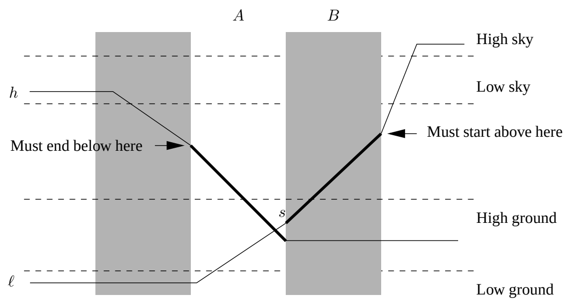


Figure 1: The forcing gadget. The arrows show how any lines used before or after the gadget are constrained.

Suppose we are given a directed graph $G = (V, E)$ with the outdegree of all vertices exactly 3 and the indegree at most 4. We will have gadgets $\mathbf{In}(v)$ and $\mathbf{Out}(u)$ for $u, v \in V$ as shown in Figure 2. Let $\mathcal{I}(v) \subset E$ be in the in-edges of v, and let $\mathcal{O}(u) \subset E$ be the out-edges of u. By our choice of G, $|\mathcal{I}(v)| \leq 4$ and $|\mathcal{O}(u)| = 3$.

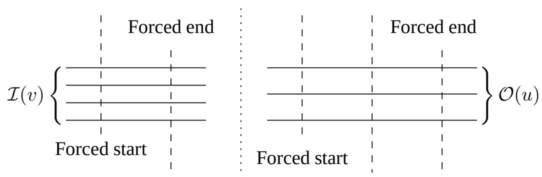


Figure 2: Graph gadgets $\mathbf{In}(v)$ and $\mathbf{Out}(u)$.

Let $n = |V|$ and $m = |E|$. We will use m primary pseudolines, each corresponding to an edge in E. There will be a number of auxiliary pseudolines used in forcing gadgets. The $\mathbf{Out}(\cdot)$ gadgets and $\mathbf{In}(\cdot)$ gadgets will be arranged in sequence as shown in Figure 3. The primary pseudoline corresponding to edge (u, v) will first pass through $\mathbf{Out}(u)$, and then pass through $\mathbf{In}(v)$.

We claim that when arranged as in Figure 3 along with appropriate forcing gadgets for each $\mathbf{In}(\cdot)$ and $\mathbf{Out}(\cdot)$ gadget, exactly one $e \in \mathcal{I}(v_i)$ is used in $\mathbf{In}(v_i)$ and exactly one $e \in \mathcal{O}(v_i)$ is not used in $\mathbf{Out}(v_i)$, $1 \leq i \leq n$.

A directed disjoint cycle cover of G is equivalent to a permutation π on the vertices, where $\pi(v)$ is the predecessor of v in the cycle containing v. If such a permutation exists, then a monotone matching exists, by not selecting the edge at $\mathbf{Out}(u)$ corresponding to $\pi^{-1}(u)$, and selecting the edge corresponding to $\pi(v)$ at $\mathbf{In}(v)$. Conversely, if a monotone matching exists, then the permutation π can be recovered by setting $\pi(v)$ equal to the edge that is used in $\mathbf{In}(v)$. This completes the reduction. ☐

The proof of NP-hardness of MONOTONE MATCHING with straight lines is based on the same idea but is somewhat more involved. The details can be found in [1]. From the duality between monotone matching and floodlight illumination, we have

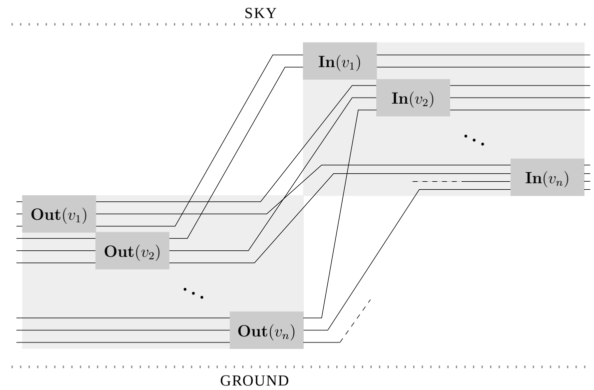


Figure 3: Overall view of the reduction from DIRECTED DISJOINT CYCLE COVER.

Theorem 3. FLOODLIGHT ILLUMINATION *is NP-hard. The tight, restricted, and uniform versions of the problem are NP-complete.*

3 Approximate Illumination

We now look at approximation algorithms to solve the floodlight illumination problem in the tight case. Let $\mathcal{F}$ be an illumination of a wedge W. $\mathcal{F}$ is a *finite-approximation* if it illuminates $W \setminus S$, where S is a finite region. $\mathcal{F}$ is an ε *angle-approximation* if it illuminates $W \setminus S_\varepsilon$, where S_ε is a union of wedges whose total angle is at most ε. We have the following result:

Theorem 4. *For the tight floodlight illumination problem, computing a finite-approximation is NP-hard, where as for any ε an ε angle-approximation can be found in polynomial time.*

References

[1] M. Cary, A. Rudra, A. Sabharwal, and E. Vee. Floodlight illumination of inifi nite wedges. Technical Report UW-CSE-2004-10-04, University of Washington, Seattle, Oct. 2004.

[2] E. Demaine and J. O'Rourke. Open problems from CCCG 2001. In *Proceedings of the 13th Canadian Conference on Computational Geometry*, Waterloo, Canada, Aug. 2001.

[3] W. L. Steiger and I. Streinu. Illumination by fbodlights. *Computational Geometry*, 10:57–70, 1998.

Proximity Problems on Line Segments Spanned by Points*

Ovidiu Daescu and Jun Luo
Department of Computer Science, University of Texas at Dallas
Richardson, TX 75080, USA

1 Abstract

We address proximity problems for lines and line segments spanned by points in the plane.

Closest (farthest) line segment from point. Given a set $S = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane, and another point q, find an extremal (closest, farthest) line segment from q among the set E of $O(n^2)$ line segments defined by the points in S.

The solutions to this problem are related to efficient solutions for the following problems.

Closest (farthest) line from point. Given a set $S = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane, and another point q, find an extremal (closest, farthest) line from q among the set L of $O(n^2)$ lines defined by the points in S.

There are a number of motivations for studying these proximity problems. For example, consider a vertex q that is to be inserted in a geometric graph G and assume one would like to place a label at q. For clarity in visualization, it is desirable the label does not intersect any edge that is not adjacent to q. If new edges can be created dynamically, a suitable test for deciding whether to place q at a particular location might include finding the closest edge to q among the edges of the complete graph G_c defined by the vertices of G.

The problems we study are related to the well known slope selection and distance selection problems, and to a number of optimization problems such as computing the largest empty disk or the smallest enclosing circle of S. The slope selection problem is to find the line in L with the k-th smallest slope and has been solved deterministically in $O(n \log n)$ time [2, 6]. A simple, randomized $O(n \log n)$ time algorithm has been presented in [3]. The distance selection problem is to find the k-th smallest distance between the points in S and can be solved in $O(n \log n + n^{2/3} k^{1/3} \log^{5/3} n)$ expected time [1] or in $O(n^{4/3} \log^2 n)$ deterministic time [6]. However, computing the smallest distance (the closest pair of S) or the largest distance (the diameter of S) are textbook problems in computational geometry and can be solved optimally in $O(n \log n)$ time and $O(n)$ space [8].

Computing the farthest or closest line of L from a point q is closely related to counting the number of lines in L that are intersected by a disk $\mathcal{D}$ centered at q. Since a line $l \in L$ that intersects $\mathcal{D}$ can be sandwiched between two lines (not necessarily in L) parallel to l and tangent to $\mathcal{D}$, using a standard

*This research was partially supported by NSF grant CCF-0430366.

point-line duality transform, the duals of the lines intersecting $\mathcal{D}$ correspond to points inside the *stabbing region* $\mathcal{D}^*$ of $\mathcal{D}$ [9]. The boundary $\partial\mathcal{D}^*$ of $\mathcal{D}^*$ is the set of points dual to the tangents of $\mathcal{D}$, and $\mathcal{D}^*$ is bounded from above by a convex x-monotone curve and from below by a concave x-monotone curve (see [9], page 255; in case of disks these curves are the two branches of a hyperbola [5]). Thus, to count the number of lines in L that are intersected by $\mathcal{D}$ it is enough to count the number of intersection points of the dual lines of S that are within $\mathcal{D}^*$. This can be done in $O(n \log n)$ time using Mount and Netanyahu's [7] algorithm for counting the number of line intersections inside a bounded region (two vertical lines corresponding to the smallest and largest slopes of the lines in L can be added). Since the only step of the algorithm in [7] that depends on the radius of $\mathcal{D}$ is the sorting step, one can use parametric search to compute the smallest radius disk centered at q and intersecting k lines of L, for any integer $1 \leq k \leq \binom{n}{2}$. The parametric search part is essentially the same as that for selecting vertices in arrangements (see [4], page 687), and takes $O(n \log^2 n)$ time. Then, the k closest lines from q can be reported in $O(n \log^2 n + k)$ time and $O(n)$ space.

Results. As with the distance selection problem, it would be interesting to see if computing an extremal line (or line segment) from q can be done faster than computing the k-th closest line (or line segment). We present $O(n \log n)$ time, $O(n)$ space algorithms for these problems. Our main result is summarized in the theorem below.

Theorem 1 *Given a set S of n points in the plane and another point q, the closest (farthest) line segment from q among those defined by S can be found in $O(n \log n)$ time and $O(n)$ space.*

Our solutions are based on simple geometric primitives and are easy to implement. A simple application of our techniques for computing the closest and farthest lines from q leads to an $O(n \log n)$ time, $O(n)$ space solution for the following problem.

Minimum (maximum) area anchored triangle. Given a set S of n points in the plane and an anchor point q, compute the minimum (maximum) area triangle defined by q with $S \setminus \{q\}$.

References

[1] T. Chan. On enumerating and selecting distances. *Internat. J. Comput. Geom. Appl.*, 11:291–304, 2001.

[2] R. Cole, J. Salowe, W. Steiger, and E. Szemeredi. An optimal-time algorithm for slope selection. *SIAM J. Comput.*, 18(4):792–810, 1989.

[3] M.B. Dillencourt, D.M. Mount, and N.S. Netanyahu. A randomized algorithm for slope selection. *Internat. J. Comput. Geom. Appl.*, 2(1):1–27, 1992.

[4] J.E. Goodman and J. O'Rourke. *Handbook of Discrete and Comput. Geometry*, CRC Press, 1997.

[5] F. Hurtado, M. Noy, P. Ramos, and C. Seara. Separating objects in the plane with wedges and strips. *Discrete Applied Mathematics*, 109:109–138, 2001.

[6] M.J. Katz and M. Sharir. Optimal slope selection via expanders. *Inf. Proc. Let.*, 47:115–122, 1993.

[7] D. Mount and N. Netanyahu. Efficient randomized algorithms for robust estimation of circular arcs and aligned ellipses. *Comput. Geom. Theory Appl.*, 19:1–33, 2001.

[8] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*, Springer-Varlag, Berlin/New York, 1985.

[9] M. Sharir and P.K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge Univ. Press, 1995.

Self-reconfiguring Robots

Daniela Rus
Computer Science and Artificial Intelligence Lab
MIT

We wish to create versatile robots by using self-reconfiguration: hundreds of small modules autonomously organize and reorganize as geometric structures to best fit the terrain on which the robot has to move, the shape of the object the robot has to manipulate, or the sensing needs for the given task. Self-reconfiguration allows large collections of small robots to actively organize as the most optimal geometric structure to perform useful coordinated work.

A self-reconfiguring robot consists of a set of identical modules that can dynamically and autonomously reconfigure in a variety of shapes, to best fit the terrain, environment, and task. Self-reconfiguration leads to versatile robots that can support multiple modalities of locomotion and manipulation. Self-reconfiguring robots constitute large scale distributed systems. Because the modules change their location continuously they also constitute ad-hoc networks.

This talk will discuss the geometric challenges of creating self-reconfiguring robots, ranging from designing hardware capable of self-reconfiguration to developing distributed controllers and planners for such systems that are scalable, adaptive, and support real-time behavior.

Unfolding Smooth Primsatoids

ABSTRACT

Nadia Benbernou* Patricia Cahn† Joseph O'Rourke‡

Abstract

We define a notion for unfolding smooth, ruled surfaces, and prove that every smooth prismatoid (the convex hull of two smooth curves lying in parallel planes), has a nonoverlapping "volcano unfolding." These unfoldings keep the base intact, unfold the sides outward, splayed around the base, and attach the top to the tip of some side rib. Our result answers a question for smooth prismatoids whose analog for polyhedral prismatoids remains unsolved.

Introduction. It is a long-unsolved problem to determine whether or not every convex polyhedron can be cut along its edges and unfolded flat into the plane to a single nonoverlapping simple polygon (see, e.g., [O'R00]), the *net*. These unfoldings are known as *edge unfoldings* because the surface cuts are along edges. In this paper,[1] we generalize edge unfoldings to certain piecewise-smooth ruled surfaces,and show that smooth prismatoids can always be unfolded without overlap. Our hope is that the smooth case will inform the polyhedral case.

Pyramids and Cones. A *pryamid* is a polyhedron that is the convex hull of a convex *base* polygon B and an *apex* v above the plane containing the base. The *side faces* are all triangles. It is trivial to unfold a pyramid without overlap: cut all side edges and no base edge. This produces what might be called a *volcano* unfolding. Examples are shown in Fig. 1(a,b) for regular polygon bases.

We generalize pyramids to *cones*: shapes that are the convex hull of a smooth convex curve base B lying in the xy-plane, and a point apex v above the plane. We define the volcano unfolding of a cone to be the natural limiting shape as the number of vertices of base polygonal approximations goes to infinity, and each side triangle approaches a segment *rib*. This limiting process is illustrated in Fig. 1(c). For any point $b \in \partial B$, the segment vb is unfolded across the tangent to B at b. Note that

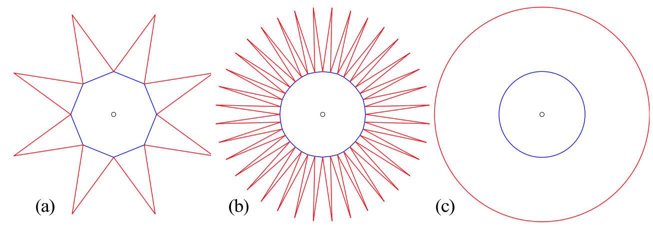


Figure 1: Unfoldings of regular pyramids (a-b) approaching the unfolding of a cone (c).

this net for a cone is no longer an unfolding that could be produced by paper, because the area increases.

Main Result. Our main result concerns a shape known as a *prismatoid*, the convex hull of two convex polygons A and B lying in parallel planes. There is no algorithm for edge-unfolding prismatoids. Our concentration in this paper is on *smooth prismatoids*, which we define as the convex hull of two smooth convex curves A above and B below, lying in parallel planes. A volcano unfolding of a smooth prismatoid unfolds every rib segment ab of the convex hull, $a \in \partial A$ and $b \in \partial B$, across the tangent to B at b, into the xy-plane, surrounding the base B, with the top A attached to one appropriately chosen rib. The main result of this paper is that every smooth prismatoid has a nonoverlapping volcano unfolding. Fig. 2 illustrates the side unfolding of a prismatoid; the top A must be carefully placed tangent to the side unfolding and on the convex hull of that unfolding.

References

[DO04] Erik D. Demaine and Joseph O'Rourke. *Folding and Unfolding in Computational Geometry.* 2004. Monograph in preparation.

[O'R00] Joseph O'Rourke. Folding and unfolding in computational geometry. In *Discrete Comput. Geom.*, volume 1763 of *Lecture Notes Comput. Sci.*, pages 258–266. Springer-Verlag, 2000. Papers from the *Japan Conf. Discrete Comput. Geom.*, Tokyo, Dec. 1998.

*Department of Mathematics `nbenbern@email.smith.edu`.

†Department of Mathematics `pcahn@email.smith.edu`.

‡Department of Computer Science, Smith College, Northampton, MA 01063, USA. `orourke@cs.smith.edu`. Supported by NSF Distinguished Teaching Scholars award DUE-0123154.

[1] See `http://arxiv.org/abs/cs/0407063` for the full version.

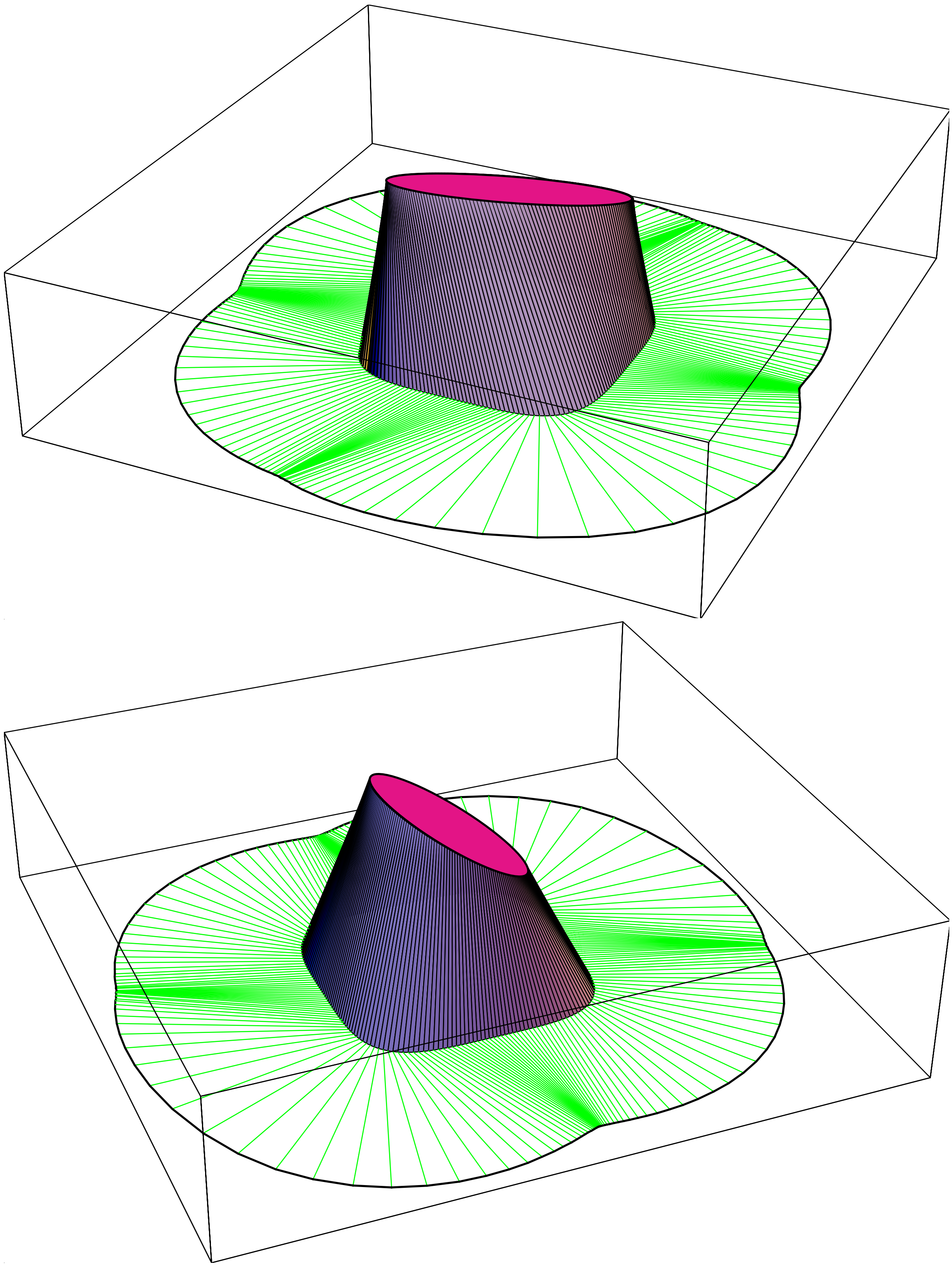


Figure 2: Two views of the side unfolding of a 3D prismatoid. The top A is an ellipse in a plane parallel to the base.

Folding Paper Shopping Bags

Devin. J. Balkcom* Erik D. Demaine†
Martin L. Demaine†

1 Introduction. In grocery stores around the world, people fold and unfold countless paper bags every day. The rectangular-bottomed paper bags that we know today are manufactured in their 3D shape, then folded flat for shipping and storage, and later unfolded for use. This process was revolutionized by Margaret Knight (1838–1914), who designed a machine in 1867 for automatically gluing and folding rectangular-bottomed paper bags [8]. Before then, paper bags were cut, glued, and folded by hand. Knight's machine effectively demolished the working-class profession of "paper folder".

Our work questions whether paper bags can be truly (mathematically) folded and unfolded in the way that happens many times daily in reality. More precisely, we consider foldings that use a finite number of creases, between which the paper must stay rigid and flat, as if the paper were made of plastic or metal plates connected by hinges. Such foldings are sometimes called *rigid origami*, being more restrictive than general origami foldings, which allow continuous bending and curving of the paper and thus effectively uncountably infinite "creasing". It is known that essentially everything can be folded by a continuous origami folding [6], but that this is not the case for rigid origami.

We prove that the rectangular-bottomed paper bag cannot be folded flat or unfolded from its flat state using the usual set of creases that are so common in reality—in fact, the bag cannot move at all from either its folded or unfolded state. However, we show that a different creasing of a paper bag enables it to fold flat from its 3D state. We also conjecture a way to unfold a paper bag from its flat state if it was already folded using the usual set of creases (by an adversary equipped with techniques from origami or reality).

2 Related Work. In the mathematical literature, the closest work to rigid folding is *rigidity*. The famous Bellows Theorem of Connelly, Sabitov, and Walz [4] says that any polyhedral piece of paper forming a closed surface preserves its volume when folded according to a finite number of creases. In contrast, as suggested by the existence of bellows in the real world, it is possible to change the volume using origami folding. Even more fundamental are Cauchy's rigidity theorem, Aleksandrov's extension, and Connelly's extension [2], which all establish an inability to fold a convex polyhedron using a finite number of creases. (In Cauchy's case, the creases must be precisely the edges of the polyhedron; in Connelly's case, any finite set of additional creases can be placed; Aleksandrov's theorem is somewhere in between.) Another result of Connelly[1] is that a positive-curvature "corner" (the cycle of facets surrounding a vertex in a convex polyhedron) cannot be turned "inside-out" no matter how we place finitely many additional creases; this result answers a problem of Gardner [7]. In contrast, a paper bag can be turned inside-out with an origami folding (and in real life) [3].

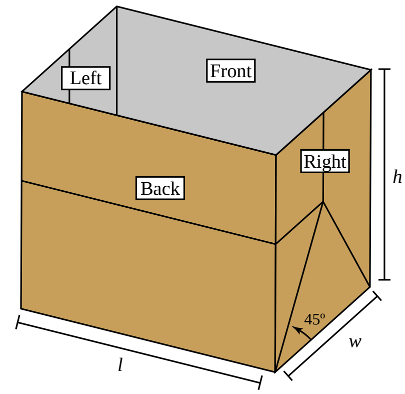


Figure 1: A shopping bag with creases in the usual places.

Few papers discuss rigid origami directly. Demaine and Demaine [5] present a family of origami "bases" that can be folded rigidly. Streinu and Whiteley [9] proved that any single-vertex crease pattern can be folded rigidly—up to but not included the moment at which multiple layers of paper coincide. Balkcom and Mason [1] demonstrate how some classes of origami can be rigidly folded by a robot.

3 Main Results. Figure 1 shows a shopping bag with the usual crease pattern, and dimensions w, l, and h. For the bag shown, $h > w/2$, and $l > w$.

Our first main result states that a shopping bag cannot be folded at all with just the usual creases:

Theorem 1 *A shopping bag with the usual crease pattern has a configuration space consisting of two isolated points, corresponding to the fully-open and fully-closed configurations.*

*Dartmouth Computer Science Department, Pittsburgh, PA 15213, devin@cs.dartmouth.edu

†MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine, mdemaine}@mit.edu

[1]Personal communication, 1998.

The results described in the previous section have two immediate consequences if we allow finitely many additional creases. First, the Bellows Theorem implies that, if the shopping bag had a top, no finite number of additional creases would allow the volume of the bag to be changed. Second, because the corners of the bag are convex, no finite number of additional creases would allow the shopping bag to be turned inside-out.

Based on these consequences, it might seem that no finite set of additional creases would allow a shopping bag to be folded flat. Our second result shows the opposite. A *short* shopping bag, with $h \leq w/2$, cannot have the usual shopping bag crease pattern, because the 45° creases do not intersect on the interior of the left and right sides of the bag; see Figure 2. In this case, we show

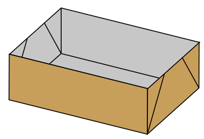


Figure 2: A short paper bag, similar to a collapsible department-store gift box.

Theorem 2 *Every short shopping bag (with $h \leq w/2$) can be collapsed flat using the creases in Figure 2.*

Now Theorem 2 suggests a method for folding a tall shopping bag: add creases to allow the tall bag to be telescoped until it is short enough to collapse flat. Figure 3 shows an animation of our procedure for shortening a bag by reducing h up to $\min\{w, l\}$. Using a sequence of these operations, we show

Theorem 3 *A tall shopping bag can be collapsed flat with the addition of finitely many creases.*

The collapsed state of the shopping bag after applying the folding technique described in the proof of theorem 3 is not the same as the collapsed state of the shopping bag with no additional creases. This difference suggests a more difficult question: can a collapsed shopping bag be opened up with the addition of a finite set of creases? We conjecture that it can, and propose a possible crease pattern in Figure 4.

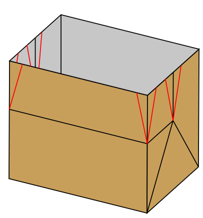


Figure 4: Conjectured creases for unfolding an already folded paper bag.

Conjecture 1 *A collapsed tall shopping bag can be unfolded with the addtion of a finite number of creases.*

If true, this conjecture would also offer a simpler way to flatten a tall shopping bag.

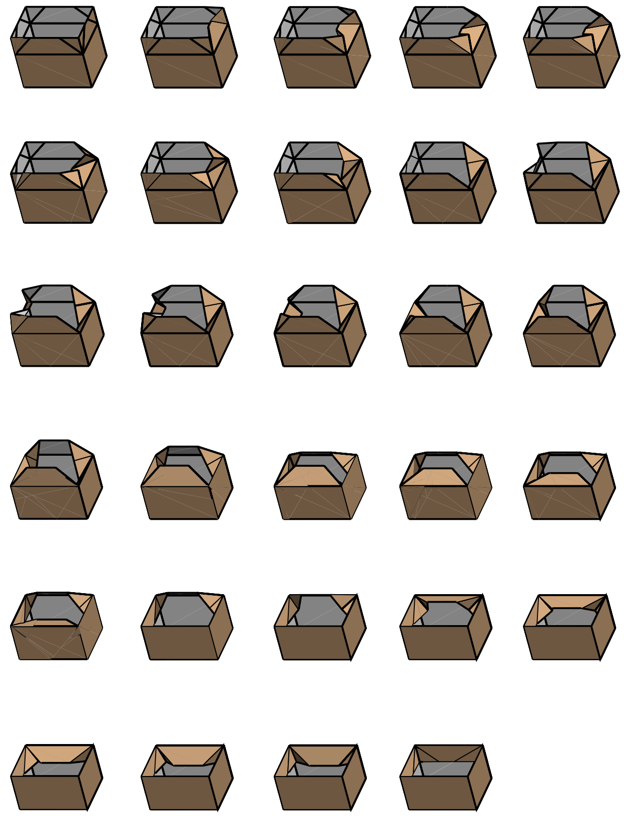


Figure 3: Procedure for shortening a rectangular tube.

References.

[1] D. J. Balkcom and M. T. Mason. Introducing robotic origami folding. In *Proc. 2004 IEEE International Conference on Robotics and Automation*, 2004.

[2] R. Connelly. The rigidity of certain cabled frameworks and the second-order rigidity of arbitrarily triangulated convex surfaces. *Advances in Mathematics*, 37(3):272–299, 1980.

[3] R. Connelly. Rigidity. In *Handbook of Convex Geometry*, volume A, pages 223–271. North-Holland, 1993.

[4] R. Connelly, I. Sabitov, and A. Walz. The bellows conjecture. *Beiträge zur Algebra und Geometrie (Contributions to Algebra and Geometry)*, 38(1):1–10, 1997.

[5] E. D. Demaine and M. L. Demaine. Computing extreme origami bases. Tech. Rep. CS-97-22, Dept. Computer Science, Univ. Waterloo, Canada, May 1997.

[6] E. D. Demaine, S. L. Devadoss, J. S. B. Mitchell, and J. O'Rourke. Continuous foldability of polygonal paper. In *Proc. 16th Canadian Conference on Computational Geometry*, pp. 64–67, Montréal, Canada, August 2004.

[7] M. Gardner. Tetraflexagons. In *The Second Scientific American Book of Mathematical Puzzles & Diversions*, ch. 2, pp. 24–31. University of Chicago Press, 1987.

[8] M. E. Knight. Paper bag machine. U.S. Patent 116,842, July 11 1871.

[9] I. Streinu and W. Whiteley. The spherical carpenter's rule problem and conical origami folds. In *Proc. 11th Annual Fall Workshop on Computational Geometry*, Brooklyn, New York, November 2001.

Hinged Dissection of Polypolyhedra

Erik D. Demaine* Martin L. Demaine*

Jeffrey F. Lindy† Diane L. Souvaine‡

1 Introduction. A *dissection* of two figures (solid 2D or 3D shapes, e.g., polygons or polyhedra) is a way to cut the first figure into finitely many (compact) pieces and to rigidly move those pieces to form the second figure. It is well-known that any two polygons of the same area have a dissection, but not every two polyhedra of the same volume have a dissection [3, 5].

A *hinged dissection* of two figures is a dissection in which the pieces are hinged together at points (in 2D or 3D) or along edges (in 3D), and there is a motion between the two figures that adheres to the hinging, keeping the hinge connections between pieces intact. While a few hinged dissections such as the one in Figure 1 are quite old (1902), hinged dissections have received most of their study in the last few years; see [6, 4]. It remains open whether every two polygons of the same area have a hinged dissection, or whether every two polyhedra that have a dissection also have a hinged dissection.

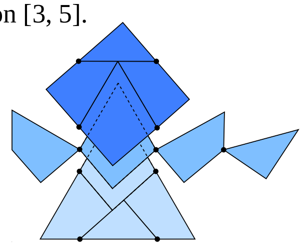


Figure 1: Hinged dissection of square and equilateral triangle, described by Dudeney [5].

2 Results. In this paper we develop a broad family of 3D hinged dissections for a class of polyhedra called polypolyhedra. For a polyhedron P with labeled faces, a *polypolyhedron of type* P is an interior-connected non-self-intersecting solid formed by joining several rigid copies of P wholly along identically labeled faces. (Such joinings are possible only for reflectionally symmetric faces.) Figure 2 shows two *polycubes* (where P is a cube).

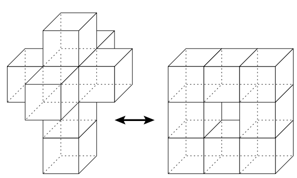


Figure 2: Two polycubes of order 8, which have a 24-piece edge-hinged dissection by our results.

For every polyhedron P and positive integer n, we develop one hinged dissection that folds into all (exponentially many) n-polyhedra of type P. The number of pieces in the hinged dissection is linear in n and the combinatorial complexity of P. For polyplatonics, we give particularly efficient hinged dissections, tuning the number of pieces to the minimum possible among a natural class of "regular" hinged dissections of polypolyhedra. For polyparallelepipeds (where P is any fixed parallelepiped), we give hinged dissections in which every piece is a scaled copy of P. All of our hinged dissections are hinged along edges and form a cyclic chain of pieces, which can be broken into a linear chain of pieces.

Our results generalize analogous results about hinged dissections of "polyforms" in 2D [4].

Like most previous theoretical work in hinged dissections, we do not know whether our hinged dissections can be folded from one configuration to another without self-intersection. However, we prove the existence of such motions for the most complicated gadget, the twister.

3 Proof Overview. Our construction of a hinged dissection of all n-polyhedra of type P divides into two parts. First, we find a suitable hinged dissection of the base polyhedron P. The exact constraints on this dissection vary, but two necessary properties are that the hinged dissection must be (1) cyclic, forming a closed chain of pieces, and (2) *exposed* in the sense that, for every face of P, there is a hinge in H that lies on the face (either interior to the face or on its boundary). For platonic solids, these hinges will be edges of the polyhedron; in the general case, we place these hinges along faces' lines of reflectional symmetry. Second, we repeat n copies of this hinged dissection of P, spliced together into one long closed chain. Finally, we prove that this new hinged dissection can fold into all n-polyhedra of type P, by induction on n.

3.1 Platonic Solids. Figure 3 shows an exposed cyclic hinged dissection of each of the platonic solids. Basically, each piece comes from carving the k-sided platonic solid into k face-based pyramids with the platonic solid's centroid as the apex. As drawn, these hinged dissections consist of k pieces, but by merging consecutive pairs of pieces along their common face, the number of pieces can be reduced to $k/2$ pieces while maintaining exposed hinges. These exposed hinged dissections have the fewest possible pieces, subject to the exposure constraint, because a hinge can simultaneously satisfy at most two faces of the original polyhedron.

3.2 General Case. In the general case, we use a 3D generalization of the straight skeleton [1] to decompose a given polyhedron into a collection of cells, exactly one cell per facet, such that exactly one cell is incident to each facet. These cells form the pieces in an exposed hinged dissection. For these pieces can be connected together into a cyclic hinged dissection, we need to first arrange for the polyhedron P to have a Hamiltonian dual graph.

In fact, we make two main modifications to P's sur-

*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA, email: {edemaine, mdemaine}@mit.edu. First author supported in part by NSF CAREER award CCF-0347776.

†Courant Institute of Mathematical Sciences, New York University, New York, NY, USA, email: lindy@cs.nyu.edu. Work begun when author was at Tufts University. Supported by NSF grant EIA-99-96237.

‡Department of Computer Science, Tufts University, Medford, MA, USA, email: dls@cs.tufts.edu. Supported by NSF grant EIA-99-96237.

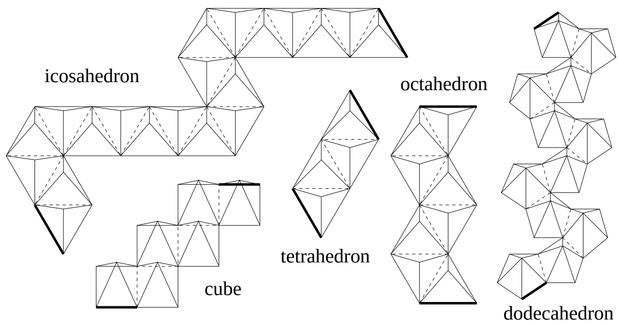


Figure 3: Unfolded exposed cyclic hinged dissections of the platonic solids. The bold lines indicate a pair of edges that are joined by a hinge but have been separated in this figure to permit unfolding. The dashed lines denote all other hinges between pieces. In the unfolding, the bases of all of the pyramid pieces lie on a plane, and the apexes lie above that plane (closer to the viewer).

face. First, we divide each reflectionally symmetric face of P along one of its lines of symmetry, producing a polyhedron P'. Second, we divide each face of P' so that any spanning tree of the faces in P' translates into a Hamiltonian cycle in the resulting polyhedron P''. This reduction is similar to the Hamiltonian triangulation result of [2] as well as a refinement for hinged dissection of 2D polyforms [4, Section 6]. We conceptually triangulate each face f of P' using chords (though we do not cut along the edges of that triangulation). Then, for each triangle, we cut from an arbitrarily chosen interior point to the midpoints of the three edges. Figure 4 shows an example. For any spanning tree of the faces of P', we can walk around the tree (follow an Eulerian tour) and produce a Hamiltonian cycle on the faces of P''.

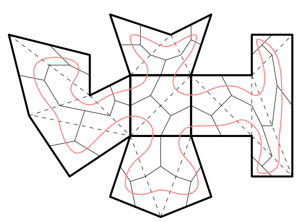


Figure 4: Hamiltonian refinement of five faces in a hypothetical polyhedron.

In particular, we can start from the matching on the faces of P' from the reflectionally symmetric pairing, and choose a spanning tree on the faces of P' that contains this matching. Then the resulting Hamiltonian cycle in P'' crosses a subdivided edge of every line of symmetry. (In fact, the Hamiltonian cycle crosses every subdivided edge of every line of symmetry.) Thus, in the exposed cyclic hinged dissection of the Hamiltonian polyhedron P'', there is an exposed hinge along every line of symmetry. Therefore all joinings between copies of P'' can use these hinges.

3.3 Putting Pieces Together. We use induction to prove that the nth repetition of the exposed cyclic hinged dissection of P described above can fold into any n-polyhedron of type P. The base case of $n = 1$ is trivial. Given an n-polyhedron Q of type P, one copy P_1 of P can be removed to produce an $(n-1)$-polyhedron Q'. By induction, the $(n-1)$st repetition of the exposed hinged dissection can fold into Q'. Also, P_1 itself can be decomposed into an instance of the exposed hinged dissection. Our goal is to merge these two hinged dissections. Essentially, we show that the hinged dissections can be placed against the shared face between P_1 and Q' in such a way that (1) a hinge of the exposed hinged dissection of P_1 coincides with a hinge of the hinged dissection of Q', and (2) the four pieces involved in these two hinges can be rehinged so that all pieces are connected in a single cycle, and that cycle is exactly the nth repetition of the exposed hinged dissection of P.

3.4 Mutually Rotated Base Polyhedra: Twisters. If a face is k-fold symmetric for $k \geq 3$, then there are several ways to glue two copies of P along this face. These different gluings produce different polypolyhedra if P itself is not k-fold symmetric. However, only one of the gluings can be produced by the inductive argument described above, because only one relative rotation will align the hinges that lie along the one chosen line of symmetry.

To enable these kinds of joinings, we embed the *twister gadget* shown in Figure 5 beneath each face of P'' that has k-fold symmetry for $k \geq 3$. This gadget consists of $8k$ cyclically hinged pieces that allow any integer multiple of $1/k$ rotation of one set of pieces with respect to the other pieces.

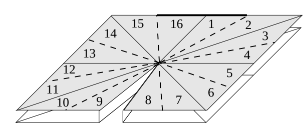


Figure 5: This 32-piece twister gadget allows turns of one-quarter of a twist. Although the pieces look two dimensional, they have thickness (they are prisms). The gaps between pieces 8 and 9 in subfigure (a) and between the top and bottom layers are for visual clarity only; in fact, the two layers are flush. Solid segments denote lengthwise hinges on the "inside" layer; dashed segments denote tiny hinges on the perimeter.

References.

[1] O. Aichholzer, F. Aurenhammer, D. Alberts, and B. Gärtner. A novel type of skeleton for polygons. *J. Univ. Comput. Sci.*, 1(12):752–761, 1995.

[2] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. *The Visual Computer*, 12(9):429–444, 1996.

[3] V. G. Boltianskii. *Hilbert's Third Problem*. V. H. Winston & Sons, 1978.

[4] E. D. Demaine, M. L. Demaine, D. Eppstein, G. N. Frederickson, and E. Friedman. Hinged dissection of polyominoes and polyforms. *Computational Geometry: Theory and Applications*. To appear. http://arXiv.org/abs/cs.CG/9907018.

[5] G. N. Frederickson. *Dissections: Plane and Fancy*. Cambridge University Press, November 1997.

[6] G. N. Frederickson. *Hinged Dissections: Swinging & Twisting*. Cambridge University Press, August 2002.

A 2-chain Can Interlock with a k-chain

ABSTRACT

Julie Glass[*] Stefan Langerman[†] Joseph O'Rourke[‡] Jack Snoeyink[§] Jianyuan K. Zhong[¶]

Abstract

One of the open problems posed in [3] is: what is the minimal number k such that an open, flexible k-chain can interlock with a flexible 2-chain? In this paper, we establish the assumption behind this problem, that there is indeed some k that achieves interlocking. We prove that a flexible 2-chain can interlock with a flexible, open 16-chain.

1 Introduction

A *polygonal chain* (or just *chain*) is a linkage of rigid bars (line segments, edges) connected at their endpoints (joints, vertices), which forms a simple path (an *open chain*) or a simple cycle (a *closed chain*). A *folding* of a chain is any reconfiguration obtained by moving the vertices so that the lengths of edges are preserved and the edges do not intersect or pass through one another. The vertices act as universal joints, so these are *flexible chains*. If a collection of chains cannot be separated by foldings, the chains are said to be *interlocked*.

Interlocking of polygonal chains was studied in [4, 3], establishing a number of results regarding which collection of chains can and cannot interlock. One of the open problems posed in [3] asked for the minimal k such that a flexible open k-chain can interlock with a flexible 2-chain. An unmentioned assumption behind this open problem is that there is some k that achieves interlocking. It is this question we address here, showing that $k = 16$ suffices.

It was conjectured in [3] that the minimal k satisfies $6 \le k \le 11$. This conjecture was based on a construction of an 11-chain that likely does interlock with a 2-chain. We employ some ideas from this construction in the example described here, but for a 16-chain. Our main contribution is a proof that $k = 16$ suffices. It appears that using more bars makes it easier to obtain a formal proof of interlockedness.[1]

Results from [3] include:

1. Two open 3-chains cannot interlock.
2. No collection of 2-chains can interlock.
3. A flexible open 3-chain can interlock with a flexible open 4-chain.

This third result is crucial to the construction we present, which establishes our main theorem, that a 2-chain can interlock a 16-chain (Theorem 1 below.)

2 Idea of Proof

We first sketch the main idea of the proof. If we could build a rigid trapezoid with small rings at its four vertices (T_1, T_2, T_3, T_4), this could interlock with a 2-chain, as illustrated in Figure 1(a). For then pulling vertex v of the 2-chain away from the trapezoid would necessarily diminish the half apex angle α, and pushing v down toward the trapezoid would increase α. But the only slack provided for α is that determined by the diameter of the rings. We make as our subgoal, then, building such a trapezoid.

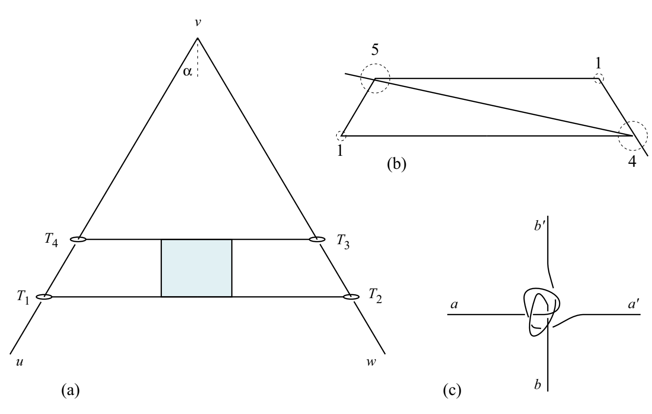


Figure 1: (a) A rigid trapezoid with rings would interlock with a 2-chain; (b) An open chain that simulates a rigid trapezoid; (b) Fixing a crossing of aa' with bb'.

We can construct a trapezoid with four links, and rigidify it with two crossing diagonal links. In fact, only

[*]Dept. of Math. & Comput. Sci., Calif. State Univ. Hayward, Hayward, CA 94542. jglass@csuhayward.edu

[†]Univ. Libre de Bruxelles, Département d'informatique, ULB CP212, Bruxelles, Belgium. Stefan.Langerman@ulb.ac.be

[‡]Dept. Comput. Sci., Smith College, Northampton, MA 01063. orourke@cs.smith.edu Partially supported by NSF DTS award DUE-0123154.

[§]Dept. Comput. Sci., Univ. North Carolina, Chapel Hill, NC 27599. snoeyink@cs.unc.edu

[¶]Dept. of Math. & Statistics Calif. State Univ., Sacramento 6000 J St., Sacramento, CA 95819. kzhong@csus.edu

[1]See http://arxiv.org/abs/cs.CG/0410052 for the full paper.

one diagonal is necessary to rigidify a trapezoid in the plane, but clearly a single diagonal leaves the freedom to fold along that diagonal in 3D. This freedom will be removed by the interlocked 2-chain, however, so a single diagonal suffices. To create this rigidified trapezoid with a single open chain, we need to employ 5 links, as shown in Figure 1(b). But this will only be rigid if the links that meet at the two vertices incident to the diagonal are truly "pinned" there. In general we want to take one subchain aa' and pin its crossing with another subchain bb' to some small region of space. See Figure 1(c) for the idea.

This pinning can be achieved by the "3/4-tangle" interlocking from [3], result (3) above; see Figure 2.

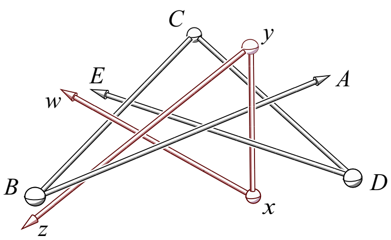


Figure 2: Fig. 6 from [3].

So the idea is replace the two critical crossings with a small copy of this configuration. This can be accomplished with 7 links per 3/4-tangle, but sharing with the incident incoming and outgoing trapezoid links potentially reduces the number of links needed per tangle. We have achieved 5 links at one tangle and 4 at the other. The other two vertices of the trapezoid need to simulate the rings in Figure 1(a), and this can be accomplished with one extra link per vertex. Together with the 5 links for the main trapezoid skeleton, we employ a total of $5 + (5 + 4 + 1 + 1) = 16$ links.

The final construction, shown in Figure 3, establishes our main result:

Theorem 1 *The 2-link chain is interlocked with the 16-link trapezoid chain.*

3 Discussion

We do not believe that $k = 16$ is minimal. We have designed two different 11-chains both of which appear to interlock with a 2-chain. However, both are based on a triangular skeleton rather than on a trapezoidal skeleton, and place the apex v of the 2-chain close to the 11-chain. It seems it will require a different proof technique to establish interlocking, for the simplicity of the proof presented here relies on the vertices of the 2-chain remaining far from the entangling chain.

Another direction to explore is closed chains, for which it is reasonable to expect fewer links. Replac-

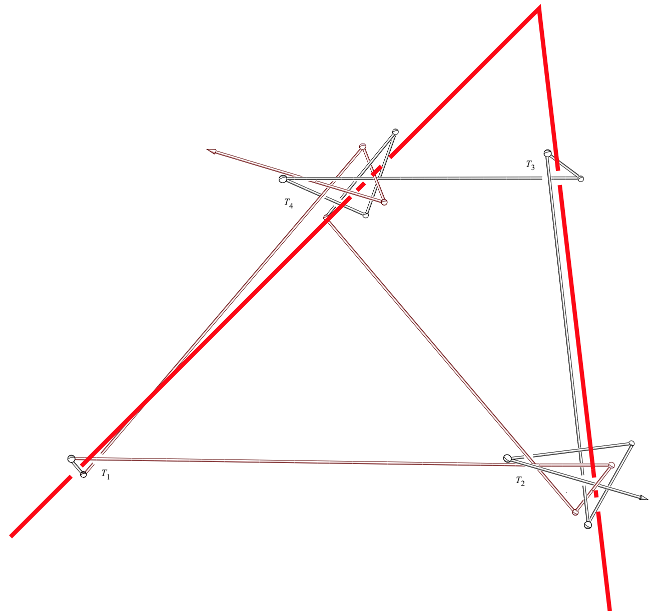


Figure 3: An open 16-chain forming a nearly rigid trapezoid, interlocked with a 2-chain (red).

ing the 3/4-tangles with "knitting needles" configurations [2][1] produces a closed chain that appears interlocked, but we have not determined the minimum number of links that can achieve this.

Acknowledgements

We thank Erik Demaine for discussions throughout this work, the participants of the DIMACS Reconnect Workshop held at St. Mary's College in July 2004 for helpful discussions, and Gillian Brunet and Meghan Irving for physical model construction: `http://cs.smith.edu/~orourke/Interlocked/Linkage.Model.html`.

References

[1] T. Biedl, E. Demaine, M. Demaine, S. Lazard, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, I. Streinu, G. Toussaint, and S. Whitesides. *Locked and unlocked polygonal chains in 3D.* Discrete Comput. Geom., 26(3):269–282, 2001.

[2] J. Cantarella and H. Johnston, *Nontrivial embeddings of polygonal intervals and unknots in* 3*-space*, Journal of Knot theory and Its Ramifications, 7(8): 1027–1039, 1998.

[3] E. D. Demaine, S. Langerman, J. O'Rourke, and J. Snoeyink, *Interlocked Open Linkages with Few Joints*, Proc. 18th ACM Sympos. Comput. Geom., 189–198, 2002.

[4] E. D. Demaine, S. Langerman, J. O'Rourke, and J. Snoeyink, *Interlocked open and closed linkages with few joints*, Comp. Geom. Theory Appl., 26(1): 37–45, 2003.

Grid Edge-Unfolding Orthostacks with Orthogonally Convex Slabs

Mirela Damian* Henk Meijer†

Abstract

We explore an instance of the question of unfolding an orthogonal polyhedron into one single connected piece that does not overlap itself and in which faces can only remain glued along whole edges.

1 Introduction

An *orthogonal* polygon P is a simple polygon with edges parallel to the coordinate axes. We use *horizontal* to refer to the x dimension and *vertical* to refer to the y dimension in the plane. P is called *orthogonally convex* if the intersection of P with any horizontal or vertical line is either empty or a single line segment.

A *slab* is a prism with an orthogonal polygon as basis and rectangular vertical faces. An *orthostack* S is an orthogonal polyhedron formed by stacking slabs $S_0, S_1, \ldots$ in the z dimension. Fig. 1 shows an orthostack example with two orthogonally convex slabs S_0 and S_1. If $z_0, z_1, z_2, \ldots$ are distinct

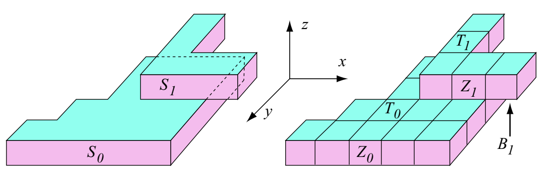


Figure 1: (a) An orthostack with two slabs S_0 and S_1. (b) Grid lines along which cuts are allowed.

z-coordinates of vertices of S, slab S_i is sandwiched between horizontal planes $z = z_i$ and $z = z_{i+1}$. The bottom B_i of S_i is the intersection of S_i with plane $z = z_i$, and the top T_i is the intersection of S_i with plane $z = z_{i+1}$. Note that T_i and B_i may be

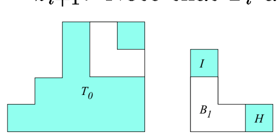


Figure 2: Top T_0 and bottom B_1 of the two slabs from Fig. 1; shaded pieces are exposed.

partly exposed (see the shaded pieces from Fig. 2) and partly interior to S.

In 3D we use *horizontal* to refer to a direction parallel to the xy plane and *vertical* to refer to a direction parallel to the xz or yz plane. The coordinate planes passing through every vertex of S induce a grid subdivision of S. We use the term *face* to refer to a surface rectangle in this subdivision. A *grid edge-unfolding* of S uses cuts along grid lines to unfold S into one connected planar piece in which faces remain connected along whole edges.

Demaine et. al[DIL04] show that all orthostacks can be *grid vertex-unfolded*, allowing faces to remain connected at single vertices. This abstract focuses on grid edge-unfolding of orthostacks with orthogonally convex slabs that satisfy an additional restriction: each maximal exposed piece of T_i contains two orthogonally incident edges of B_{i+1}.

A few more definitions are needed to present the algorithm. For any vertical face f of S, let $\alpha(f)$ be the space enclosed between the two planes orthogonal to f passing through the vertical edges of f. Define strip $s_i(f)$ as the maximum exposed connected component adjacent to f that lies in $\alpha(f) \cap (T_i \cup B_{i+1})$. For instance, the strip labeled C in Fig. 3b is precisely $s_1(L_0)$. The *band* (or *zone*) Z_i consists of all vertical faces of S_i. For any element x of S, we use the primed symbol x' to refer to x in the 2D unfolding.

2 Unfolding Algorithm

We proceed with an informal description of the unfolding with the help of the example from Fig. 1. Each band Z_i is cut along a vertical edge and unfolded in the plane horizontally, as in Fig. 3. Two vertical faces of each band Z_i are particularly important: $L_i^{'}$, the leftmost face and $R_i^{'}$, the rightmost face in the unfolding of Z_i. Bands for adjacent slabs are connected vertically by a *bridge*, a connected set of horizontal faces of S. A bridge connects $R_i^{'}$ to $L_{i+1}^{'}$ such that $L_{i+1}^{'}$ always lies vertically aligned or to the right of $R_i^{'}$ in the unfolding, as in Fig. 3. Exposed pieces of T_i and B_i are attached as vertical strips to opposite sides of $Z_i^{'}$.

2.1 Determining R_i, L_{i+1} and Bridge $R_i \to L_{i+1}$

Suppose that Z_{i-1}, L_i and the bridge connecting R_{i-1} to L_i have been flattened out in the plane.

*Dept. Comput. Sci., Villanova Univ., Villanova, PA 19085, USA. mirela.damian@villanova.edu.

†Dept. Comput. Sci., Queen's Univ., Kingston, Ontario K7L3N6, Canada. henk@cs.queensu.ca.

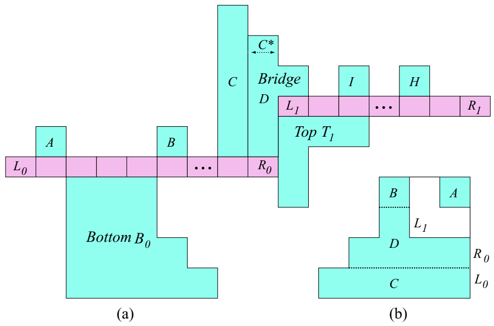


Figure 3: (a) An unfolding of S from Fig. 1 with bridge D. (b) Partition of T_0 into strips.

W.l.o.g., assume that the face of Z_i adjacent to the right edge of L_i' lies ccw from L_i, when viewed from $z = \infty$. For any face f of Z_i, let $\mathcal{H}(f)$ denote the open halfspace that contains f and is bounded by the vertical plane orthogonal to f passing through the vertical edge of f first encountered in a ccw walk from L_i.

Face R_i is not always clockwise adjacent to L_i. We pick R_i to be the face of Z_i last encountered in a ccw walk starting at L_i, with $R_i \neq L_i$, such that either R_i is adjacent to Z_{i+1}, or both of the following hold:

(1) The halfspace $\mathcal{H}(R_i)$ contains a face f of Z_{i+1} parallel to R_i

(2) There exists an exposed connected component $\mathcal{C}$ in $\mathcal{H}(R_i) \cap T_i$ or $\mathcal{H}(R_i) \cap B_{i+1}$ that is adjacent to both R_i and f.

We claim that R_i always exists. Furthermore, several faces f may satisfy conditions (1) and (2) above. We pick L_{i+1} to be the face f whose distance to the plane bounding $\mathcal{H}(R_i)$ is smallest. In case of a tie, L_{i+1} is the face *closest* to R_i, and thus uniquely defined.

Note that the subband $[L_i, R_i]$ ccw from L_i to R_i (when viewed from $z = \infty$) may not cover the entire Z_i (see Fig. 4b). However, we can show that $[L_i, R_i]$ is not too small either. In particular, $[L_i, R_i]$ contains at least two of the four monotone chains of the orthogonally convex polygon T_i. It is the existence of these chains that ensures that any face in $Z_i \setminus [L_i, R_i]$ can be attached by a vertical strip to one of the faces in $[L_i', R_i']$.

Having fixed R_i and L_{i+1}, we can now determine the bridge $R_i \to L_{i+1}$. If R_i is adjacent to Z_{i+1}, then R_i and L_{i+1} are adjacent and the bridge is empty. Otherwise, let $\mathcal{P}$ be the plane bounding the halfspace $\mathcal{H}(R_i)$ and let $\mathcal{C}$ be the maximum component that meets condition (2). The bridge $R_i \to L_{i+1}$ is $\mathcal{C}^* \cup s_i(L_{i+1})$, where $\mathcal{C}^*$ is the piece of $\mathcal{C}$ that lies between $\mathcal{P}$ and the plane parallel to $\mathcal{P}$ containing the vertical edge of L_{i+1} closer to $\mathcal{P}$.

2.2 Unfolding T_i, B_i and Remaining Z_i

Assume that the bridge $R_i \to L_{i+1}$ lies in $\mathcal{H}(R_i) \cap T_i$. To unfold T_i, we partition the exposed parts of T_i into strips as follows. First we delimit the region $\mathcal{X}$ of T_i enclosed between two planes parallel to L_i, one that contains L_i and the other passing through the vertical edge of R_i closer to L_i. Refer to Fig. 4b. We state without proof that $\mathcal{X}$ is entirely exposed. Next we partition $\mathcal{X}$ into strips parallel to L_i and $T_i \setminus \mathcal{X}$ into strips orthogonal to R_i, as in Figs. 3b and 4b. Each strip thus formed is precisely $s_i(f)$, for some f in $[L_i, R_i]$; we attach $s_i(f)$ above/below f', to complete the unfolding of T_i. We similarly unfold exposed faces of B_i (except for B_0, which we unfolded as one single piece).

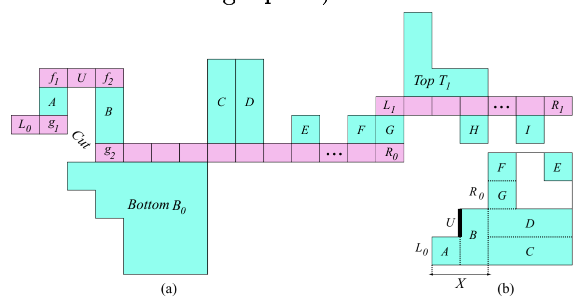


Figure 4: (a) An unfolding of S from Fig. 1 with bridge G. (b) Partition of T_0 into strips: A, B are parallel to L_0, and $C \ldots G$ are orthogonal to R_0.

If L_i and R_i are not adjacent in Z_i, the subband $Z_i \setminus [L_i, R_i]$ is yet to be unfolded. See Fig. 4b. We attach faces f in $Z_i \setminus [L_i, R_i]$ to vertical strips $s_i(f)$ already unfolded (see faces f_1 and f_2 from Fig. 4a). On top of each such face f' we attach vertically $s_{i-1}(f)$, if any. At the end of this process, all exposed faces of B_i have been unfolded.

Suppose now that two parallel faces f_1 and f_2 in $Z_i \setminus [L_i, R_i]$ are connected by a subband U of Z_i orthogonal to f_1 and f_2. Let g_1 and g_2 be the faces at the other end of the flattened strips $s_i(f_1)$ and $s_i(f_2)$. Then we cut Z_i' along the vertical edge shared by g_1 and g_2, and reconnect $s_i(f_1)$ and $s_i(f_2)$ by horizontal subband U. The case in which the bridge lies in $\mathcal{H}(R_i) \cap B_{i+1}$ is handled similarly.

References

[DIL04] E. D. Demaine and J. Iacono and S. Langerman. Grid Vertex-Unfolding of Orthostacks. In *Proc. of the Japan Conf. of Discr. and Comp. Geom.*, 2004.

A Theorem of Tutte and 3D Mesh Parameterization

Craig Gotsman

Technion – Israel Institute of Technology
and
Harvard University

Parameterization of 3D manifold mesh data involves embedding the mesh in some natural parametric domain, such as the plane or the sphere. Parameterization is important for many applications in geometry processing, including texture mapping, remeshing and morphing. The main objective is to generate a bijective mapping between the mesh surface and the parametric domain, which minimizes the distortion incurred in the transition in some meaningful sense. Examples of possible distortion are metric (edge length) distortion, conformal (angular) distortion and authalic (area) distortion.

A classical theorem of Tutte [7], originally designed to draw planar graphs, shows how to embed a manifold graph with the topology of a disk in the plane. This is achieved by fixing its boundary to a convex shape, and then solving a set of linear equations for the positions of the interior vertices. These equations express the fact that every interior vertex is positioned at the centroid of its neighbors. This basic method was later generalized by Floater [2] to arbitrary convex combinations, and Tutte's method could then be used to embed a 3D mesh in the plane, controlling the distortion by using convex weights derived from the geometry of the mesh. This class of embeddings are harmonic solutions of a discrete Laplace equation, namely requiring the weighted graph Laplacian operator to vanish at all interior vertices, with convex boundary conditions.

While Tutte's basic method remains a popular parameterization method, the constraint of a convex boundary is very severe, in most cases introducing unnecessary distortion into the result. Beyond that, it does not provide a satisfactory method to parameterize closed genus-0 meshes and meshes with higher genus. In this talk I will briefly survey some recent work of mine with colleagues on various generalizations of Tutte's method which overcome these problems.

Gortler, Gotsman and Thurston [3] provided conditions under which Tutte's method produces bijective embeddings even when the boundary is non-convex. This was used by Karni, Gotsman and Gortler [5] to generate free-boundary planar embeddings with constraints.

Gotsman, Gu and Sheffer [4] showed how to generalize the theory of Tutte to embed a closed genus-0 mesh on the sphere. This relies on recent algebraic characterizations of convex embeddings due to Colin de Verdiere [1], and related eigenvector constructions due to Lovasz and Schrijver [6]. In practice it involves solving a set of quadratic equations.

Inspired by recent work on discrete vector calculus, Gortler, Gotsman and Thurston [3] showed how the concept of a one-form from differential geometry can be defined on a discrete mesh. When such a one-form is harmonic, is may be used to generate bijective embeddings in the plane, in analogy to the Tutte method. The theory culminates in a discrete version of the Hopf-Poincare Index theorem, which may be used to provide a simple proof of the Tutte theorem, Moreover, it shows very simply (using a mere counting argument) how to generate a doubly-periodic embedding of the torus in the plane.

References

[1] Y. Colin de Verdiere. *Sur un nouvel invariant des graphes et un critere de planarite.* Journal of Combinatorial Theory B 50:11-21, 1990. [English translation: *On a new graph invariant and a criterion for planarity*, in: Graph Structure Theory (N. Robertson, P. Seymour, Eds.) Contemporary Mathematics, AMS, pp. 137-147, 1993.]

[2] M.S. Floater. *Parameterization and smooth approximation of surface triangulations.* Computer Aided Geometric Design, 14:231–250, 1997.

[3] S. J. Gortler, C. Gotsman and D. Thurston. *One-forms on meshes and applications to 3D mesh parameterization.* Harvard University CS TR-12-04, 2004.

[4] C. Gotsman, X. Gu and A. Sheffer. *Fundamentals of spherical parametrization for 3D meshes.* ACM Transactions on Graphics, 22(3):358-363 (Proc. SIGGRAPH), 2003.

[5] Z. Karni, C. Gotsman and S. Gortler. *Free-boundary parametrization in the presence of constraints.* Preprint, 2004.

[6] L. Lovasz and A. Schrijver. *On the nullspace of a Colin de Verdiere matrix.* Annales de l'Institute Fourier 49:1017-1026, 1999.

[7] W.T. Tutte. *How to draw a graph.* Proceedings of the London Mathematical Society, 13(3):743-768, 1963.

Drawing Equally-Spaced Curves between Two Points

Tetsuo Asano[1] and Takeshi Tokuyama[2]
[1] School of Information Science, JAIST, Japan
[2] Graduate School of Information Sciences, Tohoku University, Japan.

1 Problem Definition

Given two points a and b in the plane, draw n equally-spaced curves, $C_1, C_2, \ldots, C_n$ between them. More formally, those curves must satisfy the equi-distant property: for any point p on any such curve $C_i, 1 \leq i \leq n$ the two adjacent curves C_{i-1} and C_{i+1} are equally distant, that is, $d(p, C_{i-1}) = d(p, C_{i+1})$, where $d(p, C)$ is the distance from p to the point on a curve C that is closest to p. We define $C_0 = a$ and $C_{n+1} = b$ as degenerate curves. So, we have $d(p, C_0) = d(p, a)$ and $d(p, C_{n+1}) = d(p, b)$.

This problem is related to wire routing in printed circuit boards. In a new production technology more than one wires can be put between two pins. Then, it is desired to draw those wires so that they are equally spaced due to electrical constraints.

It is easy to draw one curve (line) between two points. It is simply a perpendicular bisector of the two points. It is also rather easy to draw three equally-spaced curves. The middle one is the perpendicular bisector of the two points and the remaining two curves are parabolas, each of which is a curve equidistant from a point and a line.

The case $n = 2$ is not easy. In this short paper we prove that we can draw two such equally spaced curves although we have no analytic equations for the curves.

2 Basic Properties of the Curves

For simplicity but without loss of generality we fix the two points a and b: $a = (3, 0)$ and $b = (-3, 0)$. We denote the two curves by C_a and C_b, which are characterized as follows:

(1) For any point p on C_a, $d(p, a) = d(p, C_b)$, and
(2) for any point p on C_b, $d(p, b) = d(p, C_a)$.

The curves C_a and C_b must pass through the points $(1, 0)$ and $(-1, 0)$, respectively, and the distance $d(p, C_a)$, for example, is given as the length of a line segment directed from p perpendicularly to the curve C_a.

Now, take any point p on C_a. Then, we have $d(p, a) = d(p, C_b)$ It implies that the circle centered at p with the point a on it must be tangent to the curve C_b at a unique point, which is denoted by q_p. The point q_p on C_b is called an image of the point p on C_a.

Fig. 1 shows this tangential property.

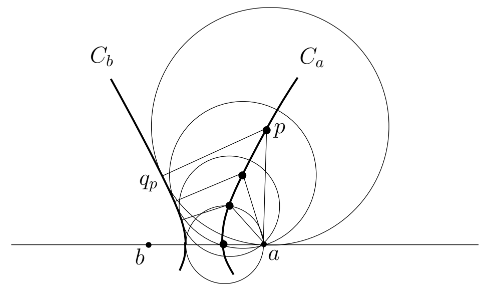


Figure 1: Tangential property.

By this tangential property we can guess a rough shapes of C_a and C_b, that is, they are smooth and convex toward the origin since they are envelopes of circles. As a point p goes to infinity along the curve C_a, the radius of its associated circle approaches to infinity, and finally it converges to a line.

3 Drawing Curves

The properties described above suggest equations specifying the curves C_a and C_b, but it seems to be hard to obtain analytic representations of the curves. Our strategy here is to compute rough shapes of the curves. For that purpose, we partition the plane into small squares of side length ε and remove all squares that cannot intersect the curves (see Fig. 2 for illustration). Due to the symmetry of the curves C_a and C_b, we only consider C_a.

Given a value of $\varepsilon > 0$, the plane is partitioned into squares. Each such square is specified by two-dimensional indices, as

$$s_{i,j} = [i\varepsilon, (i+1)\varepsilon] \times [j\varepsilon, (j+1)\varepsilon]. \qquad (1)$$

We start with finding a square $s_{0,j}$ which contains the point $(1, 0)$, the intersection of the curve C_a with the x-axis, and then remove all other squares with $i = 0$. At $i = 1$, we take squares near the square $s_{0,j}$ containing $(1, 0)$ and check their feasibility.

Feasible squares are defined as follow:

(1) The square containing the point $(1, 0)$ is feasible.
(2) A square $s_{i,j}$ is feasible if there exists a feasible square $s_{i',j'}$ such that

(i) $i' < i$,

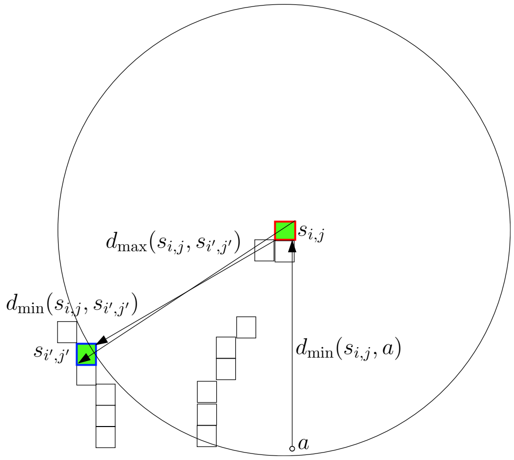


Figure 2: Feasible squares with related distances.

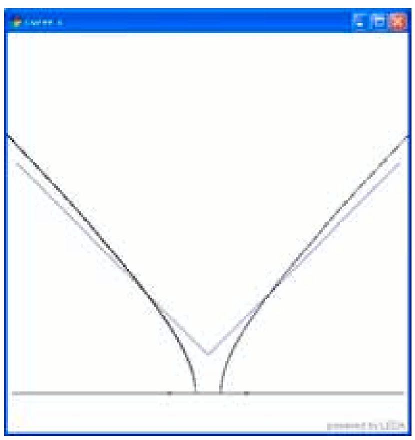


Figure 3: Two curves drawn by our approximation algorithm.

(ii) $[d_{\min}(s_{i,j},a), d_{\max}(s_{i,j},a)] \cap [d_{\min}(s_{i,j}, s_{i',-j'}), d_{\max}(s_{i,j}, s_{i',-j'})] \neq \emptyset$, and

(iii) there exists no square $s_{i'',j''}$ such that $d_{\max}(s_{i,j}, s_{i'',-j''}) < d_{\min}(s_{i,j}, a)$,

where note that if a square $s_{i,j}$ on the curve C_a is feasible then the square $s_{i,-j}$ is a feasible square on the curve C_b. $d_{\min}(s_{i,j},a)$ and $d_{\max}(s_{i,j},a)$ are the minimum and maximum distances between the square $s_{i,j}$ and the point a, and $d_{\min}(s,s')$ and $d_{\max}(s,s')$ are the minimum and maximum distances between two squares s and s'.

Once we have feasible squares $s_{i,j}, s_{i,j+1}, \ldots, s_{i,k}$ for i, a set of feasible squares for $i+1$ should start somewhere around j and end somewhere near k on $i+1$. Fig. 3 shows our implementation result.

4 More Characterizations

There are some curves and lines that characterize the equally-spaced curves C_a and C_b. Because of symmetricity of the curves we only consider the upper halves of C_a and C_b.

What is an optimal pair of half lines to approximate them to minimize the error? Because of their symmtricity, we can assume that a pair $y = cx + d, x \geq 0$ and $y = -cx + d, x \leq 0$ is optimal, where $c, d > 0$. An error for a point $p(x,y)$ on $y = cx + d$ is given by the difference between the distance from p to a and the length of a perpendicular line segment from p to the line $y = -cx + d$. If we denote the other endpoint of the segment by (x', y'), then the difference d is given by

$$\begin{aligned} d &= \sqrt{(x-1)^2 + y^2} - \sqrt{(x-x')^2 + (y-y')^2} \\ &= \frac{(x-1)^2 + y^2 - (x-x')^2 - (y-y')^2}{\sqrt{(x-1)^2 + y^2} + \sqrt{(x-x')^2 + (y-y')^2}}. \end{aligned}$$

For this difference to converge as x goes to infinity, the coefficient of x^2 in the numerator must be 0, that is, $c = 1$.

Now, the difference is simplified to

$$d = \frac{2(d-1)x + 1 + d^2}{\sqrt{(x-1)^2 + (x+d)^2} + \sqrt{2x^2}}.$$

It is minimized when $d = 1$. In fact, when $c = d = 1$, the difference converges to 0 as x goes to infinity.

We have shown that an optimal pair of half lines to approximate the curves is given by $y = x + 1, x \geq 0$ and $y = -x + 1, x \leq 0$. However, it does not imply that they are asymptotes of the curves. For any point p on C_a its corresponding point q_p on C_b that is closest to p on C_b is defined as the image of p. If we move a point p along C_a toward infinity, its image also moves on C_b in the direction away from the origin. Then, does it also go to infinity? The answer is no. Images cannot go beyond some point q_∞ on C_b.

This also means that if a point p on C_a is sufficiently far away from the origin then it must be close to the bisector of the two points a and q_∞. So, the bisectors can be considered as the asymptotes of our curves.

5 Future Direction

We are now working on a Voronoi diagram based on the curves defined here. We call it a Voronoi diagram with neutral zones. We have obtained preliminary results.

Acknowledgment

The problem was originally given by Dr. Hiroshi Murata, a visiting professor, The University of Kitakyushu, Japan. The authors would like to thank Benjamin Doerr, Alexander Wolff, and Sergei Bereg for their discussions. This work is partially supported by Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas.

A Distributed Architecture for the Visualization of Geometric Models

Sourav Dalal* Frank Dévai* Md Mizanur Rahman*

The geometric aspects of the visualisation of large data sets, in particular, digital models of real or planned solid objects in a heterogeneous distributed environment are investigated. One of the fundamental problems of rendering is visibility determination; the process of deciding what parts of the surface of the model can be seen from a possibly moving point. Any algorithm for determining the visibility of a set of polygons in three-dimensional space with a total of N edges takes $\Theta(N^2)$ time in the worst case. This growth rate is a serious difficulty. To overcome it hardware accelerators are used in low-end systems, and parallel algorithms are proposed for high-performance graphics systems. For example, the hidden-line problem can be solved in $\Theta(\log N)$ time either on N^2 EREW or on $N^2/\log N$ CREW PRAM processors, and the hidden-surface problem in the same $\Theta(\log N)$ time on N^2 CREW PRAM processors. The $\Theta(\log N)$ result cannot be further improved even if arbitrarily many processors were available [2, 3]. Algorithms with a smaller number of processors, called *coarse-grain algorithms*, can also be derived from these results, e.g., a hidden-surface algorithm that takes $O(\frac{N^2 \log N}{p} + \log N)$ time in the worst case by using p CREW PRAM processors [3].

Unfortunately, parallel architectures either suffer from the problem of latency or do not scale well, hence are expensive. Using distributed shared memory (DSM) running on a network of workstations [1] seems to be a better alternative. There are some drawbacks, however, e.g., overhead: if a processor needs to access a variable available only on a remote machine, the whole page must be transferred. Experts like Tanenbaum and van Steen [5] are sceptical: *"After almost 15 years of research on distributed shared memory, DSM researchers are still struggling to combine efficiency and programmability. To attain high performance on large-scale multicomputers, programmers resort to message passing despite its higher complexity compared to programming (virtual) shared memory systems. It seems therefore justified to conclude that DSM for high-performance parallel programming cannot fulfil its initial expectations."*

The objective of this research is the design and implementation of a light-weight, inexpensive message-passing architecture for the visualisation of large geometric models. This system also uses networks of workstations, typically available in a university environment, but unlike DSM, it is a more efficient approach using scanline algorithms. A scanline is a row of pixels of the image. The pixels of a scanline can be obtained by determining the visibility of the objects in the plane perpendicular to the screen and containing the scanline. The intersection of a polygon-mesh model (i.e., a set of polygons) and a plane is a set of line segments in the plane.

Our system is based on a *client-server architecture*, where a number of servers support typically one client. Each server is responsible for the calculation of a couple of scanlines, and the delivery of the results back to the client which displays the image.

The operation of the system is outlined as follows. The user, sitting in front of a graphics workstation running the client software, selects a geometric model or virtual environment they want to visualize. Once the geometric model has been selected, the client software distributes the corresponding data set on a high-speed network by using *scalable reliable multicast*. All workstations having the server software installed pick up the model, and store it on their local disk. On user interaction, e.g., zoom, translation or rotation of the model, the client broadcasts a 4x4 homogeneous transformation matrix. All the available servers receiving this matrix perform the required transformation on their stored model.

Each server is associated with a unique name, which is an integer in the range $[0, m-1]$, where m is the total number of the available servers at any given point of time. In reply to a client request, each server calculates a number of scanlines. Scanlines are numbered from 0 to $h-1$, where h is the height of the image, i.e., the total number of scanlines on the screen of the client's workstation. Server k calculates scanlines $i = k + jm$, for $j = 0, 1, 2, \ldots$, such that $i < h$, and sends them back to the client.

*London South Bank University, London, 103 Borough Road, London, SE1 0AA, UK, email: fl.devai@lsbu.ac.uk

The client then assembles the scanlines, and displays the next frame. It also calculates a new transformation matrix from user input, gathers server statistics, rename servers if necessary and informs each server involved. Then it goes back to the beginning of the loop broadcasting a new transformation matrix and requesting scanlines.

In the unlikely case of the output of a single server is lost, the scanlines from the previous frame are used. Often there is no difference between the same scanlines of subsequent frames (e.g., when a designer is contemplating a part of the model) and the difference is hardly noticeable on moving images. (Note that the scanlines are delivered by the servers in an interleaved pattern: a scanline from server k is followed by one from server $k+1$, and so on.)

If, however, the client notices from server statistics that the output of a server is consistently lost in the last few frames, the client renames the servers: If, say, server k is not responding, server $m-1$ is renamed as k, and m is reduced by 1. If a server re-appears, it is given the name m, and m is incremented by 1.

From the above it follows that the system is *fault tolerant* in the sense that it tolerates both lost messages and server failures. The latter is particularly important, because in this way the workstations do not have to be dedicated to the system. The servers run on workstations as background processes; when a workstation gets idle, the server joins the system, and it leaves the system when preempted by an interactive process. Thus, the hardware cost of the system is negligible, as organizations like universities or design offices already have extensive workstation networks.

Another significant advantage of the proposed architecture is that the servers only need to compute a planar visibility problem with a complexity of $\Theta(n \log n)$ rather than the 3D problem of complexity $\Theta(N^2)$, where $n \leq N$ is the number of line segments in the plane of the scanline. The planar visibility problem is well understood, and this research gave us the opportunity to implement 10 scanline algorithms, and analyse their asymptotic resource requirements.

Asymptotic analysis, however, cannot take into consideration constant factors, which can be different in different environments. Therefore a *portable testbed* was developed for the comparative evaluation of the actual performance of the algorithms on the particular hardware-software platform they are used. In this way in a heterogeneous distributed system the fastest version of the server can be installed in any given machine environment.

Considering the number of possible algorithms together with their variants, the number of time measurements required for conclusive results is substantial, hence using real 3D models would be too expensive and time consuming. Since the input to a scan-line algorithm is only a planar set of line segments, a more efficient test-data generation method based on random line segments was developed.

As some scanline algorithms exploit the fact that the input obtained from real solid models results in a set of line segments that are non-intersecting (except at their endpoints) a test-data generation method was required to produce a planar set of non-intersecting random line segments. Our method takes a total of $4n$ random numbers and $O(n \log n)$ time in the worst case to generate $4n$ coordinates for a set of n non-intersecting random line segments in the plane.

The Java language was used for both the development of a prototype distributed system and the testbed. Since the running time of scanline algorithms is very short, accurate time-measurement techniques are required. These were implemented by reading the time-stamp counters of the processors using Java native methods.

Though the planar visibility problem is well researched, we encountered some interesting problems which, we believe, are unsolved. For example, one of our algorithms, called the *priority-queue method*, gave better than expected experimental results. This algorithm uses a heap to maintain an order on line segments. It is known that n elements can be inserted in a heap in $O(n)$ expected time, but deleting the minimum element takes $\Theta(\log n)$ time on average [4]. Our algorithm, however, deletes *arbitrary elements*, which are near to leaf nodes most of the time, thus repairing the heap costs little. So far, however, we could not turn our arguments into a formal proof.

References

[1] Amza, C. *et al.* TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[2] Dévai, F. On the computational requirements of virtual reality systems. *State of the Art Reports*, EUROGRAPHICS'97, Budapest, 1997, 59–92.

[3] Dévai, F. Parallel algorithms for visibility computations. Proc. EUROGRAPHICS *UK Chapter 17th Annual Conference*, Cambridge, UK, 1999.

[4] Schaffer, R. and Sedgewick, R. The analysis of heapsort. *J. Algorithms*, 15(1):76–100, 1993.

[5] Tanenbaum, A. S. and van Steen, M. *Distributed Systems—Principles and Paradigms.* Prentice Hall, NJ, 2002 (p 32)

Pointed Binary Encompassing Trees: Simple and Optimal

Michael Hoffmann
Institute for Theoretical Comp. Sci.
ETH Zürich, CH-8092, Switzerland
hoffmann@inf.ethz.ch

Csaba D. Tóth
Department of Mathematics
MIT, Cambridge, MA 02139
toth@math.mit.edu

Abstract. For n disjoint line segments in the plane we can construct a binary encompassing tree such that every vertex is *pointed*, what's more, at every segment endpoint all incident edges lie in a halfplane defined by the incident input segment. Our algorithm runs in $O(n \log n)$ time which is known to be optimal in the algebraic computation tree model.

Introduction. Interconnection graphs of disjoint line segments in the plane are fundamental structures in computational geometry, and often more complex objects are modelled by their boundary segments or polygons. One particularly well-studied example is a crossing-free spanning graph: the *encompassing graph* for disjoint line segments in the plane is a connected planar straight line graph (PSLG) whose vertices are the segment endpoints and that contains every input segment as an edge.

A simple construction shows that not every set of n disjoint segments in the plane admits an *encompassing path*. But there is always a path that encompasses $\Theta(\log n)$ segments and does not cross any other input segment [6]. The question, whether encompassing trees of bounded degree exist, was answered in the affirmative by Bose and Toussaint [4]. Later Bose, Houle and Toussaint [3] constructed an encompassing tree of maximal degree *three* in $O(n \log n)$ time and proved that the runtime is optimal.

Recently, Hoffmann, Speckmann, and Tóth [5] have shown that for every set of disjoint segments a *pointed* binary encompassing tree can be constructed in $O(n^{4/3} \log n)$ time. A PSLG is *pointed* iff at every vertex p all incident edges lie on one side of a line through p.

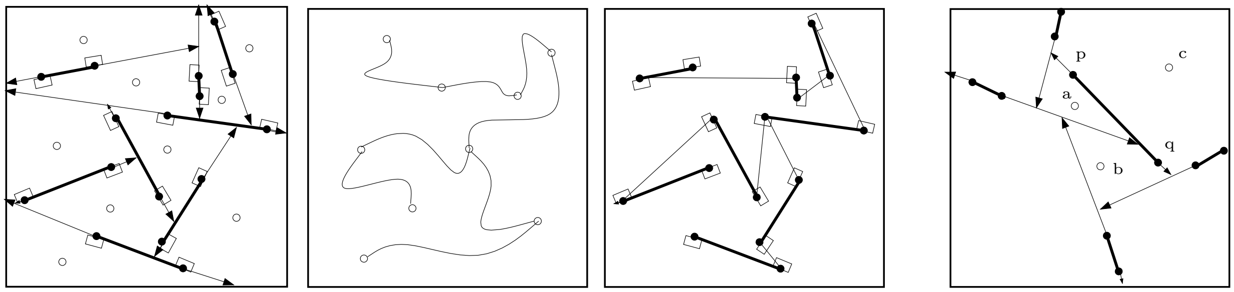
Pointed PSLGs are tightly connected to minimum pseudo-triangulations, which have numerous applications in motion planning [10], kinetic data structures [8], collision detection [1], and guarding [9]. Streinu [10] showed that a minimum pseudo-triangulation of V is a pointed PSLG on the vertex set V with a maximal number of edges. As opposed to triangulations, there is always a bounded degree pseudo-triangulation of a set of points in the plane [7]. A bounded degree pointed encompassing tree for disjoint segments leads to a bounded degree pointed encompassing pseudo-triangulation, due to a result of Aichholzer et al. [2].

In this paper, we improve all previous results on encompassing trees of n segments and give a simple algorithm to construct a pointed binary encompassing tree in optimal $O(n \log n)$ time. Moreover, for every vertex of the tree all incident edges lie on one side of the line through the incident input segment.

Theorem 1 *For a set S of n disjoint line segments in the plane, we can build in $O(n \log n)$ time a binary encompassing tree such that for every segment endpoint p of every input segment pq the edges incident to p lie in a halfplane bounded by the line through pq.*

Tunnel Graphs. The free space around the segments can be partitioned into $n + 1$ convex cells[1] by the following well known partitioning algorithm: For every segment endpoint p of every input segment pq, extend pq beyond p until it hits another input segment, a previously drawn extension, or to infinity.

[1] For simplicity, we assume that no three segment endpoints are collinear.



(a) Partition. (b) Tunnel Graph. (c) Encompassing Tree. (d) Disconnected.

Fig. 1: An example for a partition with an assignment (a), the corresponding tunnel graph (b), and the resulting tree (c). A partition for which no assignment gives a connected tunnel graph (d).

Consider a set of segments S and a convex partition $P(S)$ obtained by the above algorithm. Let us assign every p to an incident cell $\tau(p)$ of the partition. A key tool in our proof is the *tunnel* graph $T(S, P(S), \tau)$ of $P(S)$ and the assignment τ, defined as follows: The nodes of T correspond to the convex cells of $P(S)$, two nodes a and b are connected by an edge iff there is a segment $pq \in S$ such that $\tau(p) = a$ and $\tau(q) = b$. It is easy to see that the tunnel graph is planar. As T has $n+1$ nodes and n edges, it is connected iff it is a tree.

Theorem 2 *For any set S of n disjoint line segments, we can construct in $O(n \log n)$ time a convex partition $P(S)$ and an assignment τ such that the tunnel graph $T(S, P(S), \tau)$ is a tree.*

We note that Theorem 2 does not hold for every partition: Fig. 1(d) shows 7 disjoint line segments and a convex partition such that there is no assignment for which the tunnel graph is connected. The proof of Theorem 2 can be found in the full version of this paper. Our main result follows from Theorem 2.

Proof of Theorem 1. Consider a partition $P(S)$ and an assignment τ provided by Theorem 2. In each cell connect the segment endpoints assigned to it by a simple path.

The resulting graph is clearly a PSLG that encompasses the input segments. The maximal degree is three because we add at most two new edges at every segment endpoint. It remains to prove connectivity. Let p and r be two segment endpoints. We know that the tunnel graph is connected, so there is an alternating sequence of cells and segments $(a_1 = \tau(p), p_1q_1, a_2, \ldots, p_{k-1}q_{k-1}, a_k = \tau(r))$ such that $\tau(p_i) = a_i$ and $\tau(q_i) = a_{i+1}$, for every i. As all segment endpoints assigned to the same cell are connected, this path corresponds to a path in the encompassing graph. □

References

[1] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang, Deformable free space tilings for kinetic collision detection, in *Proc. 4th WAFR*, 2001, 83–96.

[2] O. Aichholzer, M. Hoffmann, B. Speckmann, and Cs. D. Tóth, Degree bounds for constrained pseudo-triangulations, in: *Proc. 15th CCCG*, 2003, pp. 155–158.

[3] P. Bose, M. E. Houle, and G.T. Toussaint, Every set of disjoint line segments admits a binary tree, *Discrete Comput Geom.* **26** (2001), 387–410.

[4] P. Bose and G. T. Toussaint, Growing a tree from its branches, *J. Algorithms* **19** (1995), 86–103.

[5] M. Hoffmann, B. Speckmann, and Cs. D. Tóth, Pointed binary encompassing trees, in *Proc. 9th SWAT*, 2004.

[6] M. Hoffmann and Cs. D. Tóth, Alternating paths through disjoint line segments, *Inf. Proc. Letts.* **87** (2003), 287–294.

[7] L. Kettner, D. Kirkpatrick, A. Mantler, J. Snoeyink, B. Speckmann, and F. Takeuchi, Tight degree bounds for pseudo-triangulations of points, *Comput. Geom. Theory Appl.* **25** (2003), 1–12.

[8] D. Kirkpatrick and B. Speckmann, Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons, in *Proc. 18th SoCG*, 2002, pp. 179–188.

[9] B. Speckmann and Cs. D. Tóth, Allocating vertex π-guards in simple polygons, *14th SODA*, 2003, pp. 109–118.

[10] I. Streinu, A combinatorial approach to planar non-colliding robot arm motion planning, *41st FOCS*, 2000, pp. 443–453.

Hamiltonian Cycles in Sparse Vertex-Adjacency Duals

Perouz Taslakian and Godfried Toussaint
School of Computer Science
McGill University
Montréal, Québec, Canada
{perouz, godfried}@cs.mcgill.ca

1 Introduction

In computer graphics, a common way of representing objects is with their triangulations. The performance of operations executed on these objects depends highly on how their surfaces are triangulated and how these triangles are transmitted to the processing engine. Thus, to speed up many operations, such as rendering or compression [9, 10], it is desirable that triangles be arranged such that their adjacency information is preserved.

In this paper we present a *sparse vertex-adjacency dual* of a polygon triangulation, which is a graph that preserves the vertex-adjacency information of the triangles and contains a Hamiltonian cycle. The size of this graph is linear in the number of polygonal vertices.

Effort has been made to design algorithms that produce Hamiltonian triangulations, where the dual graph of the triangulation is a path. In [1], Arkin et al. show that any set of n points has a Hamiltonian triangulation and describe two algorithms which construct such triangulations. They also show that the problem of determining whether a polygon (with holes) has a Hamiltonian triangulation is NP-complete. In the same paper, a *sequential triangulation of a set of points* is defined to be a Hamiltonian triangulation whose dual graph contains a Hamiltonian path, and it is proved that such triangulations do not always exist for any given set of points.

Hamiltonian properties of general triangulations have been studied extensively. Various results that construct a Hamiltonian cycle in a given triangulation can be classified based on the model considered. In one model, the given triangulation is allowed to be modified by adding new vertices or *Steiner* points. In [7], Gopi and Eppstein present an algorithm for constructing a Hamiltonian cycle in a given triangulation by inserting new vertices within existing triangles.

In the second model, the input triangulation cannot be modified. In this case, the problem is that of arranging adjacent triangles in some order such that the resulting graph contains a Hamiltonian cycle. An important property here is how *adjacency* is defined. In the dual graph of a triangulation, adjacency is defined as *edge-adjacency* where two triangles are adjacent when they share an edge. Unfortunately, it is not always possible to find Hamiltonian cycles in the dual graph.

Hamiltonian cycles in triangulations are studied when adjacency is defined as *vertex-adjacency*, where two triangles are considered to be adjacent if they share at least one vertex. In [5], a triangulation is represented with a *vertex-facet incidence graph* which has a vertex f for each facet (triangle), a vertex v for each triangle-vertex and an edge (v, f) whenever v is a vertex of triangle f. A *facet cycle* is defined by a walk $(v_0, f_1, v_1, f_2, v_2, \ldots, f_k, v_k, f_0, v_0)$ where no arc is repeated and that includes each facet vertex exactly once, but may repeat triangle-vertices. The authors prove that any triangulation has a facet cycle if it is not a *checkered* polygonal triangulation - that is if it does not have a 2-coloring of the triangles such that *every* white triangle is adjacent to three black ones.

A similar result under the same facet cycle model is found in [2]. Here, Bartholdi III and Goldsman refer to general triangulation as *Triangulated Irregular Networks (TINs)*. The authors describe an algorithm to construct a cycle in a 2-adjacent TIN (a triangulation in which each triangle shares an edge with at least two other triangles). Their algorithm runs in $O(n^2)$ time in the worst case.

In [4], Chen, Grigni and Papadimitriou define the *map graph* of a planar subdivision P (or a *map*) to be a graph G where the vertices of G correspond to the faces of P and two vertices u and v are adjacent if their corresponding faces in P share any point on their boundary. This characterization is equivalent to the dual graph of a triangulation in which two vertices u and v of the dual are connected by an edge whenever the triangles corresponding to u and v share a triangular edge or a triangular vertex. Chen et al. study sparsity and coloring of map graphs.

Bartholdi III and Goldsman [3] introduce the same concept of a map graph that they call the *vertex-adjacency dual* of a general triangulation. The authors show that the vertex-adjacency dual contains a Hamiltonian cycle, and they describe a linear time algorithm to construct such a cycle. Here we note that the model described in [3] is a variation of the facet-cycle model described in [5]: In the facet cycle model a continuous walk enters every triangle from a vertex v and leaves from a *different* vertex u. In the vertex-adjacency dual model, u and v are allowed to be the same vertex. The vertex-adjacency dual described in [3] can have $O(n^2)$ edges in the worst case. Here we consider linear size subgraphs of the vertex-adjacency dual that still contain Hamiltonian cycles, and that may be computed in linear time. We call such graphs *sparse vertex-adjacency duals*.

2 Constructing a Sparse Vertex-Adjacency Dual

Here we illustrate an approach with the simple case of sequential triangulations [6].

Let P be a simple polygon with n vertices and let T_P be a sequential triangulation of P. We will refer to the vertices of P as *polygonal vertices* and to the vertices of the dual graph D of T_P as *dual vertices*.

To construct the sparse vertex-adjacency dual of T_P, first construct its dual graph D, which in this case is a path. Then, for every polygonal vertex v_P, if v_P is shared by $k > 2$ consecutive triangles $t_1, t_2, \ldots, t_k$, insert an edge between the first and last triangles t_1 and t_k. The resulting graph G is a sparse vertex-adjacency dual. To show that it contains a Hamiltonian cycle, consider the following. In the dual graph D of T_P every consecutive vertices u, v and w are vertex-adjacent. Thus, connecting every u to w in the sparse vertex-adjacency dual is equivalent to connecting the vertices which are at distance 2 apart. The resulting graph is known as the *square* of D. In [8], it is shown that if removing the leaves of a tree T produces a path, then the square of T is Hamiltonian. In our case, our tree is the dual graph D, which will still be a path if we remove its leaves. Thus, from the results in [8] we can conclude that our construction for a sequential triangulation yields a Hamiltonian cycle (figure 1).

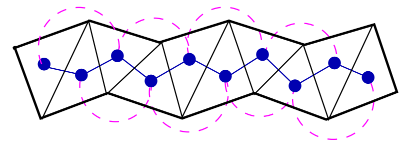


Figure 1: The sparse vertex-dual of a sequential triangulation is equivalent to the square D^2 of the dual graph D.

We can show that for any serpentine triangulation, the above construction will produce a graph that contains a Hamiltonian cycle. For general polygonal triangulations however, this construction needs a slight modification in order to contain such a cycle while preserving adjacency information of the triangles.

References

[1] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. *Visual Computing*, 12(9):429–444, 1996.

[2] J. J. Bartholdi III and P. Goldsman. Multiresolution indexing of triangulated irregular networks. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):1–12, 2004.

[3] J. J. Bartholdi III and P. Goldsman. The vertex-adjacency dual of a triangulated irregular network has a hamiltonian cycle. *Operations Research Letters*, 32:304–308, 2004.

[4] Z. Chen, M. Grigni, and C. H. Papadimitriou. Map graphs. *Journal of the ACM (JACM)*, 49(2):127–138, 2002.

[5] E. D. Demaine, D. Eppstein, J. Erickson, G. W. Hart, and J. O'Rourke. Vertex-unfoldings of simplicial manifolds. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 237–243, Barcelona, Spain, 2002.

[6] R. Flatland. On sequential triangulations of simple polygons. In *Proceedings of the* 16^{th} *Canadian Conference on Computational Geometry*, pages 112–115, 1996.

[7] M. Gopi and D. Eppstein. Single-strip triangulation of manifolds with arbitrary topology. In *Computer Graphics Forum (EUROGRAPHICS)*, volume 23, 2004.

[8] F. Harary and A. Schwenk. Trees with hamiltonian square. *Mathematika*, 18:138–140, 1971.

[9] M. Isenburg. Triangle strip compression. In *Proceedings of Graphics Interface 2000*, pages 197–204, Montreal, Quebec, Canada, May 2000.

[10] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

Trees on Tracks*

Justin Cappos and Stephen Kobourov

Department of Computer Science
University of Arizona
{justin,kobourov}@cs.arizona.edu

1. Introduction

Simultaneous embedding of planar graphs is related to the problems of graph thickness and geometric thickness. Techniques for simultaneous embedding of cycles have been used to show that the degree-4 graphs have geometric thickness at most two [3]. Simultaneous embedding techniques are also useful in visualization of graphs that evolve through time.

The notion of simultaneous embedding is related to that of graph thickness. Two vertex-labeled planar graphs on n vertices can be simultaneously embedded if there exist a labeled point set of size n such that each of the graphs can be realized on that point set (using the vertex-point mapping defined by the labels) with straight-line edge segments and without crossings. For example, any two paths can be simultaneously embedded, while there exist pairs of outerplanar graphs that do not have a simultaneous embedding.

In this paper we present new results about embedding labeled trees and outerplanar graphs on labeled tracks, as well as related results on simultaneous embedding of tree-path pairs. In particular, we show that labeled trees cannot be embedded on labeled parallel straight-line tracks, but they can be embedded on labeled concentric circular tracks; see Fig. 1. The results generalize to outerplanar graphs as well. We also show that tree-path pairs can be simultaneously embedded when edges of the path are represented by circular arcs. Finally, we show how to embed a straight-line tree and a path with $O(\log n)$-bends per edge, where n is the number of vertices.

1.1. Related Work

The existence of straight-line, crossing-free drawings for a single planar graphs is well known [5]. The existence of simultaneous geometric embeddings for pairs of paths, cycles, and caterpillars is shown in [1]. Counter-examples for pairs of general planar graphs, pairs of outer-planar graphs, and triples of paths are also presented there. It it not known whether tree-tree or tree-path pairs allow simultaneous geometric embeddings. If the straight-line edge condition is relaxed, it is known how to embed tree-path pairs using one bend per tree edge and how to embed tree-tree pairs using at most 3 bends per edge [4].

A related problem is the problem of *graph thickness* [7], defined as the minimum number of planar subgraphs into which the edges of the graph can be partitioned into. *Geometric thickness* is a version of the thickness problem where the edges are required to be straight-line segments [2]. Thus, if two graphs have a simultaneous geometric embedding, then their union has geometric thickness two. Similarly, the union of any two planar graphs has graph thickness two. Simultaneous geometric embedding techniques are used to show that degree-four graphs have geometric thickness two [3].

Simultaneous drawing of multiple graphs is also related to the problem of embedding planar graphs on a fixed set of points in the plane. Several variations of this problem have been studied. If the mapping between the vertices V and the points P is not fixed, then the graph can be drawn without crossings using two bends per edge in polynomial time [6]. However, if the mapping between V and P is fixed, then $O(n)$ bends per edge are necessary to guarantee planarity, where n is the number of vertices in the graph [8].

1.2. Our Contributions

We begin with results on track embeddability. Given a set of labeled parallel lines (tracks) L_i, $1 \leq i \leq n$ and a tree $T = (V, E)$ with n vertices labeled with the numbers 1 though n, it is not always possible to obtain a straight-line crossings-free drawing of T such that vertex v_i is on track L_i; see Fig. 1(a). However, if the tracks are concentric circles, such drawings are always possible and we describe a linear time algorithm for obtaining such drawings; see Fig. 1(b). The algorithm easily generalizes to outerplanar graphs as well. Thus, parallel line tracks do not allow tree or outerplanar embeddings on predetermined tracks, while circular tracks do. Tracks defined by circular arcs, stairs, sin-waves also suffice; see Fig. 2.

Our motivation for the problem of track embeddings comes from two open problems in simultaneous geometric embedding. Formally, in the problem of simultane-

*This work is partially supported by the NSF under grant ACR-0222920 and by ITCDI under grant 003297.

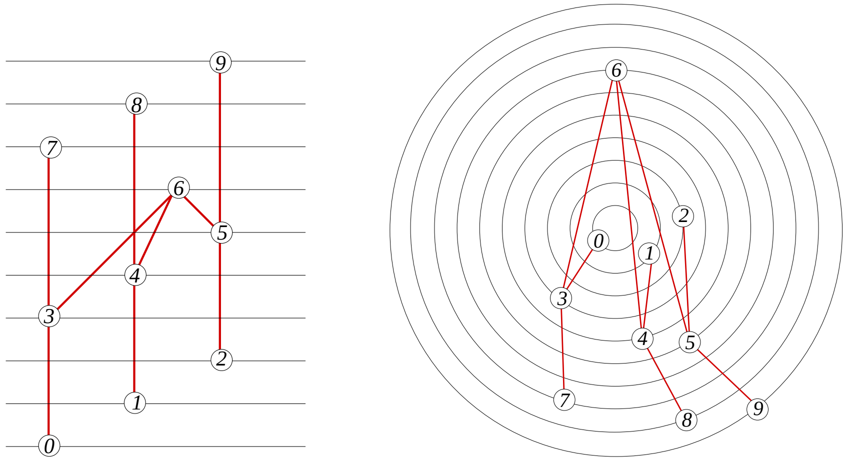


Figure 1: A tree that cannot be drawn on line tracks without crossings but which can be drawn on circular tracks.

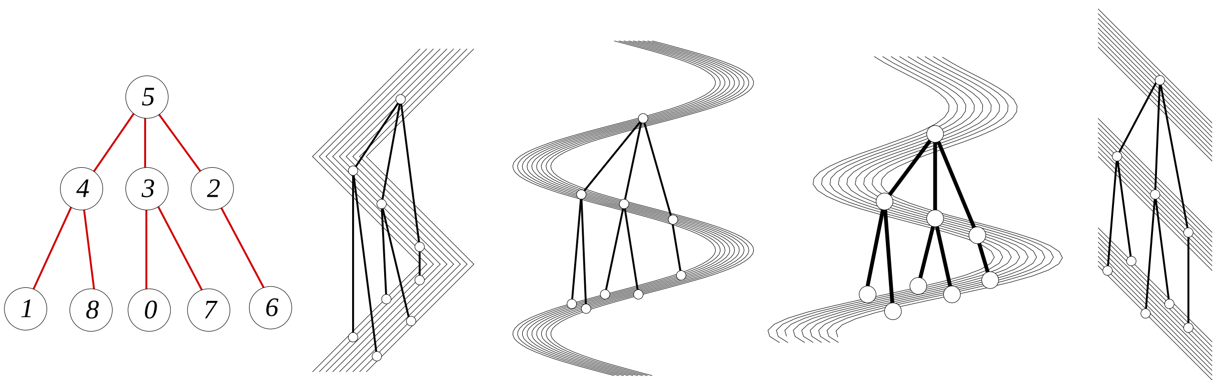


Figure 2: A tree drawn on various staircases: (a) staircase; (b) $\sin(x)$; (c) $x\sin(x)$; (d) $x - \lfloor x \rfloor$.

ous geometric embedding we are given two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ and we would like to find plane straight-line drawings D_1 and D_2 such that for all vertices $v \in V$ the location of the corresponding vertices in D_1 and D_2 is the same (i.e., $D_i(v) = D_j(v)$). While path-path, cycle-cycle, caterpillar-caterpillar pairs can be simultaneously embedded, it is not known whether tree-tree or tree-path pairs have such embeddings.

The circular track layout of trees and outerplanar graphs can be used to obtain simultaneous embeddings of tree-path pairs so that the tree edges are straight-line and crossings-free and the path edges are crossings-free circular-arc segments. Moreover, the staircase layout of trees can be used to obtain simultaneous embeddings of tree-path pairs so that the tree edges are straight-line and crossings-free and the path edges are crossings-free and have at most $\log n$ bends per edge.

2. References

[1] P. Brass, E. Cenek, C. A. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. S. B. Mitchell. On simultaneous graph embedding. In *8th Workshop on Algorithms and Data Structures*, pages 243–255, 2003.

[2] M. B. Dillencourt, D. Eppstein, and D. S. Hirschberg. Geometric thickness of complete graphs. *Journal of Graph Algorithms and Applications*, 4(3):5–17, 2000.

[3] C. A. Duncan, D. Eppstein, and S. G. Kobourov. The geometric thickness of low degree graphs. In *20th Annual ACM-SIAM Symposium on Computational Geometry (SCG)*, pages 340–346, 2004.

[4] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. In *12th Symposium on Graph Drawing (GD)*. To appear in 2004.

[5] I. Fáry. On straight lines representation of planar graphs. *Acta Scientiarum Mathematicarum*, 11:229–233, 1948.

[6] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal of Graph Algorithms and Applications*, 6(1):115–129, 2002.

[7] P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: a survey. *Graphs Combin.*, 14(1):59–73, 1998.

[8] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17:717–728, 2001.

On the Non-Redundancy of Split Offsets in Degree Coding

Martin Isenburg
isenburg@cs.unc.edu

Jack Snoeyink
snoeyink@cs.unc.edu

University of North Carolina at Chapel Hill

Abstract

The connectivity coder by Touma and Gotsman encodes a planar triangulation through a sequence of vertex degrees and occasional "split" symbols that have an associated offset value. We show that the split offsets of the TG coder are not redundant by giving examples of degree sequences that have two different decodings if the split offsets are not specified. Surprisingly, such examples are rare and a large number of encodings remain unique.

1 Introduction

Recent years have seen a number of schemes that compactly encode triangle mesh connectivity by a sequence of symbols that specify how to grow a "compression boundary" enclosing an already encoded region, one triangle at a time. A popular scheme is the Triangle Mesh Compression method by Touma and Gotsman [7], or TG coder for short. For planar triangulations, the TG coder generates a sequence of vertex degrees that usually contains a few "split" symbols with associated offset values.

There has been speculation that it might be possible to modify the TG coder to operate without explicitly storing the offsets values. The Cut-border Machine [3] and Dual-Graph Method [5] explicitly include split offsets, but the otherwise identical Edgebreaker [6] and Face-Fixer [4] schemes avoid them, getting by only with the "end" symbols in the code sequence. However, the TG coder does not store explicit "end" symbols. It maintains more state information on the compression boundary than Edgebreaker or Face Fixer that—together with explicit offsets—makes "end" symbols implicit. But if we omit the offsets we can find sequences with two valid decodings even if we add explicit "end"s to the code sequence.

2 Connectivity coding with the TG Coder

To encode a triangulation, the TG coder [7] grows an encoded region, maintaining one or more compression boundaries into which it includes triangle after triangle. It usually includes the triangle adjacent to the *gate* edge, which advances in clockwise order around the *focus* vertex. However, it immediately includes any triangle that shares two edges with the compression boundary.

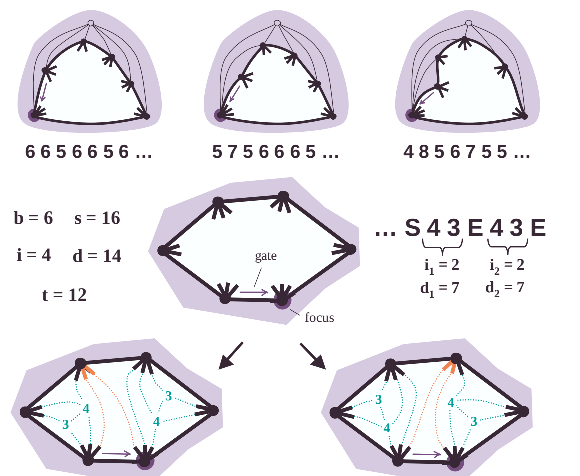


Figure 1: The smallest scenarios where an offset-less encoding with "split" and "end" symbols is not unique occur in triangulations of 11 vertices. The top right example, however, is unique.

The encoder starts with an compression boundary of length two around an arbitrary edge, records the degree of the two initial boundary vertices, and sets their *slot count* to be one less than their degree. Whenever all boundary vertices have a slot count of one or higher, the triangle adjacent to the gate shares only one edge with the compression boundary. Usually the third vertex of the triangle at the gate is a previously unprocessed vertex, and the encoder simply "adds" this vertex as a new boundary vertex and records its degree. Occasionally, the third vertex of this triangle is already on the boundary, and the encoder splits the compression boundary into two loops, temporarily stores one on a stack, and continues encoding on the other. Here the encoder records a split symbol and *offset*, which is the number of slots that are on the boundary part that is stored on the stack, from which it can derive how many slots are clockwise along the boundary between the gate and the split vertex.

In the full paper we consider four offset-less encodings, each with less information about the split operations that occur during the encoding process. The strongest is if we omit the offset, but still record "end" symbols that mark the completion of a boundary loop. Figure 1 illustrates the smallest examples for which this encoding is non-unique: in triangulations with 11 vertices there are two possible ways of splitting the boundary of length 6 that has its 16 slots distributed in this particular configuration and where both resulting boundary parts still enclose two unprocessed vertices of degree 3 and 4. However, there are not always two valid decodings in this scenario. Sometimes the focus is already connected to other vertices of the boundary through previously decoded triangles. This places additional constraints on the possibilities for splitting the boundary. The top-right triangulation from Figure 1, for example, has only one valid decoding.

Theorem. *For the TG coder without split offsets, but with end symbols, there exist two different triangulations that have the same offset-less encoding.*

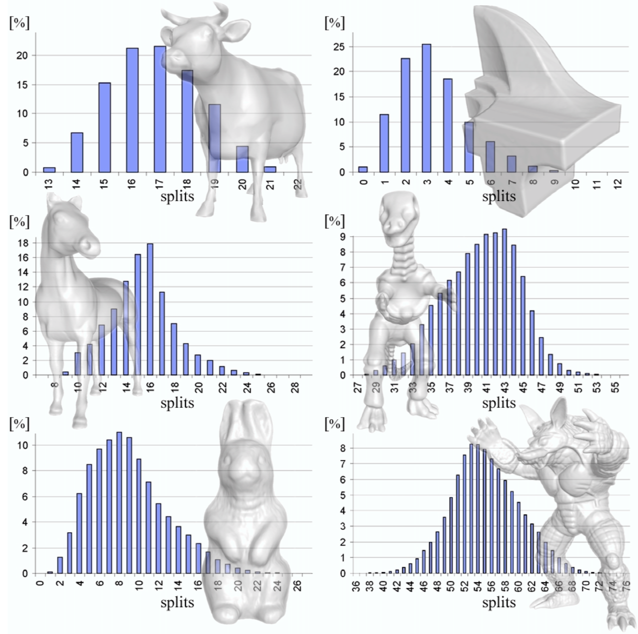
3 Searching for valid decodings

In order to find all valid decodings of an offset-less encoding we search through all possibilities of performing split operations. For each attempt it recursively starts to decode the first boundary part and in case this is successful does the same for the second. Only if both recursions are successful it returns a success, otherwise it tries out the next possibility or returns a failure if there are none left. A few observations help us to immediately eliminate some splits from further consideration.

Initial experiments seemed to indicate that the split offsets of the TG coder might in fact be replaced by "end"s. On our standard set of example meshes the search for split offsets would find the correct answer every run we tried. Table 1 shows that non-unique encoding are surprisingly rare. The full paper has further experiments showing that only a small fraction of random triangulations have non-unique encodings.

4 Closing discussion

There have been attempts to establish a guaranteed bound on the coding costs of the TG coder. However, the infrequently occuring "split" symbols and their offsets made this a difficult task. Our work shows that these split offsets are not completely redundant. There remains the task of determining if any degree-based coder can avoid offsets. Alliez and Desbrun [1] suggested an adaptive traversal heuristic that lowered the number of split operations and the remaining number of "splits" seemed negligibly small. Therefore the authors restricted their worst case analysis to the vertex degrees. But Gotsman [2] has shown that the entropy analysis of Alliez and Desbrun includes many degree distribution that do not correspond to actual triangulations, and that there are fewer valid permutations of degrees than triangulations and that additional information is necessary to distinguish between. So split information does contribute a small but necessary fraction to the encoding.



meshes			splits			non-unique
name	vertices	encodings	min	max	avg	encodings
cow	2,904	17,412	13	22	16.8	0
fandisk	6,475	38,838	0	12	3.3	0
horse	48,485	290,898	7	29	15.4	9
dinosaur	56,194	337,152	27	56	40.4	10
rabbit	67,039	402,222	0	27	9.0	56
armadillo	172,974	1,037,832	36	76	55.2	146

Table 1: We show several meshes with their histograms of splits. The table lists numbers of vertices and encodings, split statistics, and number of non-unique encodings for each mesh.

References

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Proceedings of Eurographics'01*, pages 480–489, 2001.

[2] C. Gotsman. On the optimality of valence-based connectivity coding. *Computer Graphics Forum*, 22(1):99–102, 2003.

[3] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *Proceedings of SIGGRAPH'98*, pages 133–140, 1998.

[4] M. Isenburg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. In *Proceedings of SIGGRAPH'00*, pages 263–270, 2000.

[5] J. Li and C. C. Kuo. A dual graph approach to 3D triangular mesh compression. In *Proceedings of ICIP'98*, pages 891–894, 1998.

[6] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transanctions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[7] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface'98*, pages 26–34, 1998.

Fast almost-linear-sized nets for boxes in the plane

Hervé Brönnimann*
Jonathan Lenchner
CIS, Polytechnic University, Six metrotech, Brooklyn, NY 11201, USA.

1 Introduction

Let $\mathcal{B}$ be any set of n axis-aligned boxes in $\mathbb{R}^d$, $d \geq 1$. For any point p, we define the subset $\mathcal{B}_p$ of $\mathcal{B}$ as $\mathcal{B}_p = \{B \in \mathcal{B} : p \in B\}$. A box B in $\mathcal{B}_p$ is said to be *stabbed* by p. A subset $\mathcal{N} \subseteq \mathcal{B}$ is a $(1/c)$-*net* for $\mathcal{B}$ if $\mathcal{N}_p \neq \emptyset$ for any $p \in \mathbb{R}^d$ such that $|\mathcal{B}_p| \leq n/c$. The number of distinct subsets $\mathcal{B}_p$ is $O((2n)^d)$, so the set system described above has so-called finite VC-dimension d. This ensures that there always exists $(1/c)$-nets of size $O(dc\log(dc))$, and that they can be found in time $O_d(n)c^d$, using quite general machinery (see for example the books by Matoušek [3] or by Pach and Agarwal [7]). For some set systems, such as halfplanes in $\mathbb{R}^2$ and translates of a simple closed polygon, it was shown that there exist $(1/c)$-nets of size $O(c)$ [4]. This was extended to halfspaces in $\mathbb{R}^3$ and pseudo-disks[1] in $\mathbb{R}^2$ [2].
In this paper, we investigate a fast, $O(n \log c)$-time construction of $(1/c)$-nets of size $O(c)$ for any value $1 < c \leq n$ and $d = 2$. Until right before JCDCG, I thought I could prove the following (which unfortunately remains a conjecture):

Conjecture 1 *Let $\mathcal{B}$ be a set of n axis-aligned boxes in $\mathbb{R}^2$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O(c)$.*

We can prove this result and also provide algorithms that run in time $O(n \log c)$ only for special cases: segments on the real line (the one-dimensional case), quadrants of the form $(-\infty, x] \times (-\infty, y]$ in $\mathbb{R}^2$, and unbounded boxes of the form $[x_1, x_2] \times (-\infty, y]$ (which we call a *skyline*). For the general case of boxes, we can prove a size bound of $O(c \log\log c)$. But the conjecture still stands.

*Research of this author has been supported by NSF CAREER Grant CCR-0133599.

[1]In this context, a collection of shapes is called a pseudo-disk set system if given any three points, there is at most one shape in the collection whose boundary passes through these three points.

2 Intervals on the line

We first prove that it is easy to find small nets for intervals on the line, the one-dimensional case of the problem above.

Theorem 2 *Let $\mathcal{B}$ be a set of n intervals on the real line $\mathbb{R}$ and c be any parameter $1 < c \leq n$. There exists a subset $\mathcal{N}$ of at most $2\lceil c-1 \rceil$ boxes in $\mathcal{B}$ that is a $(1/c)$-net for $\mathcal{B}$. Such a set can be found in $O(n + n\log c)$ time.*

3 Rectangles in the plane

We generalize the method of the previous paragraph to the plane. We begin with the easier problem when all the boxes are south-west quadrants, i.e. they contain the point $(-\infty, -\infty)$.

Theorem 3 *Let $\mathcal{B}$ be a set of n quadrants with the same orientation in $\mathbb{R}^2$, and c be any parameter $1 < c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $\lceil c-1 \rceil$. Such a net can be found in time $O(n \log c)$.*

Let us add one more side to the quadrants: a *skyline* is a set of boxes that all intersect a common line. We are only interested in what happens on one side of that line, so we can consider unbounded boxes of the form $[x_1, x_2] \times (-\infty, y]$. We can extend the previous result to a skyline.

Theorem 4 *Let $\mathcal{B}$ be a set of n axis-aligned boxes, all unbounded in some common direction, and c be any parameter $1 < c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size at most $\lceil 2c-1 \rceil$. Such a net can be found in time $O(n \log c)$.*

Using this result, we can now solve the general problem for boxes. Unfortunately, we cannot solve the conjecture, but we can prove:

Theorem 5 *Let $\mathcal{B}$ be a set of n axis-aligned boxes in $\mathbb{R}^2$ and c be any parameter $1 < c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O(c \log \log c)$. Such a net can be found in time $O(n \log c)$.*

4 k-oriented objects

A natural generalization of boxes in $\mathbb{R}^2$ is that of a k-oriented convex polygon [6], which is simply a convex polygon whose sides are constrained to be parallel to a set of k fixed directions ($k = 2$ for boxes). Our proof extends there as well. In fact, we suspect that any result for boxes would extend to k-oriented polygons where the constants in the $O()$ notations become functions of k. But we have no proof of such a general statement.

5 Orthants in higher dimension

As shown in the previous section, the key problem is that for the generalized orthants, which we call orthants for short. We are now interested in this problem for any dimension. We first prove that finding nets for orthants does indeed help for boxes.

Theorem 6 *Assume that there exists ε-nets of size $s(\varepsilon)$ for any set of orthants in $\mathbb{R}^d$, and that $s()$ is non-decreasing and has polynomial growth (which implies $s(O(x)) = O(s(x))$). Let $\mathcal{B}$ be a set of n boxes in $\mathbb{R}^d$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O_d(s(1/c))$. Such a net can be found in time $O(n \log c)$.*

Now we show how small nets we can find efficiently for orthants. The bound is far from optimal for dimensions greater than 2, as nets of size $O_d(c \log c)$ exist but take much longer to compute (see the introduction).

Theorem 7 *Let $\mathcal{B}$ be a set of n orthants with the same orientation in $\mathbb{R}^d$ and c be any parameter $1 \leq c \leq n$. Then there exists a $(1/c)$-net $\mathcal{N}$ for $\mathcal{B}$ of size $O_d(c^{d-1})$. Such a net can be found in time $O(n \log c)$.*

For points and halfspaces in $\mathbb{R}^d$, Matoušek, Seidel, and Welzl [4] have shown that there exist ε-nets of size $O(1/\varepsilon)$. They also show that it suffices to restrict to points in convex position, albeit by having nets bigger by a factor of d. We prove an analog result for orthants, without the blowup factor. The analogue of convex position for orthants is maximal position, as defined in [8].

Lemma 8 *Suppose there exists a ε-net of size $s(\varepsilon)$ for any set of orthants in $\mathbb{R}^d$ in maximal position. Then there exists an ε-net for any set of orthants in $\mathbb{R}^d$ of size $s(\varepsilon)$.*

6 Conclusion

This shows another set system where the general bound $O(c \log c)$ for a $(1/c)$-net could be improved to $O(c)$, and more efficient algorithms can be found. Komlos, Pach and Woeginger [2] have shown that there exist set systems for which $(1/c)$-nets must have size $\Omega(c \log c)$.

This also poses the analog problem of finding good approximations, in the sense that not only does p hit few boxes if it misses $\mathcal{N}$, but the number of hits in $\mathcal{N}$ reflects the number of hits in $\mathcal{B}$ (scaled by $|\mathcal{N}|/|\mathcal{B}|$). The approach above seems to collapse because nothing guarantees the representativity of $\mathcal{N}$.

References

[1] J. Pach and P.K. Agarwal. *Combinatorial Geometry.* J. Wiley, New York, 1995.

[2] J. Komlòs, J. Pach, G.J. Woeginger. Almost Tight Bounds for epsilon-Nets. *Discrete & Computational Geometry* 7:163–173, 1992.

[3] J. matoušek. *Lectures on Discrete Geometry.* Springer, Berlin, 2002.

[4] J. Matoušek, R. Seidel and E. Welzl. How to net a lot with little: small ε-nets for disks and halfspaces. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, p.16-22, June 07-09, 1990, Berkeley, California.

[5] Frank Nielsen, Fast Stabbing of Boxes in High Dimensions. *Theoretical Computer Science*, Elsevier Science, Volume 246, Issue 1-2, 2000.

[6] F. Nielsen. On Point Covers of c-Oriented Polygons. *Theoretical Computer Science*, Elsevier Science, Volume 265, Issue 1-2, pp. 17-29, 2001.

[7] J. Pach and P.K. Agarwal. *Combinatorial Geometry.* J. Wiley, New York, 1995.

[8] F. Preparata and M.I. Shamos. *Computational Geometry.* Springer, New York, 1985.

The Metric TSP and the Sum of its Marginal Values

Moshe Dror*
Yusin Lee†
James B. Orlin ‡

August 31, 2004

Abstract

This paper examines the relation between the length of an optimal Traveling Salesman tour and the sum of its nodes' marginal values (a node's marginal value is the difference between the length of an optimal TSP tour over a given node set and the length of an optimal TSP tour over the node set minus the node). To our knowledge, this problem has not been studied previously. We find that in metric spaces $L_1, L_{4/3}, L_2, L_4, L_\infty$, the event in which the sum of TSP marginal values is greater than the length of the optimal tour is very rare. We present a number of cases for which the sum of marginal values is never greater that the optimal tour. We establish a worst case ratio of 2 for any metric TSP. In addition, for 6 node TSPs we determine the worst ratio for L_1, L_∞ norms, triangular inequality, and symmetric distance, of 10/9, 10/9, 1.2, and 1.5 respectively, by solving the appropriate mixed integer programming problems.

*MIS Department, College of Business and Public Administration, University of Arizona, Tucson, Arizona 85721, and Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139, U.S.A.

†Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139, U.S.A. and Department of Civil Engineering, National Cheng Kung University, Tainan, Taiwan.

‡Sloan School of Management, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts 02139, U.S.A.

Optimal Area Triangulation with Angular Constraints

J. Mark Keil, Tzvetalin S. Vassilev
Department of Computer Science, University of Saskatchewan
57 Campus Drive, Saskatoon, Saskatchewan, S7N 5A9 Canada
e-mails: keil@cs.usask.ca, tsv552@mail.usask.ca

18 October 2004

Extended Abstract

1 Introduction

We study triangulations of planar sets of points. The problem of interest is to optimize the area of the individual triangles in the triangulation, or to find the MinMax and MaxMin area triangulations of the given point set. These two problems admit polynomial time algorithms in the case when the point set is a convex polygon. In the general case, the problems are of unknown complexity. The problem of finding the MinMax area triangulation is mentioned as a hard open problem in Edelsbrunner's book [2]. In this paper we discuss an approach for approximating these two optimal area triangulations with a triangulation that has angular restrictions imposed. This results in a subcubic algorithm for finding the approximating triangulation. The approximation ratio depends on angular parameters, analytical results on this are shown.

2 Angular restrictions and forbidden zones

Suppose the optimal area triangulation (either MaxMin or MinMax area) contains small angles, which is known to be unsuitable for practical purposes. Denote the smallest angle in the optimal triangulation (which we don't know) as β. We call triangulation that has all of its angles larger than β a β -triangulation. We want to construct an α-triangulation, which is "fatter" ($\alpha > \beta$), that approximates the optimal with a practical coefficient. Given the fact that all of the angles of the α-triangulation are going to be larger than α, we can define a region surrounding each edge of the triangulation, called forbidden zone. The forbidden zone of an edge is by definition a region that is empty of points of the original point set if the edge is in the triangulation. In these circumstances, the forbidden zone is a polygonal region, recursively defined by adding to the edge isosceles triangles with a base the edge itself, and base angles of α, and continuing this process outwards of the already tiled area infinitely. First four steps are shown in Figure 1. The parameters of the forbidden zone are fully determined by the length of the edge a and the angle α. The forbidden zone entirely contains a trapezoid with the given edge as a base, base angles of 3α and heighth of $(a/2)\tan\alpha$. The zone also entirely contains a circle surrounding each of the endpoints of the edge. Please refer to Figure 2 for an illustration.

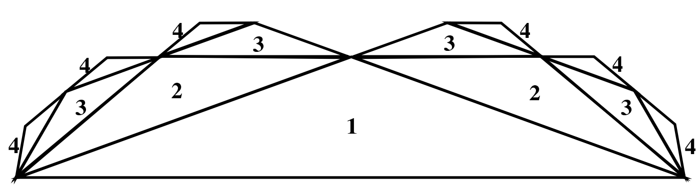


Figure 1: Recursive construction of the forbidden zone

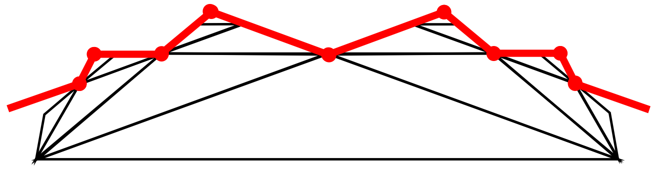


Figure 2: The border of the forbidden zone up to third order

3 Perfect matchings and bounds

To obtain the bounds for approximation factors, we use the perfect matching between the β -triangulation and the α-triangulation, as described in Aichholzer's paper [1]. We study all the possible cases of matched triangles and positions of the points with respect to the forbidden zones. Based on this we derive the following bounds of the approximation factors for the MaxMin area triangulation:

$$f_1 = max\left(\frac{1}{\tan\alpha\tan^2\beta\tan\frac{\beta}{2}}, \frac{1}{\sin\alpha\tan^2\beta}, \frac{2(1+k)(1+2k)}{\tan\alpha}, \frac{k^2}{\sin\alpha}\right)$$

and for the MinMax area triangulation:

$$f_2 = max\left(\frac{1}{\tan\beta\tan^2\alpha\tan\frac{\alpha}{2}}, \frac{1}{\sin\beta\tan^2\alpha}, \frac{1}{2(1-k)(1-2k)\sin\beta\tan\frac{\alpha}{2}}, \frac{1}{k_1^2\sin 2\beta}\right)$$

where k, k_1 and n are the following parameters of the forbidden zone:

$$k = \frac{\tan\alpha}{2\sin 3\alpha} \quad k_1 = \frac{1}{2^n\cos^n\alpha} \quad n = \left\lceil \frac{180^\circ}{2\alpha} - \frac{1}{2} \right\rceil$$

The approximation factor f_1 shows how many times the smallest area triangle in the approximating α -triangulation is smaller than the smallest area triangle in the optimal (MaxMin area) triangulation. Similarly, f_2 gives the ratio of the largest area triangle in the approximating triangulation, compared to the largest area triangle in the optimal (MinMax area) triangulation.

4 Algorithmic results and sample values

Based on the fact that we can compute the optimal 30°-triangulation (if it exists) by modified Klincsek's algorithm, or we can relax Delaunay by area equalizing flips, we achieve a subcubic algorithm that approximates the optimal area triangulations, by the above given factors. The value of α can be chosen from practical considerations. Here are some sample results, summarized in a table:

α	30	30	25	25	20	20	15
β	25	20	20	15	15	10	10
f_1	35.930	74.149	91.807	226.87	290.66	1010.1	1372.0
f_2	24.010	30.716	56.994	77.418	311.28	455.06	7900.1

Table 1: Sample values for f_1 and f_2

References

[1] O. Aichholzer, F. Aurenhammer, S.-W. Cheng, N. Katoh, G. Rote, M. Taschwer, and Y.-F. Xu. Triangulations intersect nicely. *Discrete and Computational Geometry*, 16(4):339–359, 1996.

[2] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, UK, 2001.

Detecting Duplicates Among Similar Bit Vectors

(of course, with geometric applications)

Boris Aronov[1] and John Iacono[2]

Abstract We show how to detect duplicates in a sequence of k n-bit vectors presented as a list of single-bit changes between consecutive vectors, in $O((n+k)\log n)$ time.

Problem We are given a sequence $S = \{v_1, \ldots, v_k\}$ of k n-bit vectors, presented as follows: The first bit vector is all zeros and each subsequent vector v_i is obtained from the previous vector v_{i-1} by flipping a single bit in position b_i, $0 \le b_i < n$. S is represented as $b_2, b_3, \ldots, b_k$. The problem is to detect duplicates in the sequence $v_1, v_2, \ldots, v_k$. More formally, we seek a labeling $S \to \{1, \ldots, k\}$, $v_i \mapsto c_i$, such that $c_i = c_j$ iff $v_i = v_j$.

Solution Without loss of generality in the remainder of this note we assume that n is a power of two. Let T be the perfectly balanced binary tree on n leaves. We number the leaves of T from 0 to $n-1$ and associate each with a bit position. Each interior node x of T is similarly associated with a block $B(x)$ of consecutive bit positions corresponding to the leaves of the subtree rooted at x. For a bit vector v_i, let $v_i(x)$ be its substring in $B(x)$. The idea behind our data structure is simple: each node x has an associated data structure that stores implicitly the set $\bigcup_{i=1}^{k}\{v_i(x)\}$. The data structure stored at node x consists of two arrays D_x and F_x that store the following data:

- $D_x[1, \ldots, d_x]$ contains the sorted set including 1 and all distinct values i, $1 < i \le k$, such that $v_{i-1}(x) \ne v_i(x)$.
- $F_x[1, \ldots, d_x]$ contains integers in the range $1, \ldots, d_x$ with the property that $F_x[i] = F_x[j]$ iff $v_{D_x[i]}(x) = v_{D_x[j]}(x)$.

[1]Research supported in part by NSF ITR Grant CCR-00-81964 and by a grant from US-Israel Binational Science Foundation; part of work has been carried out while visiting Max-Planck-Institut für Informatik. Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn, NY 11201 USA; http://cis.poly.edu/~aronov.

[2]Research supported in part by NSF grant CCF-0430849. Department of Computer and Information Science, Polytechnic University, 5 MetroTech Center, Brooklyn, NY 11201 USA; http://john.poly.edu.

We now complete the description of our algorithm, by explaining how to initialize D_z and F_z for all leaves $z \in T$ and how to compute D_x and F_x from D_l, F_r, D_l, F_r for any internal node x with children l and r. F_{root} describes the desired labeling of S, since D_{root} contains all the numbers $1, \ldots, k$.

If one stores all of the leaves z in an array in numerical order, a linear scan of the sequence $b_2, b_3, \ldots, b_k$ of bit updates allows one to initialize arrays D_z and F_z, for all z. Specifically, we store the current bit vector v_{i-1} explicitly in a bit array $V[0, \ldots, n-1]$. Since $b_i = j$ indicates a bit flip in position $z = j$ (recall that bit positions, and thus leaves are identified with integers $0, \ldots, n-1$), we flip the value of $V[j]$, add j to D_z, and depending on the resulting value of $V[j]$, set the next entry in F_z to zero or one.

Algorithm 1 The pseudocode for computing D_x, F_x from D_l, F_l, D_r, F_r.

```
i ← j ← k ← 1
D_l[d_l + 1] ← D_R[d_r + 1] ← ∞
repeat
    P_x[k] ← (F_r(i), F_l(j), k, 0)
    if D_l[i] < D_r[j] then
        D_x[k] ← D_l[i]; i ← i + 1
    else if D_l[i] = D_r[j] then
        D_x[k] ← D_l[i]; i ← i + 1; j ← j + 1;
    else if D_l[i] > D_r[j] then
        D_x[k] ← D_r[j]; j ← j + 1;
    end if
    k ← k + 1;
until i = d_l + 1 and j = d_r + 1
d_x ← k − 1          ▷ d_x is the length of P_x and D_x
Sort P_x lexicographically on the first two fields,
    by radix sort
for k ← 2 to d_x do
    if P_x[k − 1][1] = P_x[k][1] and P_x[k − 1][2] =
        P_x[k][2] then
        P_x[k][4] ← P_x[k − 1][4]
    else
        P_x[k][4] ← P_x[k − 1][4] + 1
    end if
end for
for k ← 1 to d_x do
    F_x[P_x[k][3]] ← P_x[k][4]
end for
```

Now, we describe, for an internal node x of T with children l and r, how to construct D_x, F_x from arrays D_l, D_r, F_l, F_r; see Algorithm 1. The new sorted array $D_x[1, \ldots, d_x]$ is built by merging the arrays D_l and D_r, eliminating any duplicates, in time $O(d_l + d_r) =$

$O(d_x)$. Simultaneously, we create an auxiliary array $P_x[0, \ldots, d_x]$ that records how the merge step has proceeded; it consists of quadruples of items. Refer to lines 1–13. We clearly have

Lemma 1. *The array D_x as constructed is correct. The first two columns of the array P_x contain pairs of integers in the range $1, \ldots, d_x$ with the property that $(P_x[i][1], P_x[i][2]) = (P_x[j][1], P_x[j][2])$ iff $v_{D_x[i]}(x) = v_{D_x[j]}(x)$. The third column just numbers the rows of P_x consecutively.*

The array P_x contains all the information we need, in a sense, but not in the right order. At this point we radix-sort P_x according to the first two fields, in two passes. As P_x is of size d_x and each of these fields is a positive integer no larger than d_x, this takes $O(d_x)$ time. In lines 16–22, we number the rearranged lines of P_x consecutively, ignoring duplicate pairs in the first two columns. These integers will be used to fill in F_x and are guaranteed to be in the range $1, \ldots, d_x$. This takes time $O(d_x)$. We finally fill the array F_x using the data in P_x, as detailed in lines 23–25. A few minutes of contemplation will convince the reader that the following lemma holds.

Lemma 2. *The array F_x contains integers in the range $1, \ldots, d$ with the property that $F_x[i] = F_x[j]$ iff $v_{D_x[i]}(x) = v_{D_x[j]}(x)$.*

Application Suppose one is interested in computing an arrangement of n simple shapes (such as disks, triangles, or halfplanes) in $\mathbb{R}^2$. There are plenty of algorithms, including deterministic ones, that can solve this problem in $O(n^2 \log n)$ time for a variety of shapes. But what if, in addition to the face structure of the arrangement, one is interested in labeling the faces with the bit vector indicating which of the objects each face belongs to? Clearly, an explicitly stored labeling is too expensive, requiring $\Theta(n^3)$ bits in the worst case. However, traversing the arrangement by an Eulerian path of the face incidence graph allows one to encode the bit vectors using single-bit flips between consecutive vectors along the path. In particular using the algorithm described above, we can detect which faces correspond to identical vectors and thus are contained in identical sets of shapes. The process takes $O(n^2 \log n)$ time. An entirely analogous process can process an arrangement of n objects in any dimension, traversing k cells in $O((n + k) \log n)$ time, provided adjacent cells differ only in a single containment and an adjacency structure encoding local bit differences is available.

Note that the assumption that the first bit vector in the sequence is all zeros can be dropped without changing our algorithm. Also observe that our algorithm can be used with slight modifications to detect duplicates among bit vectors coming from several sequences—one just needs to artificially concatenate the sequences together by using dummy intermediate vectors, if the number of sequences is small. This will result in an additional $O(n \log n)$ cost per concatenation. A less brute-force approach to dealing with multiple sequences which results in a $O(n)$ concatenation cost will be described in the full version of this paper.

A geometric application with two sequences of bit vectors was presented in [1].

Reference

[1] P.K. Agarwal, B. Aronov, V. Koltun, "Efficient Algorithms for Bichromatic Separability," manuscript, 2004. Preliminary version appeared in *Proceedings 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, 2004, pp. 675–683.

Approximation Algorithms for Two Optimal Location Problems in Sensor Networks

Alon Efrat   Sariel Har-Peled   Joseph S. B. Mitchell

Abstract

This paper studies two problems that arise in optimization of sensor networks: First, we devise provable approximation schemes for locating a base station and constructing a network among a set of sensors each of which has a data stream to get to the base station. Subject to power constraints at the sensors, our goal is to locate the base station and establish a network in order to maximize the lifespan of the network.

Second, we study optimal sensor placement problems for quality coverage of given domains cluttered with obstacles. Using line-of-site sensors, the goal is to minimize the number of sensors required in order to have each point "well covered" according to precise criteria (e.g., that each point is seen by two sensors that form at least angle α, or that each point is seen by three sensors that form a triangle containing the point).

1 Introduction

Consider a wireless sensor network with a large number of deployed sensors, each capturing data on a continuous basis. The sensors may be capturing video data, audio data, environmental data, etc. There is a base station that collects all of the data streams from all of the sensors. Each sensor passes data packets along some route in a network, from sensor to sensor, so that all data arrives at the base station. Since each sensor is generally powered by some form of battery, the duration of the sensor node is determined in large part by its power dissipation rate and energy provision. A fundamental issue associated with wireless sensor networks is maximizing their useful lifetime, given their power constraints. This can be significantly affected by the location of the base station as well as the forwarding protocols used (i.e. which sensor forwards packages of which other sensor) in establishing the network. Hou[1] has suggested the use of the length of time until the first sensor exhausts its battery as a definition of the *lifespan* of the system. In the paper, we show how to find a location of the base station such that the lifespan of the system is optimized to within any desired approximation bound. Specifically, we give a method for locating the base station that provably obtains a lifespan of at least $(1-\varepsilon)$ times that of the optimal lifespan, where $\varepsilon > 0$ is any pre-determined fixed value. The algorithm is based on solving $O(n\varepsilon^{-4}\log^2(n/\varepsilon))$ instances of a linear programming problem. It is simple and easy to implement.

Theorem 1.1 *Given a set of sensors $\mathcal{S} = \{s_1, \ldots, s_n\}$ and a parameter $\varepsilon > 0$, one can compute a location of the base station and a transmission scheme such that the network lifespan is at least $(1-\varepsilon)t_{opt}$, where t_{opt} is the lifespan of an optimal transmission scheme for $\mathcal{S}$. This algorithms requires $M = O(n\varepsilon^{-4}\log^2(n/\varepsilon))$ preprocessing time, and needs to solve M instances of linear programming.*

[1]Thomas Hou, personal communication, 2003.

We also study another optimal location problem in sensor networks: How does one choose locations of sensors for line-of-sight coverage of a given region, under the assumption that a point is only "covered" if it is "well seen"? We consider two definitions of "well seen": A point p is well seen (or *robustly covered*) if either (a) there are two sensors that see p, and these sensors are separated by angle at least α with respect to p; or (b) there are three sensors that see p and they form a triangle that contains p. The objective is to minimize the number of sensors to achieve robust coverage. Our results on this problem give efficient approximation algorithms for minimizing the number of sensors.

Theorem 1.2 *Given P and Q as above, an angle α, and a grid Γ of edge-length δ inside P, we can find a set G of sensors in P such that G 2-guards Q at angle $\alpha/2$, and $|G| = O(k_{opt} \log k_{opt})$, where k_{opt} is the cardinality of smallest set of vertices of Γ that 2-guard Q at angle α. The running time of the algorithm is $O(nk_{opt}^4 \log^2 n \log m)$, where m is the number of vertices of $\Gamma \cap P$.*

Theorem 1.3 *Deciding if k sensors within P suffice to triangle-guard Q is NP-hard.*

Theorem 1.4 *Given P and Q as above, and a grid Γ of edge-length δ, we can find a set G of sensors in P, that triangle-guard Q, with $|G| = O(k_{opt} \log k_{opt})$, where k_{opt} is the cardinality of smallest set of vertices of Γ that triangle-guard Q. The running time of the algorithm is $O(nk_{opt}^2 \log^2 n \log m)$, where m is the number of vertices of $\Gamma \cap P$.*

This second problem is a variant of the classical art gallery problem, in which one is to place the fewest sensors ("guards") to see all points of a certain geometric domain. Art gallery problems have been studied extensively; see, e.g., [Kei00, Urr00] for recent surveys. The algorithmic problem of computing a minimum number of guards is known to be NP-hard, even if the input domain, $\mathcal{D}$, is a simple polygon. Thus, efforts have concentrated on approximation algorithms for optimal guarding problems. Recently, researchers [EH02, GL01] have applied set cover methods that exploit finiteness of VC-dimension. In particular, Efrat and Har-Peled [EH02] obtain an $O(\log k^*)$-approximation algorithm for guarding a polygon with vertex guards, using time $O(n(k^*)^2 \log^4 n)$, where k^* is the optimal number of vertex guards. Cheong et al. [CEH04] have recently shown how to compute k guards in order to optimize (approximately) the total area seen by the guards. The triangle-guarding coverage problem we study is related to recent work of Smith and Evans [SE03].

References

[CEH04] O. Cheong, A. Efrat, and S. Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 1091–1100, 2004.

[EH02] A. Efrat and S. Har-Peled. Locating guards in art galleries. *2nd IFIP Internat. Conf. Theo. Comp. Sci.*, pages 181–192, 2002.

[GL01] H. González-Banos and J.-C. Latombe. A randomized art-gallery algorithm for sensor placement. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 232–240, 2001.

[Kei00] J. M. Keil. Polygon decomposition. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 491–518. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[SE03] J. Smith and W. Evans. Triangle guarding. In *Proc. 15th Canad. Conf. Comput. Geom.*, pages 76–80, 2003.

[Urr00] J. Urrutia. Art gallery and illumination problems. In J. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.

Farthest Point from Line Segment Queries*

Ovidiu Daescu† and Robert Serfling ‡

We discuss farthest point from line segment queries in $\mathbb{R}^3$ and related problems. The solution for answering farthest point from line segment queries relies on a solution for the **Farthest Point From Line Problem (FPFL):** Preprocess a set P of n points in $\mathbb{R}^d$, $d \geq 3$, such that for a query line L one can efficiently report the farthest point of P from L. We use $O^*(\cdot)$ to hide $O(\log^{O(1)} n)$ factors.

Theorem 1 *A set P of n points in $\mathbb{R}^d$, $d \geq 3$ a constant, can be preprocessed with $O(s \cdot \log^{O(1)} n)$ space and time such that for a query line L the farthest point of P from L can be found in $O(n \log n / s^{1/\lfloor f(d)/2 \rfloor})$ time, where $n \leq s \leq n^{f(d)/2}$.*

Lemma 1 *Let P be a set of n points in $\mathbb{R}^3$ and let $(q, \mathcal{L}_q, \Delta)$ be a triplet with q a point and $\mathcal{L}_q$ a line through q such that all points in P are within distance Δ from $\mathcal{L}_q$. Let h_q denote the plane orthogonal to $\mathcal{L}_q$ at q. If there is a $p \in P$ such that $|qp| > \Delta$, then the point of P farthest from q in the halfspace defined by h_q and containing p is a vertex of the convex hull of P.*

Proof. Let $P = \{p_1, p_2, \ldots, p_n\}$. Let $CH(P)$ be the convex hull of P, let $C = CH(P) \bigcap h_q$, and consider the part $CH(P, q)$ of $CH(P)$ in one of the two halfspaces defined by h_q. For each tetrahedron $qp_ip_kp_j$, with $p_i, p_k, p_j \in CH(P, q)$, one of p_i, p_k or p_j is the farthest point to q. For each tetrahedron of the form qp_ip_ja, with $p_i, p_j \in CH(P, q)$ and a being a vertex of the boundary ∂C of C, let p_k be the farthest point to q within that tetrahedron (see Fig. 1(a)). Let p'_k be the intersection of the line qp_k with the triangle p_ip_ja. Let $c = ap'_k \bigcap p_ip_j$. Observing that $|qa| \leq \Delta$ and $|\overline{qc}| \leq max\{|\overline{qp_i}|, |\overline{qp_j}|\}$, we have that the farthest point to q satisfying the condition in the lemma could only be one of p_i or p_j. For each tetrahedron of the form qp_iab, with $p_i \in CH(P, q)$ and a, b vertices of ∂C, let p_k be the farthest point to q within that tetrahedron (see Fig. 1(b)). Let p'_k be the intersection of the line qp_k with the triangle p_iab. Let $c = p_ip'_k \bigcap ab$. Clearly, c is within distance Δ from q. Thus, by a similar argument as above on triangle qp_ic, the farthest point from q could only be p_i. □

Our solution for computing farthest points from line segments also uses simplicial partitions. A *simplicial partition* $\Psi(P) = \{(P_1, t_1), (P_2, t_2), \ldots, (P_r, t_r)\}$ for a set P of n points in $\mathbb{R}^3$ is a collection of pairs, where the P_i's are disjoint subsets of P whose union is P, and t_i is a tetrahedron containing P_i, for $i = 1, 2, \ldots, r$. For a given simplicial partition $\Psi(P)$, the *crossing number* of a plane h is the number of tetrahedrons of $\Psi(P)$ that h intersects. The crossing number of $\Psi(P)$ is the maximum crossing number over all possible planes. We say that a simplicial partition $\Psi(P)$ is *fine* if $|P_i| \leq 2n/r$, for every $1 \leq i \leq r$. Matousek [2] proved the following result.

*This research was partially supported by NSF grant CCF-0430366.

†Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080.

‡Department of Mathematical Sciences, University of Texas at Dallas, Richardson, TX 75080.

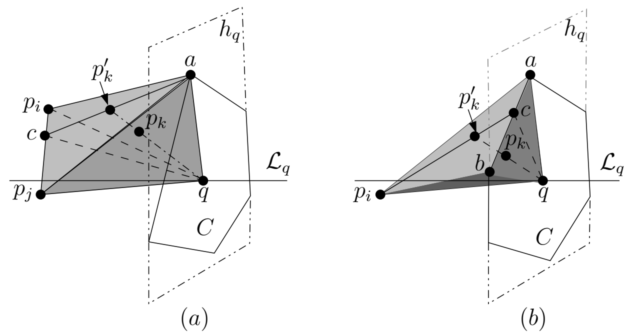


Figure 1: The point p_k farthest from q is a vertex of $CH(P,q)$ if $|qp_k| > \Delta$.

Theorem 2 *[2] For any given set P of n d-dimensional points, $d \geq 2$, and a parameter r, $1 \leq r \leq n/2$, a fine simplicial partition of size r and crossing number $O(r^{1-1/d})$ exists. If r is a constant such a simplicial partition can be constructed in $O(n)$ time with $O(n)$ space.*

A simplicial partition can be used to construct an efficient data structure, called a *partition tree*: the root of the partition tree has r children, each associated with a simplex t_i in a simplicial partition $\Psi(P)$, as well as any secondary information related to the enclosed point set, and being the root of a recursively defined partition tree on the point set that belongs to this node.

Theorem 3 *Given a set P of n points in $\mathbb{R}^3$, in $o(n^{4/3})$ time one can construct a data structure of size $o(n^{4/3})$ such that the farthest point of P from a query line segment s can be reported in $O(n^{2/3+\epsilon})$ time, for arbitrary $\epsilon > 0$.*

Proof. Let L_s be the line supporting s. The farthest point from s is either (i) the farthest point from L_s or (ii) the farthest point $p \in P$ from one of the endpoints of s, with the property that the plane orthogonal to s at that endpoint has s and p on different sides. The point for (i) can be obtained using **FPFL**. This gives us a value Δ such that all points in P are within distance Δ from L_s. The point for (ii) can then be found by performing two queries, each on a data structure associated with $CH(P, q_i)$, where q_i, $i = 1, 2$, is an endpoint of s, and $CH(P, q_i)$ and q_{3-i} are on different sides of the defining plane for $CH(P, q_i)$. Consider the query for $CH(P, q_1)$. The problem is to report the farthest point $p \in P$ to q_1 such that $p \in CH(P, q_1)$ and $|pq_1| > \Delta$, if such a point exists. From Lemma 1, it follows that one should only consider the vertices of $CH(P, q_1)$. We first construct a partition tree based on a fine simplicial partition $\Psi(P)$. Let P_v be the subset of P stored at a node v of the tree, and let t_v be the tetrahedron of $\Psi(P)$ that contains P_v. For each node v, we compute the convex hull CH_v of P_v. We also compute and store a data structure for the subset of P_v corresponding to the vertices of CH_v, that can answer farthest neighbor queries in $\mathbb{R}^3$. In $\mathbb{R}^d$, d a constant, with $O^*(m)$ time preprocessing and $O(m)$ space one can answer farthest neighbor queries in $O(n/m^{1/\lceil d/2 \rceil})$, for any m such that $n \leq m \leq n^{\lceil d/2 \rceil}$ (see [1], page 594). Thus, in $\mathbb{R}^3$, the query time is $O(n/m^{1/2})$. A careful analysis of this data structure shows it requires $O^*(n)$ time preprocessing and $O(n \log n)$ storage. The query time in this data structure is $O(n^{2/3+\epsilon})$, for any constant $\epsilon > 0$. Balancing the space and preprocessing time for **FPFL** to obtain $O(n^{2/3+\epsilon})$ query time, we get $O^*(n^{4/3-4\epsilon})$ preprocessing time and space. □

References

[1] J.E. Goodman and J. O'Rourke. *Handbook of Discrete and Comput. Geometry*, CRC Press, 1997.

[2] J. Matoušek. Efficient partition trees. *Discrete and Computat. Geom.*, 8, 1992.

Computational Geometric Aspects of Musical Rhythm

Godfried Toussaint*
McGill University
Montréal, Québec, Canada

For our purpose a rhythm is represented as a cyclic binary string. Consider the following three 12/8 time ternary rhythms expressed in box-like notation: [x . x . x . x . x . x .], [x . x . x x . x . x . x] and [x . . . x x . . x x x .]. Here "x" denotes the striking of a percussion instrument, and "." denotes a silence. It is intuitively clear that the first rhythm is the most *even* (well spaced) of the three. Traditional rhythms have a tendency to exhibit such properties of evenness. Therefore mathematical measures of evenness find application in the new field of mathematical ethnomusicology [2], [17], where they may help to identify, if not explain, cultural preferences of rhythms in traditional music.

Clough and Duthett [3] introduced the notion of *maximally even sets* with respect to pitch scales represented on a circle. Block and Douthett [1] went further by constructing several mathematical measures of the amount of *evenness* contained in a scale. One of their measures simply adds all the interval arc-lengths (geodesics along the circle) determined by all pairs of pitches in the scale. However, this measure is too coarse to be useful for comparing rhythm timelines such as those studied in [13] and [15]. Using interval chord-lengths (as opposed to geodesic distances), proposed by Block and Douthet [1], yields a more discriminating measure, and is therefore a function that receives more attention. In fact, this problem had been investigated by Fejes Tóth [12] some forty years earlier without the restriction of placing the points on the circular lattice. He showed that the sum of the pairwise distances determined by n points on a circle is maximized when the points are the vertices of a regular n-gon.

One may also examine the *spectrum* of the frequencies with which all the durations are present in a rhythm. In music theory this spectrum is called the *interval vector* (or full-interval vector) [7]. For example, the interval vector for the clave *Son* pattern [x . . x . . x . . . x . x . . .] is given by [0,1,2,2,0,3,2,0].

*This research was partially supported by NSERC and FCAR. e-mail: godfried@cs.mcgill.ca

Examination of such rhythm histograms leads to questions of interest in a variety of fields of enquiry: musicology, geometry, combinatorics, and number theory. For example, David Locke [9] has given musicological explanations for the characterization of the *Gahu* bell pattern, given by [x . . x . . x . . . x . . . x .], as "rhythmically potent", exhibiting a "tricky" quality, creating a "spiralling effect", causing "ambiguity of phrasing" leading to "aural illusions." Comparing the full-interval histogram of the *Gahu* pattern with the histograms of other popular 4/4 time traditional clave-bell rhythms leads to the observation that the *Gahu* is the only pattern that has a histogram with a maximum height of 2, and consisting of a single connected component of occupied histogram cells.

In 1989 Paul Erdős [5] asked whether one could find n points in the plane (no three on a line and no four on a circle) so that for every i, $i = 1, ...n-1$ there is a distance determined by these points that occurs exactly i times. Solutions have been found for $2 \leq n \leq 8$. A musical scale whose pitch intervals are determined by points drawn on a circle, and that has the property asked for by Erdős is known in music theory as a *deep* scale [7]. We will transfer this terminoly from the pitch domain to the time domain and refer to cyclic rhythms with the Erdős property as *deep* rhythms.

The analysis of cyclic rhythms suggests yet another variant of the question asked by Erdős. From the musicological point of view it is desirable (especially in African timelines) not to allow empty semicircles. Such constraints suggest the following problem. Is it possible to have k points on a circular lattice of n points so that for every i, $i = k_s, k_{s+1}, ..., k_f$ (s and f are pre-specified) there is a *geodesic* distance that occurs exactly i times, with the further restriction that there is no empty semicircle?

These problems are closely related to the general problem of reconstructing sets from interpoint distances: given a distance multiset, construct all point sets that realize the distance multiset. This problem has a long history in crystallography [8], and more recently in DNA sequencing [11]. Two noncongruent sets of points are called *homometric* if the multisets of their pairwise distances are the same [10].

The preceeding suggests that it would be desirable to be able to eficiently generate rhythms that contain prescribed histogram shapes, (such as *deep* rhythms) and to find approximations when such rhythms do not exist.

The problem of comparing two binary strings of the same length with the same number of one's suggests an extremely simple edit operation called a *swap*. A swap is an interchange of a one and a zero that are adjacent to each other in the binary string. The swap distance between two rhythms is the *minimum* number of swaps required to convert one rhythm to the other.

Consider two n-bit (cyclic) binary strings, A and B, represented on a circle (necklace instances). Let each sequence have the same number k of 1's. We are interested in computing the *necklace-swap-distance* between A and B, i.e., the minimum number of swaps needed to convert A to B, minimized over all rotations of A. This distance may be computed in $O(n^2)$ time by solving a linear time problem in each of the n rotated positions. The open problem is whether the $O(n^2)$ may be improved. In contrast, the *necklace-Hamming-distance* may be computed in $O(n \log n)$ time using the Fast Fourier Transform [6].

For additional discussion of the preceeding topics the reader is referred to [13], [15], [14], [17], [16], [4], and the references therein.

References

[1] Steven Block and Jack Douthett. Vector products and intervallic weighting. *Journal of Music Theory*, 38:21–41, 1994.

[2] M. Chemillier. Ethnomusicology, ethnomathematics. The logic underlying orally transmitted artistic practices. In G. Assayag, H. G. Feichtinger, and J. F. Rodrigues, editors, *Mathematics and Music*, pages 161–183. Springer, 2002.

[3] J. Clough and J. Douthett. Maximally even sets. *Journal of Music Theory*, 35:93–173, 1991.

[4] Miguel Díaz-Bañez, Giovanna Farigu, Francisco Gómez, David Rappaport, and Godfried T. Toussaint. El compás flamenco: a phylogenetic analysis. In *Proc. BRIDGES: Mathematical Connections in Art, Music and Science*, Southwestern College, Kansas, July 30 - August 1 2004.

[5] Paul Erdős. Distances with specified multiplicities. *American Math. Monthly*, 96:447, 1989.

[6] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, 1997.

[7] Timothy A. Johnson. *Foundations of Diatonic Theory: A Mathematically Based Approach to Music Fundamentals.* Key College Publishing, Emeryville, California, 2003.

[8] Paul Lemke, Steven S. Skiena, and Warren D. Smith. Reconstructing sets from interpoint distances. Tech. Rept. DIMACS-2002-37, 2002.

[9] David Locke. *Drum Gahu: An Introduction to African Rhythm.* White Cliffs Media, Gilsum, New Hampshire, 1998.

[10] Joseph Rosenblatt and Paul Seymour. The structure of homometric sets. *SIAM Journal of Algebraic and Discrete Methods*, 3:343–350, 1982.

[11] S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bulletin of Mathematical Biology*, 56:275–294, 1994.

[12] L. Fejes Tóth. On the sum of distances determined by a pointset. *Acta. Math. Acad. Sci. Hungar.*, 7:397–401, 1956.

[13] Godfried T. Toussaint. A mathematical analysis of African, Brazilian, and Cuban *clave* rhythms. In *Proc. of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 157–168, Towson University, MD, July 27-29 2002.

[14] Godfried T. Toussaint. Algorithmic, geometric, and combinatorial problems in computational music theory. In *Proceedings of X Encuentros de Geometria Computacional*, pages 101–107, University of Sevilla, Sevilla, Spain, June 16-17 2003.

[15] Godfried T. Toussaint. Classification and phylogenetic analysis of African ternary rhythm timelines. In *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science*, pages 25–36, Granada, Spain, July 23-27 2003.

[16] Godfried T. Toussaint. A comparison of rhythmic similarity measures. In *Proc. 5th International Conference on Music Information Retrieval*, pages 242–245, Barcelona, Spain, October 10-14 2004. Universitat Pompeu Fabra.

[17] Godfried T. Toussaint. A mathematical measure of preference in African rhythm. In *Abstracts of Papers Presented to the American Mathematical Society*, volume 25, page 248, Phoenix, January 7-10 2004. American Mathematical Society.

Provably Better Moving Least Squares

Ravikrishna Kolluri James F. O'Brien Jonathan R. Shewchuk
University of California, Berkeley

1 Introduction

We analyze a variant of the implicit *moving least squares* (MLS) algorithm proposed by Shen, O'Brien, and Shewchuk [4]. We show that under certain sampling conditions the surface reconstructed by the MLS algorithm is geometrically and topologically correct.

The input to the MLS algorithm is a set of sample points S near a surface F, with approximate normals. For each sample $s \in S$ we define a linear point function that approximates the signed distance function of F in the local neighborhood of s. These functions are blended together using Gaussian weight functions yielding a smooth function I whose zero set U is the reconstructed surface. We prove that I is a good approximation to the signed distance function of the sampled surface F, and that U is homeomorphic to F and geometrically close to F.

Shen, O'Brien, and Shewchuk originally proposed their MLS construction with different weight functions for building manifold surfaces from polygon soup. Kolluri [3] showed that for reconstructing surfaces from points sets, a variant of this algorithm is geometrically and topologically correct under uniform sampling conditions. In this work we extend the analysis to handle adaptively sampled point data in which the sampling density is proportional to the local surface complexity. Our sampling requirements defined in Section 2, are similar to the sampling requirements of Delaunay-based algorithms like Crust [1].

2 Sampling Requirements

The *local feature size* (lfs) at a point $p \in F$ is the distance from p to the nearest point of the medial axis of F, as shown in Figure 1. S is an ϵ-sample of the surface F if the distance from any point $p \in F$ to its closest sample in S is less than $\epsilon\,\mathrm{lfs}(p)$. Our results are valid for values of $\epsilon \leq 0.01$.

Amenta and Bern [1] show that the function lfs is 1-Lipschitz. We extend the definition of the function lfs beyond the points on the surface F. This extension is used in defining our sampling requirements and our MLS construction. We define the *extended local feature size* of a point p as

$$\mathrm{elfs}(x) = \min_{p \in F}\{\mathrm{lfs}(p) + d(x,p) - |\phi(x)|\}.$$

Here, $\phi(x)$ is the signed distance from x to the surface F and $d(x,p)$ is the distance between point x and point p. It is easy to show that the function elfs is 1-Lipschitz and reduces to the function lfs for points on the surface.

Observation 1 *For any two points, p and q, $|\mathrm{elfs}(p) - \mathrm{elfs}(q)| \leq d(p,q)$. For any point $p \in F$, $\mathrm{elfs}(p) = \mathrm{lfs}(p)$.*

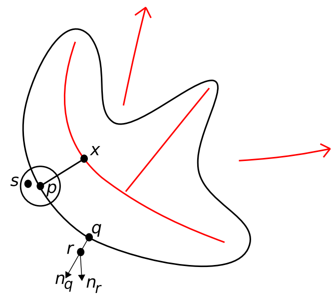


Figure 1: A closed curve along with its medial axis. The local feature size of p is the distance to the closest point x on the medial axis.

Our sampling requirements allow for noisy data when the amount of noise in the sample coordinates is small compared to the sample spacing. We assume that for each sample s, the distance to its closest surface point $p \in F$ is less than $\epsilon^2\mathrm{elfs}(s)$. We also allow a small amount of noise in the estimated sample normal. Consider a sample r with estimated normal $\vec{n}_r$, as shown in Figure 1, whose closest point in F is q with true normal $\vec{n}_q$. The angle between $\vec{n}_r$ and $\vec{n}_q$ should be less than ϵ.

Our MLS construction builds the function I by blending together functions associated with each sample point. Hence arbitrary oversampling in one region of the surface can distort the value of the function in other regions. To prohibit such oversampling, we require that local changes in the sampling density be bounded. Let α be the number of samples inside a ball of radius $\epsilon\,\mathrm{elfs}(p)$ centered at a point p. If $\alpha > 0$, the number of samples inside a ball of radius $2\epsilon\,\mathrm{elfs(p)}$ at p is at most 8α.

3 Surface Definition

The input to the MLS algorithm is a set of sample points S near the surface F. Each sample $s \in S$ has an associated vector $\vec{n}_s$ that approximates the outside normal of the surface near s.

We build a point function for each sample $s \in S$ that approximates the signed distance function of F near s. The point function $P_s(x)$ of sample point s with normal $\vec{n}_s$ is the signed distance from x to the tangent plane at s, $P_s(x) = (x - s) \cdot \vec{n}_s$. A weighted average of the point functions gives the function I whose zero set is the implicit surface we seek.

$$I(x) = \frac{\sum_{s \in S} W_s(x)((x - s) \cdot \vec{n}_s)}{\sum_{s \in S} W_s(x)}.$$

The weight functions are Gaussian functions modified by a

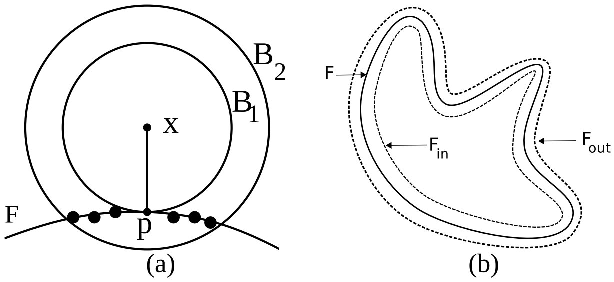


Figure 2: (a)The function I at point x is mostly determined by the point functions of the samples inside the thin shell bounded by B_1 and B_2. (b) The offset curves F_{in} and F_{out} of a curve F.

normalization factor associated with each sample point.

$$W_s(x) = e^{-\|x-s_i\|^2/\text{elfs}^2(x)}/A_s.$$

The normalization factor associated with each sample point s accounts for oversampling near s. Let $\alpha > 0$ be the number of samples inside a ball B_ϵ of radius $\epsilon\,\text{elfs}(s)$ centered at sample s, including s itself. The value of A_s is given by

$$A_s = \frac{\alpha}{\epsilon^3 \text{elfs}^3(s)}.$$

4 Results

Consider a point x whose closest point on the surface is p as shown in Figure 2(a). Let B_1(x) be a ball of radius $|\phi(x)|$ centered at x. Consider a second ball $B_2(x)$, that is slightly bigger than $B_1(x)$, also centered at x. The radius of $B_2(x)$ is $|\phi(x)| + \tau\,\text{elfs}(x)$. Here $\tau = 2\epsilon$ is a constant that depends on the sampling density. Our results are based on the observation that the value of the function at point x is mostly determined by the samples inside the thin shell bounded by $B_1(x)$ and $B_2(x)$.

Let F_{out} be the τ-offset surface outside of F that is obtained by moving each point $p \in F$ along the normal at p by a distance $\tau \cdot \text{elfs}(p)$. Similarly, let F_{in} be the τ-offset surface inside of F as shown in Figure 2(b). The τ-neighborhood is the region bounded by the inside and the outside offset surfaces. Our first geometric result is that the zero set U of I is inside the τ-neighborhood of F.

Theorem 2 *For each point x outside F_{out}, $I(x) > 0$ and for each point y inside F_{in}, $I(y) < 0$.*

Theorem 2 proves that the function I does not have any spurious zero crossings far away from the sample points. Our second geometric result is about the gradient of I at points in the zero set of I.

Theorem 3 *Let x be a point in the τ-neighborhood of F and let p be the point on F closest to x. Let $\vec{n}$ be the normal of p. Then, $\vec{n} \cdot \nabla I(x) > 0$.*

Theorem 3 proves that the gradient can never be zero inside the τ-neighborhood. From Theorem 2, the zero set of I is inside the τ-neighborhood of F. Hence, from the *implicit function theorem* [2], zero is a *regular* value of I and the zero set U is a compact, two-dimensional manifold.

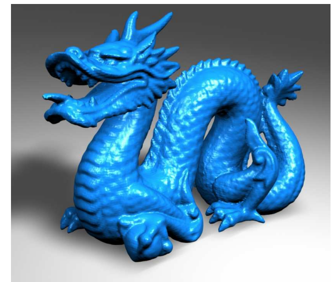


Figure 3: MLS reconstruction of the Stanford Dragon model from raw data.

We use these geometric results to define a homeomorphism between F and U. As F and U are compact, a one-to-one, onto, and continuous function from U to F defines a homeomorphism.

Definition: Let $\Gamma : \mathbb{R}^3 \rightarrow F$ map each point $q \in \mathbb{R}^3$ to its closest point on F.

Theorem 4 *The restriction of Γ to U defines a homeomorphism from U to F.*

5 Discussion

Our sampling requirement that $\epsilon \leq 0.01$ is probably an artifact of our proof technique. The MLS algorithm works quite well on data obtained from laser range, for which ϵ is much larger than 0.01 as shown in Figure 3.

Our definition of the MLS surface requires knowledge of the $\text{elfs}(x)$, function which is unknown. In our analysis, elfs can be replaced by any 1-Lipschitz function f such that $f(x) \leq \text{elfs}(x)$ at all points x, and the input sample is an ϵf-sample for $\epsilon \leq 0.01$. We can relax our requirements and assume that the elfs function is known only at the sample points. A 1-Lipschitz function function $f(x)$ can now be defined as

$$f(x) = \min_{s \in S}\{d(x,s) + \text{elfs(s)} - d(x, \text{nn}(x))\},$$

where $\text{nn}(x)$ is the sample nearest x in S.

References

[1] Nina Amenta and Marshall Bern. Surface Reconstruction by Voronoi Filtering. *Discrete & Computational Geometry*, 22:481–504, 1999.

[2] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaufman, 1997.

[3] Ravikrishna Kolluri. Provably Good Moving Least Squares. In *ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[4] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics*, 23(3):896–904, 2004.

Contour Tree Simplification With Local Geometric Measures

Hamish Carr
Dept. of Computer Science
Univ. College Dublin

Jack Snoeyink
Dept. of Computer Science
UNC Chapel Hill

Michiel van de Panne
Dept. of Computer Science
Univ. of British Columbia

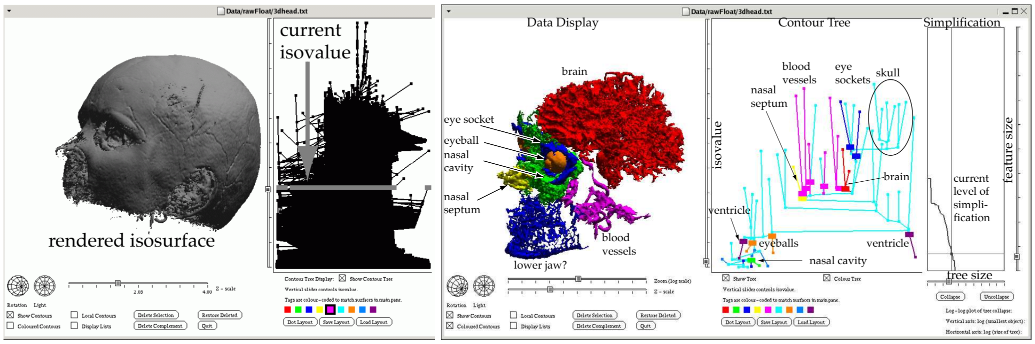


Figure 1: Left, an isosurface of the UNC Head (109 × 256 × 256 MRI) shows mostly the skull: the contour tree is unmanageable (1,573,373 edges). Right, contour surfaces chosen using a simplified contour tree. (Annotation and colour chosen to emphasize the structure of the data.)

Abstract

The contour tree, an abstraction of a scalar field that encodes the nesting relationships of isosurfaces, has several potential applications in scientific and medical visualization, but noise in experimentally-acquired data results in unmanageably large trees. We attach geometric properties of the contours to the branches of the tree and apply simplification by persistence to reduce the size of contour trees while preserving important features of the scalar field.

Keywords: Isosurfaces, contour trees, topological simplification

1 Introduction

The *contour tree* is a topological abstraction of a scalar field used in scientific and medical visualization [BPS97; vKvOB+97; PCM02; CSvdP04]. It represents changes in isosurface connectivity. In this paper, we simplify the contour tree using geometric properties of contours, permitting online simplification of the contour tree.

Figure 1 shows a conventional isosurface and a flexible isosurface [CS03] extracted from the same data set after contour tree simplification. On the left, we see that the outermost surface (the skull) occludes other surfaces, making it difficult to study structures inside the head, and the contour tree has too many edges to be useful. On the right, we see the result of using the simplified contour tree as an interface tool to enable a user to explore, color, and annotate the contours – the structures inside the head can be seen in relation to each other.

2 Related Work

The contour tree, a special case the Reeb graph [Ree46], is the result of contracting each contour in a scalar field to a single point; it tracks how *contours*, connected components of isosurfaces of a data set, appear, merge, split, and vanish as we vary the chosen isovalue. Efficient algorithms for constructing the contour tree have been reported for various meshes and interpolants [vKvOB+97; CSA03; PCM02; CLLR02; TNTF04]; the contour tree has applications ranging from fast isosurface extraction [vKvOB+97; CS03] and volume rendering [TNTF04] to mesh simplification, abstract representation of scalar fields [BR63; BPS97] and contour manipulation [CS03]. Unfortunately, noise in the input data can create many new contours by creating local minima and maxima. For noisy experimentally-acquired data such as the UNC head data set shown in Figure 1, contour trees commonly have millions of edges – too many to serve as a visual representation for the input data.

To simplify the contour tree, we would like to assign an importance to each edge and collapse edges of lower importance. This is a simple case of the ideas of *topological persistence* [EHZ03; ELZ02; BEHP03] applied to trees. Two works have applied persistence to the isovalues: [HSKK01] simplify the Reeb graph using hierarchical quantization of the data values, which can introduce errors at edges that span the quantization boundaries, and [TNTF04] simplify the contour tree using data values. We allow any geometric property to guide simplification (and are most efficient with decomposable properties, such as volume and surface area.)

3 Contour Tree Simplification

Given a contour tree and a scalar field, we simplify the contour tree with graph operators, then reflect the simplication back to the input data or use the simplified contour tree directly to extract individual contours from the simplified data set.

To compute geometric measures for individual contours, we replace the single isovalued sweep in [BPS97] with multiple separate sweeps of individual contours corresponding to sweeping individual points through the tree. Doing this requires combining partial sweep results whenever a saddle in the tree is swept past. In three dimensions, we can compute surface area, volume or *hypervolume*: isovalue integrated inside the contour.

To simplify the contour tree, we then choose a leaf edge that corresponds to contours for which the chosen geometric property is small and *prune* the leaf from the tree. By removing only leaves, we guarantee that the structure remains a tree and corresponds to a subregion of the scalar field in which isovalued contour sweeps can still be performed without discontinuous jumps.

Leaf pruning can result in a redundant vertex in the tree, as in Figure 2. We remove such vertices with no loss of topological information in the tree. Since there is no geometric cost to doing so, we prefer these vertex simplications where available and also prefer leaf prunes that maximize the number of future vertex simplifications.

Removing a leaf of the tree corresponds to flattening a local extremum of the data set as shown in Figure 2. By minimizing the geometric cost of our simplification, we are able to achieve simplification of the tree by 4 orders of magnitude without causing significant errors in the underlying field being represented, and preserving details of the contours that remain.

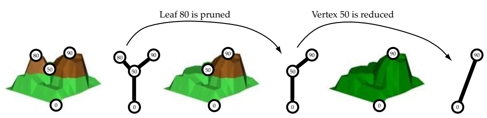


Figure 2: Leaf Pruning Levels Extrema; Vertex Reduction Leaves Scalar Field Unchanged

4 Results and Discussion

We have tested this form of simplification on a variety of data sets using the flexible isosurface interface [CS03]. In Figure 1, we show a typical result using hypervolume as the importance measure. Contours for the skull were not selected because they occlude the internal organs.

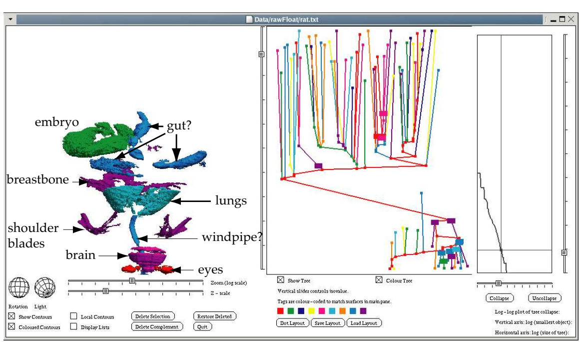


Figure 3: A Pregnant Rat MRI ($240 \times 256 \times 256$). Despite low quality data, simplifying the contour tree from 2,943,748 to 125 edges allows identification of several anatomical features.

Similarly, Figure 3 shows the result of a similar exploration of a $240 \times 256 \times 256$, low-quality MRI scan of a rat from the *Whole Frog Project* at `http://www-itg.lbl.gov/ITG.hm.pg.docs/Whole.Frog/Whole.Frog.html`. Again, simplification reduces the contour tree to a useful size. After using the *dot* tool from the graphviz package (`http://www.research.att.com/sw/tools/graphviz/`) to lay out the contour tree, these images took less than 10 minutes to explore and annotate. The result is purely a function of the topology of the isosurfaces of the input data, and uses no special constants.

5 Conclusions and Future Work

We have presented a novel algorithm for the simplification of contour trees based on local geometric measures. The algorithm is *on-line*, meaning that simplifications can be done and undone at any time. This addresses the scalability problems of the contour tree in exploratory visualization of 3D scalar fields. The simplification can also be reflected back onto the input data to produce an on-line simplified scalar field. The algorithm is driven by local geometric measures such as area and volume, which make the simplifications meaningful. Moreover, the simplifications can be tailored to a particular application or data set.

Future directions of research include extension to vectors of geometric measures, user-directed local simplification of the contour tree, utilization of the contour tree as a query structure for geometric properties, application of similar methods to volume rendering and to non-isovalue segmentation, extension to time-varying data sets, parallelization and improvements to contour tree layout algorithms.

6 Acknowledgements

Acknowledgements are due to NSERC, NSF and IRIS for funding, and to the sources of volumetric data used.

References

Peer-Timo Bremer, Herbert Edelsbrunner, Bernd Hamann, and Valerio Pascucci. A Multi-resolution Data Structure for Two-dimensional Morse-Smale Functions. In *Proceedings of IEEE Visualization 2003*, pages 139–146, 2003.

Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. The Contour Spectrum. In *Proceedings of IEEE Visualization 1997*, pages 167–173, 1997.

Roger L. Boyell and H. Ruston. Hybrid Techniques for Real-time Radar Simulation. In *Proceedings of the 1963 Fall Joint Computer Conference*, pages 445–458. IEEE, 1963.

Yi-Jen Chiang, Tobias Lenz, Xiang Lu, and Günter Rote. Simple and Output-Sensitive Construction of Contour Trees Using Monotone Paths. Technical Report ECG-TR-244300-01, Institut für Informatik, Freie Universtät Berlin, 2002.

Hamish Carr and Jack Snoeyink. Path Seeds and Flexible Isosurfaces: Using Topology for Exploratory Visualization. In *Proceedings of Eurographics Visualization Symposium 2003*, pages 49–58, 285, 2003.

Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.

Hamish Carr, Jack Snoeyink, and Michiel van de Panne. Progressive Topological Simplification Using Contour Trees and Local Spatial Measures. Accepted to IEEE Visualization 2004.

H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom.*, 30:87–107, 2003.

H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

M. Hilaga, Yoshihisa Shinagawa, T. Kohmura, and Tosiyasu L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH 2001*, pages 203–212, 2001.

Valerio Pascucci and K. Cole-McLaughlin. Efficient Computation of the Topology of Level Sets. In *Proceedings of IEEE Visualization 2002*, pages 187–194, 2002.

Georges Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de l'Acadèmie des Sciences de Paris*, 222:847–849, 1946.

Shigeo Takahashi, Gregory M. Nielson, Yuriko Takeshima, and Issei Fujishiro. Topological Volume Skeletonization Using Adaptive Tetrahedralization. In *Geometric Modelling and Processing 2004*, 2004.

Marc van Kreveld, René van Oostrum, Chandrajit L. Bajaj, Valerio Pascucci, and Daniel R. Schikore. Contour Trees and Small Seed Sets for Isosurface Traversal. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.

OrthoMap: Homeomorphism-guaranteeing normal-projection map between surfaces

Frédéric Chazal * André Lieutier † Jarek Rossignac ‡

Extended abstract

In certain graphics applications, there is a need to establish a bijection between two surfaces for texture transfer (for instance in simplification) and for discrepancy measures. Furthermore, there is a desire to express one surface as the normal offset of another for compression, multi-resolution, detail preservation during editing. For this, given two surfaces A and B, one associates to each point of A a normal displacement distance (scalar field) to the corresponding point on B. The difficulty lies in the fact that in general, such mappings are difficult or impossible to establish and, when possible, often not very satisfactory. A natural correspondence would be to map each point p of A onto its closest point $B.n(p)$ on B. The normal mapping $N(A,B)$ from A onto B associates with each point p on A its normal projection $B.n(p)$ on B. In general $N(A,B)$ is not a bijection (two different points p and q on A may have the same images $B.n(p) = B.n(q)$) neither well defined (the closest point of a point p on A may not be uniquely defined). The set of points p for which $B.n(p)$ is not unique is the medial axis $\mathcal{M}(B)$ of B [3]. In this work, we develop a simple condition on A and B and prove that it guarantees that both $N(A,B)$ and $N(B,A)$ are bijective.

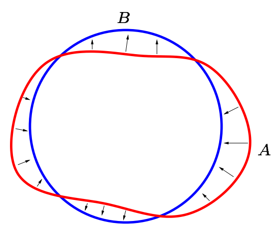


Figure 1: Two conformal curves A and B

This condition involves the Hausdorff distance between A and B and the regularity of A and B. The Hausdorff distance $H(A,B)$ between A and B may be defined in terms of r-thickening. The r-thickening $Gr(A)$ of A is the union of all open balls of radius r and center on A. Note that $Gr(A)$ is the Minkowski sum of A with an open ball of radius r and center at the origin. The r-thickening operator was used as a tool for offsetting, rounding and filleting operations [5] and for shape simplification [6]. The Hausdorff distance, $H(A,B)$, between two sets A and B is the smallest radius r such that $A \subset Gr(B)$ and $B \subset Gr(A)$. A surface A is r-regular if every point of it may be approached from both sides by an open ball of radius r that is disjoint from A. More precisely, the r-thinning $Sr(M)$ of a set M is the difference between M and the union of open balls with center out of M and the r-filleting $Fr(A)$ of A is defined as $Sr(Gr(A))$. The surface A is said to be r-regular if $Fr(A) = A$ [2]. Note that $Fr(A)$ contains all points that cannot be reached by a

*Université de Bourgogne, Institut de Mathématiques de Bourgogne, France
†Dassault Systèmes (Aix-en-Provence) and LMC/IMAG, Grenoble France
‡College of Computing, IRIS, GVU, Georgia Tech., Atlanta, Georgia

ball of radius r whose interior does not interfere with A. The values r for which A is r-regular are related to the local feature size $\mathrm{lfs}(A)$ [1], and defined as the minimum distance between A and its medial axis $\mathcal{M}(A)$. The surface A is r-regular for all $r \leq \mathrm{lfs}(A)$.

Definition: *A and B are said to be conformal (to each other) when A and B are both r-regular for $r = H(A,B)/(2-\sqrt{2})$.*

The following theorem is the main result of this paper:

Theorem: *If surfaces A and B are conformal, then the normal mapping $N(A,B)$ is bijective.*

Moreover, $N(A,B)$ allows to define an explicit isotopy (i.e. a continuous deformation of A into B) between A and B (see [4] for a precise definition). The proof of the theorem is cast in precise mathematical formalism which allows to prove this theorem for manifolds in any dimension. We also show that conformality is a tight condition by giving an example of two curves A and B that are r-regular for any $r \leq H(A,B)/(2-\sqrt{2})$, but not conformal, and hence $N(A,B)$ is not bijective (see figure 2).

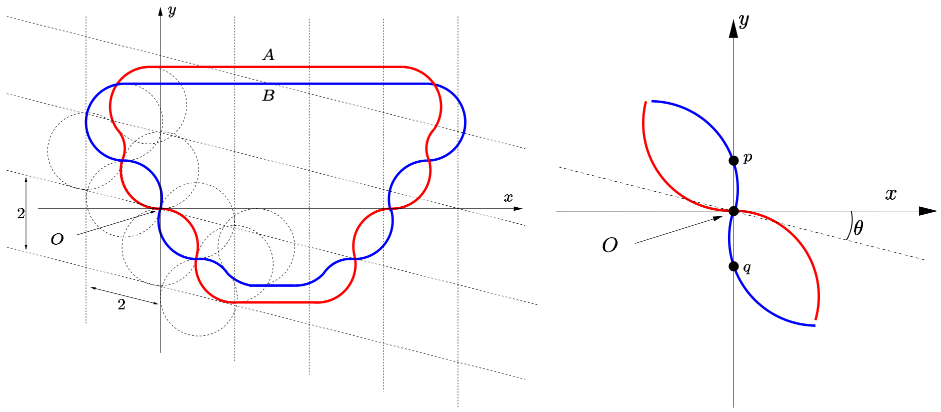


Figure 2: Two curves A and B showing that optimality is tight. Right part is a zoom of a neigborhood of 0 : $A.n(p) = A.n(q)$

In summary, we have provided a sufficient and tight condition to ensure that the normal mapping between two smooth $(n-1)$-dimensional manifolds in n-D is a bijection. The condition links the minimum regularity of the manifolds to their Hausdorff distance.

References

[1] N. Amenta and M. Bern *Surface Reconstruction by Voronoi Filtering* Discrete and Computational Geometry, no 22 pp.481-504(1999)

[2] D. Attali, *r-Regular shape reconstruction from unorganized points*, Computational Geometry (1997) 248-253.

[3] H. Blum, *A transformation for extracting new descriptors of shape*, In W. Wathen-Dunn, editor, Models for the Perception of Speech and Visual Form, p.362-380, MIT Press, 1967.

[4] F. Chazal, D. Cohen-Steiner, *A condition for isotopic approximation,* proc. ACM Symp. Solid Modeling and Applications 2004.

[5] J. Rossignac, A. Requicha, *Offsetting Operations in Solid Modelling*, Computer-Aided Geometric Design, Vol. 3, pp. 129-148, 1986.

[6] J. Williams, J. Rossignac, *Mason: Morphological Simplification* GVU Tech. Report GIT-GVU-04-05 (Mason2.pdf) available from http://www.gvu.gatech.edu/ jarek/papers.html

Parallel Guaranteed Quality Planar Delaunay Mesh Generation by Concurrent Point Insertion*

Extended Abstract

Andrey N. Chernikov and Nikos P. Chrisochoides
Computer Science Department
College of William and Mary
Williamsburg, VA 23185

Abstract *We develop a theoretical framework for constructing parallel guaranteed quality Delaunay planar meshes using commercial off-the shelf software (COTS). We call two points Delaunay-independent if they can be inserted concurrently without destroying the conformity and Delaunay properties of the mesh. First, we present a sufficient condition of Delaunay-independence. It is based on the distance between points, can be verified very efficiently and used in practice. Second, we show that a simple block mesh decomposition can be utilized in order to guarantee a-priori Delaunay-independence of points in certain regions. Third, we derive an expression which relates three mesh quality and size parameters that allow to conduct the preprocessing step of our approach using a sequential Delaunay refinement algorithm. We conclude with our current work in progress that includes extending the presented approach to generate nonuniform graded meshes.*

Introduction

Nave, Chrisochoides, and Chew [7] presented a practical provably-good parallel mesh refinement algorithm for polyhedral domains. The approach in [7], due to absence of sequential code reuse, as well as intensive unpredictable communication and setbacks, is labor intensive. In this paper, we develop an approach which allows to use COTS, requires only structured bulk communication, and eliminates setbacks by ensuring that the inserted points are Delaunay-independent.

Linardakis and Chrisochoides [6] described a Parallel Domain Decoupling Delaunay method for 2-dimensional domains, which is capable of leveraging the serial meshing codes. However, it is based on the Medial Axis which is very expensive and difficult to construct for 3-dimensional geometries. The approach developed in the present work is domain decomposition independent, i.e. it does not require an explicit construction of internal boundaries.

Blelloch, Hardwick, Miller, and Talmor [2] describe a divide-and-conquer projection-based algorithm for constructing Delaunay triangulations of pre-defined point sets in parallel. Our goal, though, is to refine an existing mesh by inserting triangle circumcenters, i.e., the set of points in the final mesh is not known in advance.

Kadow in [5] extended [2] for parallel mesh generation. The principal difference between [7] and [5] is that in [5] the need to construct an initial mesh sequentially is eliminated.

Edelsbrunner and Guoy [4] define the points x and y as independent if the closures of their *prestars* (or *cavities* [7]) are disjoint. We start with proving a similar condition of point independence [3]. Our formulation is less restrictive: it allows the cavities to share a point. However, computing the cavities and their intersections for all candidate points is very expensive. That is why we do not use coloring methods that are based on the cavity graphs and we prove a theorem, which allows to use only the distance between the points for checking their Delaunay-independence. The minimum separation distance argument in [4] is used to derive the upper bound on the number of inserted vertices and prove termination, but not to ensure point independence.

Spielman, Teng, and Üngör [9] presented the first theoretical analysis of the complexity of parallel Delaunay refinement algorithms. However, the assumption is that the global mesh is completely retriangulated each time a set of independent points is inserted [11]. In [10] the authors developed a more practical algorithm which takes $\mathcal{O}(\log m)$ time (i.e. number of parallel iterations) using m processors, where m is the size of the output. In contrast, our approach [3] uses only four parallel refinement iterations with a fixed number of processors, where each iteration on a single processor is performed by a sequential mesher [8]. The present work is an extension of the work we presented in [3].

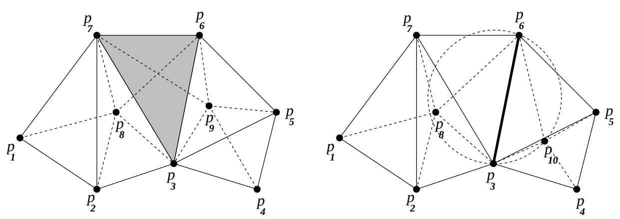
The Theoretical Framework

Sequential Delaunay refinement algorithms are based on inserting circumcenters of triangles which violate the required bounds, e.g. the upper bound $\bar{\rho}$ on circumradius-to-shortest edge ratio, and the upper bound $\bar{\Delta}$ on triangle area. Let the *cavity* $\mathcal{C}_{\mathcal{M}}(p)$ *of point* p *with respect to mesh* $\mathcal{M}$ be the set of triangles in $\mathcal{M}$, whose open circumdisks include p. We expect our parallel Delaunay refinement algorithm to insert multiple circumcenters concurrently in such a way that at every iteration the mesh will be both conformal and Delaunay. Figure 1 illustrates how the concurrently inserted points can violate one of these conditions.

Theorem 1 *Let* $\bar{r}$ *be the upper bound on triangle circumradius in the mesh and* $p_i, p_j \in \Omega \subset \mathbb{R}^2$*. Then if* $\|p_i - p_j\| \geq 4\bar{r}$*, then independent insertion of* p_i *and* p_j *will result in a mesh which is both conformal and Delaunay.*

To show that Theorem 1 is applicable throughout the run of the algorithm, we prove that the execution of the

*This work was supported by NSF grants: CCR-0049086, ACI-0085969, EIA-9972853, EIA-0203974, and ACI-0312980



(a) **(b)**

Figure 1: **(a)** If $\triangle p_3p_6p_7 \in \mathcal{C}(p_8) \cap \mathcal{C}(p_9)$, then concurrent insertion of p_8 and p_9 yields a non-conformal mesh. Solid lines represent edges of the initial triangulation, and dashed lines — edges created by the insertion of p_8 and p_9. Note that the intersection of edges p_8p_6 and p_9p_7 creates a non-conformity. **(b)** If edge p_3p_6 is shared by $\mathcal{C}(p_8) = \{\triangle p_1p_2p_7, \triangle p_2p_3p_7, \triangle p_3p_6p_7\}$ and $\mathcal{C}(p_{10}) = \{\triangle p_3p_5p_6, \triangle p_3p_4p_5\}$, the new triangle $\triangle p_3p_{10}p_6$ can have point p_8 inside its circumdisk, thus, violating the Delaunay property.

# of processors	Pipe with holes		Unit square	
	Time, sec.	# of elmnts, $\times 10^6$	Time, sec.	# of elmnts, $\times 10^6$
4	179.1	14.6	293.7	23.8
64	273.6	233.3	300.1	470.7
121	212.7	441.1	293.7	873.5

Table 1: Scaled workload, the area bound is inversely proportional to the number of processors.

Bowyer/Watson kernel [7], either sequentially or in parallel, does not violate the condition that $\bar{r}$ is the upper bound on triangle circumradius in the entire mesh.

Theorem 2 *The condition that $\bar{r}$ is the upper bound on triangle circumradius in the entire mesh holds both before and after the insertion of a point.*

In order not to check the independence condition for every pair of candidate points, we utilize a coarse-grained domain decomposition scheme. A coarse uniform lattice is overlapped over the triangulation domain in such a way that any pair of points in non-adjacent cells are guaranteed to be no less that $4\bar{r}$ apart. To enforce the $\bar{r}$ circumradius bound in the mesh we derive the following relation which allows to use the standard sequential Delaunay refinement algorithms for preprocessing:

Theorem 3 *If $\bar{\rho}$ and $\bar{\Delta}$ are upper bounds on triangle circumradius-to-shortest edge ratio and area, respectively, then $\bar{r} = 2(\bar{\rho})^{3/2}\sqrt{\bar{\Delta}}$ is an upper bound on triangle circumradius.*

Some results for shared and distributed memory implementations[1] are shown in Tables 1 and 2. Table 1 also indicates that there is potential for improvement by using the Load Balancing Library [1].

[1]This work was performed using computational facilities at the College of William and Mary which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund.

# of processors	Time, sec. MPI	Time, sec. OpenMP	# of elmnts, $\times 10^6$
4	220.3	214.1	14.6

Table 2: Pipe cross-section, distributed (MPI) and shared (OpenMP) memory implementations.

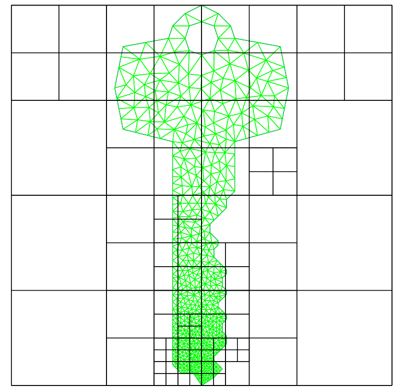


Figure 2: Graded mesh of Jonathan Shewchuk's key. The parallel refinement is guided by a quadtree.

Conclusions and Work In Progress

The approach we developed allows the use of sequential COTS for guaranteed quality parallel meshing.

Currently, we are working on extending our results to graded meshes like the one shown in Fig. 2 by using a quadtree instead of a uniform lattice, and to 3 dimensions.

References

[1] K. Barker, A. Chernikov, N. Chrisochoides, and K. Pingali. A load balancing framework for adaptive and asynchronous applications. *IEEE TPDS*, 15(2):183–192, Feb. 2004.

[2] G. E. Blelloch, J. Hardwick, G. L. Miller, and D. Talmor. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 24:243–269, 1999.

[3] A. N. Chernikov and N. P. Chrisochoides. Practical and efficient point insertion scheduling method for parallel guaranteed quality Delaunay refinement. *Proc. 18th Intl. Conf. on Supercomputing*, 48–57. ACM Press, 2004.

[4] H. Edelsbrunner and D. Guoy. Sink-insertion for mesh improvement. *Proc. 17th Simp. on Comp. Geom.*, 115–123. ACM Press, 2001.

[5] C. Kadow. Adaptive dynamic projection-based partitioning for parallel Delaunay mesh generation algorithms. *SIAM Workshop on Combinatorial Sci. Comp.*, Feb. 2004.

[6] L. Linardakis and N. Chrisochoides. Parallel domain decoupling Delaunay method. *SIAM J. Sci. Comp.*, under revision, 2004.

[7] D. Nave, N. Chrisochoides, and L. P. Chew. Guaranteed–quality parallel Delaunay refinement for restricted polyhedral domains. *Comp. Geom.: Theory and Applications*, 28:191–215, 2004.

[8] J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Proc. 1st Workshop on Appl. Comp. Geom.*, 123–133, 1996.

[9] D. A. Spielman, S.-H. Teng, and A. Üngör. Parallel Delaunay refinement: Algorithms and analysis. *Proc. 11th Intl. Meshing Roundtable*, 205–217, 2001.

[10] D. A. Spielman, S.-H. Teng, and A. Üngör. Time complexity of practical parallel Steiner point insertion algorithms. *Proc. 16th Simp. on Parallelism in alg. and arch.*, 267–268. ACM Press, 2004.

[11] S.-H. Teng. Personal Communication. *SIAM PP'04*, Feb. 2004.

Tightening: Curvature-Limiting Morphological Simplification

Jason Williams and Jarek Rossignac

Given a planar set S of arbitrary topology and a radius r, we define an r-tightening of S, which is a set that has a radius of curvature everywhere greater than or equal to r and that only differs from S in a morphologically-defined tolerance zone. This zone, which we call the mortar, contains only the details of S, such as high curvature portions of its boundary, thin gaps and constrictions, and small holes and connected components. We describe how to approximately compute r-tightenings for shapes represented as binary images using constrained, level set curvature flow.

Our work addresses a formulation of the shape smoothing problem different from those given in most prior art. The energy minimization-based fairing methods in the CAD/CAM literature (such as [3]) and the various polygon mesh smoothing techniques in the graphics literature (such as [2]) typically do not guarantee a bound on curvature or confine shape changes to a tolerance zone like the mortar.

The mortar, which we introduced in [7], is defined in terms of the morphological operations of rounding and filleting, which are described in detail in [4]. S rounded by r, denoted $R_r(S)$, is the union of disks of radius r contained in S, while S filleted by r, denoted $F_r(S)$, is the complement of the union of disks contained in the complement of S. The mortar is $F_r(S)$-$R_r(S)$. It is empty away from thin and high-curvature regions of S, and around those regions it is a subset of all points within a distance r of the boundary of S.

We define a simple closed curve C lying in a set T as tight with respect to T if it locally minimizes length, so that there exists a Φsuch that for all t and all $\Phi \leq \Phi$, $d(C(t),C(t+\Phi)) = \Phi$, where d(A,B) is the length of the shortest path connecting A and B in T and C is parameterized by arclength. We define a point on the boundary of a shape as concave if the line segment connecting the intersections of a small circle centered on the point with the boundary lies completely outside the shape. Tight loops through a set consist of concave portions of its boundary connected by tangent line segments. Because concave portions of the boundary of the mortar have a radius of curvature greater than or equal to r, if we define an r-tightening of S as a set T, $R_r(S)\ \Phi\ T\ \Phi\ F_r(S)$, such that the bounding loops of T are tight with respect to the mortar of S, it follows that the boundary of an r-tightening also has a radius of curvature greater than or equal to r.

When $R_r(S)$ and the complement of $F_r(S)$ each consist of a single connected component, the tightening is unique, and its boundary is the shortest loop around $R_r(S)$ lying in $F_r(S)$. In this case the tightening corresponds to the relative convex hull or minimum perimeter polygon [6] of $R_r(S)$ in $F_r(S)$. When $R_r(S)$ and $F_r(S)$ have more complex topologies, there may be several different tightenings, each of which may have holes and multiple connected components

We conjecture that for shapes of arbitrary topology represented as binary images, level-set curvature flow [5] constrained to the mortar always converges to a tightening, which includes as a corollary that a tightening always exists. In our implementation of curvature flow, we initialize the level set function Φ to be the signed Euclidean distance to the boundary of the core of the input shape, approximately computed using

Danielsson's vector propagation algorithm [1], which we also use for implementing the morphological operations. At each iteration, we compute $\nabla_t = -F\ |\nabla\nabla|$, where F is the velocity of the level set, which is equal to the curvature in the mortar and zero outside the mortar. The curvature is given by

$$- = - -\frac{—}{|—|} = \frac{-_{xx}-_{y}^{2} - 2-_{x}-_{y}-_{xy} + -_{yy}-_{x}^{2}}{\left(-_{x}^{2} + -_{y}^{2}\right)^{3/2}}$$

Where $|\nabla\nabla| = (\nabla_x^{\,2}+\nabla_y^{\,2})^{1/2}$, and the partial derivatives are computed using finite differences. We then compute $\nabla\ (t+\Delta t, x, y) = \nabla\ (t, x, y) + \Delta t \bullet \nabla_t(t, x, y)$, where Δt is inversely proportional to the maximum value of F at time t, so that the level set crosses at most one pixel each iteration. During most iterations, we only update ∇ in a narrow band of pixels around the zero level set.

Curvature flow converges slowly where the radius of curvature spans several pixels. We therefore downsample the image representation of the core by a factor of two until r corresponds to 1-2 pixels. We perform the flow at this coarse resolution, then iteratively upsample by a factor of two and re-perform the flow. We find we need less than 100 iterations at each level of resolution. We anticipate adapting this technique to generate three-dimensional results using mean curvature flow.

References

[1] P. Danielsson. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing* 14, 227-248.

[2] M. Desbrun, M. Meyer, et. al. Implicit Fairing of irregular meshes using diffusion and curvature flow. *SIGGRAPH 99*, 317-324.

[3] H. Moreton and C. Sequin. 1992. Functional optimization for fair surface design. *SIGGRAPH 92,* 167-176.

[4] J. Rossignac. 1985. Blending and offsetting solid models. Ph.D. thesis, University of Rochester.

[5] J. Sethian. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

[6] J. Sklansky, R. Chazin, and B. Hansen. 1972. Minimum-perimeter polygons of digitized silhouettes. *IEEE Transactions on Computers*, 21, 3, 260-268.

[7] J. Williams and J. Rossignac. 2004. Mason: morphological simplification. GVU Technical Report 04-05, http://www.cc.gatech.edu/gvu/research/techreports.html. Submitted for publication.

Compact Data Representations and their Applications

Moses Charikar
Princeton University

Several algorithmic techniques have been devised recently to deal with large volumes of data. At the heart of many of these techniques are ingenious schemes to represent data compactly. This talk will present some constructions of such compact representation schemes (also referred to as sketches) for estimating distances between sets, vectors, and distributions on an underlying metric (where distance is measured by the Earth Mover Distance (EMD)). The construction of these compact representation schemes is motivated by techniques used in approximation algorithms to round solutions of LP and SDP relaxations for optimization problems. There are interesting connections between such sketching methods and low distortion embeddings of metric spaces into ℓ_1. Such compact representation directly lead to efficient approximate nearest neighbor search algorithms. We will also see how such schemes lead to efficient, one-pass algorithms for processing large volumes of data (streaming algorithms).

Finally, I will talk about some recent work applying these ideas to designing compact data structures for content-based image retrieval systems. The main challenge here is to achieve high-quality similarity searches while using very compact meta-data. We adapt the ideas from the sketch constructions for EMD, as well as other ideas from embeddings of normed spaces to produce compact sketches for images. I will discuss results from a prototype implementation on a database with 10,000 images. Our results show that our method can achieve more effective similarity searches than previous approaches with meta-data significantly smaller than previous systems.

The work on content-based image retrieval is joint work with Qin Lv and Kai Li at Princeton.

NEARPT3 — Nearest Point Query in E3 with a Uniform Grid

[Extended Abstract]

W. Randolph Franklin
ECSE Dept, 6026 JEC, RPI, Troy NY 12180
geom@wrfranklin.org, http://wrfranklin.org

1. INTRODUCTION

We present NEARPT3, an algorithm and preliminary implementation to preprocess a large set of fixed points and then perform nearest point queries against them. With fixed and query points drawn from the same distribution, NEARPT3's expected preprocessing and query time are $\Theta(N)$, with a very small constant factor. NEARPT3, designed for large datasets, has been tested on the largest datasets in the Georgia Tech Large Geometric Models Archive, [7]. Therefore, processing tens of millions of points is quite feasible.

The prior art includes various data structures and algorithms for variants of nearest neighbor searching. The cost of a Voronoi diagram, [6], in E^3 is data dependent, amd runs from $\Omega(N \log N)$ to $O(N^2)$ in time and space for preprocessing, with each query costing $\theta(\log N)$. Range trees, [6], cost $\theta(N \log N)$ time to preprocess, with each query also costing $\theta(\log N)$. ANN (Approximate Nearest Neighbors), [2], is a C++ library for approximate and exact nearest neighbor searching in E^d, allowing a variety of metrics, implemented with several different data structures, based on kd-trees and box-decomposition trees. All these algorithms and data structures are more general, hence bigger, than NEARPT3, which is optimized specifically for the L_2 metric in E^3, although its ideas would generalize.

NEARPT3 appears to be the only method that enthusiastically rejects hierarchical data structures and search techniques. Trees and subdivision searching are more robust against adversarially chosen input. However, we believe, based on tests on real data, that they are often suboptimal in practice. This is true even when the real data is moderately unevenly distributed. The extreme data unevenness that would destroy NEARPT3's performance would also force hierarchical data structures to have many levels. In that case, where the hierarchies would then be faster than NEARPT3, though not fast, a shallow hierarchy would perhaps be the least slow.

NEARPT3 has three stages, as follows. The data structure is a uniform grid, [1, 4].

Prepreprocess: This step, which does not depend on the data, need be performed only once. Hence it is excluded from the time statistics, just as the compilation time is also excluded. Indeed, this prepreprocessing could be forced into the C++ compilation step using the template specialization facilities, though that would be silly.

1. Generate the coordinates (x, y, z) of all grid cells with $0 \le x \le y \le z \le R$ for some fixed R.
2. Sort them by $\sqrt{x^2 + y^2 + z^2}$.
3. Pass down the list in order. For each cell c_1, find the last cell, c_2, whose closest point to the origin is at least as close as the farthest point of c_1. Call c_2 the *stop* cell.

   Since the stop cells are monotonically increasing, all this requires only one pass down the cell list.

   The point is that if a point has been found in c_1, we have to continue searching through c_2 to be sure of finding any closer points.
4. Write the sorted list of cells and stop cells to a file.

Preprocess: Here the fixed points are built into the data structure.

1. Compute a uniform grid resolution, G from the number of fixed points, N_f or get it from the user. A reasonable value is $G = r\sqrt[3]{N_f}$, for $1/2 \le r \le 2$.
2. Allocate a uniform grid with one word per cell, to store a count of the number of points in each cell.
3. Read the fixed points, determine which cell of the uniform grid each would fall in, and update the counts.
4. Allocate a ragged array for the uniform grid, with just enough space in each cell for the points in that cell.

   A ragged array contains storage for the points plus a dope vector pointing to the first point of each cell. The total variable storage is one word per cell, plus the storage for the points.
5. Process the fixed points again, computing for a second time the cell that each falls into. This time, store each point in its proper cell.

   The goal is to minimize both the storage used and the number of storage reallocations. Storage reallocations become especially costly as the program's memory working set approaches the computer's available real memory.

   A possible alternative would be to use a linked list for the points in each cell. However, the space used for the pointers would be significant, and the points in each cell would be scattered throughout the memory, which might reduce the cache performance.

Another alternative would be to use a C++ STL vector, which reallocates its storage as it grows. Our experience finds this to be very suboptimal.

Query: This reports the closest fixed point to a query point.

1. Determine which cell, c, contains the query point.
2. Using the sorted cell list computed in the preprocessing step, spiral out from c until a cell with at least one point is found. Often this is c itself.

   For each cell with coordinates (x, y, z) in the sorted cell list, up to 47 other reflected and rotated cells are derived, such as $(-x, z, y)$. If any ordinate is zero, or any two are equal, there will be fewer other cells.

   It would be possible to do this reflection, rotation, and duplicate deletion in the prepreprocessing stage. This would cause a much larger sort cell list. However it would reduce the query time because that code would have fewer conditionals, which should make it more optimizable.
3. Continue spiralling out until c's stop cell to find any closer points, if one exists.

   This spiralling process is conservative since it ignores the location of the query point inside c.

   On the average 200 cells are searched for each query, but checking each cell is very fast.

2. TESTS

NEARPT3's performance is data dependent. An improper choice of input, such as query points that are very far from all the fixed points, will be intolerably slow. Nevertheless, all the data sets tested so far perform quite well, including these:

Data set name	Source	# fixed points	# queries	CPU time, secs
Bunny	GIT	17973	17974	1.9
Bone6	GIT	284818	284818	28
Dragon	GIT	218882	218883	21
Hand	GIT	163661	163662	16
Uniform random	generated	1M	1M	128

The environment is a 2002-vintage IBM T30 Thinkpad laptop computer with 768 MB of memory, a 1600 MHz Pentium 4 Mobile CPU, and Intel's icpc 8.1 C++ compiler, with all optimizations enabled. The times include reading the data and writing the results. These experiments also validate that the cost is linear. The preprocessing cost is $\Theta(N)$. Each query may cost $O(N)$, but typically costs $\Theta(1)$.

NEARPT3's cost is affected by the grid resolution, however values within a factor of two of the optimum typically change the time less than a factor of 2.

3. EXTENSIONS

NEARPT3 could return approximate nearest matches in much less time since the spiral search could stop sooner. In E^d for other d, the cost of searching is exponential in d, as for any search procedure.

NEARPT2 is a simplified version for preprocessing and searching for points in E^2. We tested 1M queries against 1M fixed points, both sets randomly generated, using NEARPT2, CGAL 3.0.1's NEAREST_NEIGHBOR_SEARCHING, [3], and ANN 0.2, [5]. A proper choice of compiler flags would probably speed both CGAL and ANN, but not reduce their storage cost. None of these tests required any data I/O since the input was randomly generated and the output not written. Performing 1M queries against 1M fixed points cost as follows.

Program	Time	Storage
NEARPT2	9.4	46MB
CGAL NNS	41	120MB
ANN	41	128MB

We then tried 10M fixed and 10M query points but CGAL and ANN required too much memory. NEARPT2 used 458MB and 98 seconds.

4. SUMMARY

The general lesson of NEARPT3 is that simple data structures like the uniform grid can be quite efficient in both time and space in E^3.

5. ACKNOWLEGEMENTS

This research was supported by NSF grant CCR-0306502.

6. REFERENCES

[1] Akman, V., W. R. Franklin, M. Kankanhalli, and C. Narayanaswami. Geometric computing and the uniform grid data technique. Computer Aided Design **21**(7), (1989), 410–420.

[2] Arya, S. and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*. 271–280.

[3] CGAL. The CGAL home page. `http://www.cgal.org/`, 2003.

[4] Franklin, W. R. and M. Kankanhalli. Parallel object-space hidden surface removal. In *Proceedings of SIGGRAPH'90*, volume 24. 87–94.

[5] Mount, D. and S. Arya. ANN: library for approximate nearest neighbor searching version 0.2 (beta release). `http://www.cs.umd.edu/~mount/ANN/`, 1998.

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

[7] Turk, G. and B. Mullins. Large geometric models archive, 2003. URL `http://www.cc.gatech.edu/projects/large_models/`.

Algebraic Number Comparisons for Robust Geometric Operations

John Keyser, Koji Ouchi
Department of Computer Science
Texas A&M University

In this talk, we describe a method for exact comparison of algebraic numbers, with application to geometric modeling.

Motivation: Computations with algebraic numbers are of key importance in several geometric computations. Algebraic numbers arise as solutions to systems of polynomials—a common operation in many geometric applications, particularly those involving curved objects. Polynomials regularly describe the relationships between even basic geometric objects, for example the (squared) distance between two points. For more complex curved geometric objects, polynomials are often used to describe the actual shapes. Finding solutions to systems of polynomials thus becomes a key operation in numerous geometric applications.

Unfortunately, the robustness issues well-known in traditional computational geometry become even more significant when dealing with algebraic numbers and curved geometry. Bounding and handling the numerical error that arises in these computations becomes more problematic, as does the detection and elimination of degeneracy problems. For reliable computation on curved geometry, we therefore need robust operations on algebraic numbers. We choose to use an exact-computation approach, achieving robustness by eliminating numerical error, while supporting straightforward operations even in the presence of degeneracies.

Our work is particularly motivated from the field of computer-aided geometric design. Finding intersections of geometric objects involves solving systems of polynomials, usually of moderate degree in a few variables. Our methods apply, however, to a far wider range of problems.

Background: Our earlier work focused on techniques for exact manipulation of algebraic curves and 2D points [MAPC00], and applied these to solid modeling, producing the first exact boundary evaluation system [ESOLID04]. Although this work yielded greater robustness by eliminating numerical error, degeneracies could not be handled effectively.

Computations with algebraic numbers has been a topic of recent research interest among a variety of other researchers. LEDA supports a limited set of constructions for algebraic numbers, though it does not solve polynomial systems [LEDA]. The Core library supports a wider variety of number types, including real algebraic numbers [CORE]. Such exact computation approaches have been incorporated in larger projects, such as Exacus [EXACUS] and CGAL [CGAL]. Recently, Emiris and Tsigaridas have developed an approach for exact comparison of algebraic numbers of relatively small degree (at most 4) that is asymptotically faster than an expliit solution [ET04].

Major Results: We describe a method for comparing complex algebraic numbers exactly. More precisely, one can know whether or not the real and imaginary parts of given a pair of complex algebraic numbers are identical. In particular, one can test whether or not the real and imaginary parts of a given complex algebraic number vanish.

Our method is based on the rational univariate reduction (RUR). The RUR computes common roots of systems of multivariate polynomials with rational coefficients. The roots are represented in terms of a set of univariate polynomials with rational coefficients. These polynomials, when evaluated at the roots of another univariate polynomial, yield the coordinates of the common roots of the original system. The RUR can be computed exactly, i.e. the coefficients of these polynomials can be computed to full precision.

As a key advantage over our earlier methods, this RUR method works even in the presence of "degenerate" situations. For example, the RUR approach handles roots of high multiplicity, finds roots at singularities, and works even when the underlying set of roots is positive dimensional. As such, it offers a general method for achieving robust calculations with algebraic numbers.

Our RUR implementation gives an exact representation of points with algebraic coordinates. This allows us to exactly determine geometric predicates such as whether a point lies on a curve or surface. We can also determine how surfaces intersect, e.g. whether the surfaces meet in "general position." Thus, our representation is very general, and can serve as a single representation for all such algebraic points. Although this point representation is more robust than our earlier approach that could not represent degeneracies, it is also less efficient. We therefore propose the use of the RUR computation in a hybrid fashion, using it in cases where there are likely to be difficulties due to degeneracies.

We describe several applications, with special emphasis on geometric modeling. In particular, we describe a new implementation that has been used successfully on certain degenerate boundary evaluation problems. We describe how the RUR can be used to detect when degeneracies occur, and how it can then be combined with a numerical perturbation scheme to achieve an overall more robust boundary evaluation..

References:

[CGAL] Computational Geometry Algorithms Library: http://www.cgal.org

[CORE] Core library: http://www.cs.nyu.edu/exact/core

[ESOLID04] John Keyser, Tim Culver, Mark Foskey, Shankar Krishnan, Dinesh Manocha, "ESOLID: A System for Exact Boundary Evaluation," *Computer Aided Design*, vol. 36, no. 2, pp. 175-193, 2004.

[ET04] Ioannis Emiris, Elias Tsigaridas, "Comparing Real Algebraic Numbers of Small Degree," Proceedings of European Symposium on Algorithms, 2004.

[EXACUS] Efficient and Exact Algorithms for Curves and Surfaces: http://www.mpi-sb.mpg.de/projects/exacus/

[LEDA] Library for Efficient Data Types and Algorithms: http://www.algorithmic-solutions.com/enleda.htm

[MAPC00] John Keyser, Tim Culver, Dinesh Manocha, Shankar Krishnan, "Efficient and Exact Manipulation of Algebraic Points and Curves," *Computer Aided Design*. Vol 32, No. 11. pp. 649-662. 2000. Special Issue on Robustness.

An Efficient k-Center Clustering Algorithm for Geometric Objects

Guang Xu and Jinhui Xu
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260, USA
{ *guangxu, jinhui*}*@cse.buffalo.edu*

1 Overview

Clustering is a fundamental problem in computational geometry and finds numerous applications in many different fields such as data mining, image processing, and pattern classification and recognition. Extensive research has been done on investigating theoretically and/or practically efficient approaches for solving various variants of this problem, and a number of important results have been obtained in recent years. Among many of its variants, the Euclidean k-center clustering problem (or more generally, the k-center clustering problem in a metric space) of a set of points has received considerable attention from the fields of computational geometry and approximation algorithms, and an $O(n \log k)$-time, 2-approximation algorithm has been obtained [4, 5] by using a farthest point method. The success of this constant approximation algorithm relies on a key fact that the distances among the set of points satisfy the metric property. This algorithm, although simple, is actually quite close to the limit of approximation. It has been shown that approximating the Euclidean k-center problem within any ratio smaller than 1.822 is NP-hard [3].

In this paper, we consider an interesting generalization of the Euclidean k-center clustering problem, called *k-center clustering problem of point sets (or KCS for short)* in d-dimensional Euclidean space R^d. In the KCS problem, we are given a sequence of finite point sets in R^d space, denoted by $S_1, S_2, \ldots, S_n$, respectively, and an integer $k \geq 1$, and are required to find k congruent spheres $B_1, B_2, \ldots, B_k$ of minimum radius so that for each $S_i, 1 \leq i \leq n$, there is at least one ball $B_j, 1 \leq j \leq k$, containing all points in S_i. The KCS problem models the k-center clustering problem of a set of bounded geometric objects represented by polyhedra. This is because for any polyhedron, it is sufficient to consider the vertices of its convex hull. Different from the ordinary k-center clustering problem, in our KCS problem, the geometric objects represented by S_i and S_j, for any $i, j \leq n$, could overlap with each other.

The KCS problem we study is motivated by applications in biology where a set of replication/transcription sites, each represented by a 3-D polyhedron, needs to be clustered into clusters to form replication or transcription zones for studying the behavior of chromatic domains in cell nucleus. It also appears in many applications of the Euclidean k-center clustering problem where each point in the k-center clustering problem is actually the representative point (e.g., the center) of some geometric object. When the sizes of the geometric objects are small enough, simplifying each object to a point does not affect the quality of clustering too much, since the distance function of the set of objects can be well approximated by that of the representative points. However, when the size of each object is large, the distance between two objects could be significantly different from that of their representative points. Thus the quality of the obtained clustering through representative points could be rather poor.

In general, the distances among a set of geometric objects do not satisfy the triangle inequality, thus making almost all previously known k-center clustering algorithms fail. To overcome the difficulty caused by the loss of the triangle inequality property, we use the ideas of shape simplification and core set. More specifically, we first simplify each object (or point set) to some simple geometry shape, and then determine several representative points from each simplified shape. The representative points of each object may not belong to its original object. In order to achieve a quality solution, the simplified shapes and representative points should have the following features:

1. The size and shape of the original objects are somewhat preserved.

2. The position of the original objects are well reflected.

Based on several interesting observations, we will show that each geometric object can be represented by a small set of representative points, and an efficient 3-approximation algorithm can be obtained by clustering the set of representative points. In summary, we prove the following theorems.

Theorem 1 *There is an $O(m+n\log k)$ time 3-approximation algorithm for the k-center clustering problem of point sets in R^d space, where $m=\sum_{i=1}^{i=n}|S_i|$.*

The constant hidden in the O-notation of the running time has exponential dependence on the dimension d. If the dimension d is very high, we have the following result.

Theorem 2 *There is an $O(\frac{dm}{\epsilon^2}+dnk+\frac{1}{\epsilon^{O(1)}})$ time $3+\epsilon$-approximation algorithm for the k-center clustering problem of point sets in R^d space.*

References

[1] M. Bădoiu, S. Har-Peled, P. Indyk. "Approximate Clustering via Core-Sets," *Proceedings of the 34th ACM Symposium on Theory of Computing*, pp. 250-257, 2002.

[2] M. Bern and D. Eppstein. "Approximation algorithms for geometric problems," In *Approximation algorithms for NP-Hard problems*, D.S. Hochbaum (editors), PWS Publishing Company, 1997.

[3] T. Feder and D. H. Greene. "Optimal algorithm for approximation clustering," *Proc. 20th ACM Symp. Theory of Computing,* pages 434-444, 1988.

[4] T. Gonzalez. "Clustering to minimize the maximum intercluster distance," *Theoritical Computer Science*, Vol. 38, pages 293-306, 1985.

[5] P. Vaidya. "An $O(n\log n)$ algorithm for the all-nearest-neighbors problem," *Disc. and Comp. Geometry*, Vol. 4, pages 101-115, 1989.

[6] V. V. Vazirani. "Approximation Algorithms," Springer, 2001.

Analysis of Layered Hierarchy for Necklaces

Kathryn Bean* Sergey Bereg*

1 Introduction

Molecular configurations can be modeled as sets of spheres in 3D. By imposing geometric constraints on the spheres based on molecular biology one can apply efficient techniques [3], for example, to analyze the complexity of molecular surface [1]. Guibas *et al.* [2] introduced a model called *necklace* that finds applications in computer graphics, computer vision, robotics, geographic information systems and molecular biology. They studied two data structures, *wrapped hierarchy* and *layered hierarchy*, for representing necklaces and for performing collision tests.

Guibas *et al.* [2] proved that the wrapped hierarchy admits a separating family of size $O(n^{2-2/d})$. This is the first subquadratic bound proved for collision detection using predefined hierarchies. Although the layered hierarchy can be used for collision detection, "no subquadratic bound on the size of a separating family based on the layered hierarchy is currently known" [2]. The main result of this paper is that the same upper bound holds for the layered hierarchy.

One of the advantages of layered hierarchy over wrapped hierarchy is the "local" definition of the cages. We propose a modification of the layered hierarchy so that some deformations of a necklace can be maintained efficiently. In particular, we show that *rigid-body conformational changes* can be handled in $O(\log n)$ time only.

2 Necklaces and Bounding-Volume Hierarchies

We define the notion of necklace in a slightly more general form[1].

Definition 1 (Necklace) *A* necklace *is a sequence of beads $\mathcal{N} = \langle B_1, B_2, \ldots, B_n \rangle$ in $\mathbb{R}^d$ space that has the following properties:*

1. *The radius of each bead is in the interval $[\rho_{\min}, \rho_{\max}]$ where $\rho_{\min}, \rho_{\max}$ are positive constants.*
2. *The distance between the centers of two adjacent beads is bounded by a constant δ.*

Collision detection problem arises in such applications as protein folding and protein docking. A bounding volume hierarchy is a common approach that models and detects collisions and self-collisions of different geometrical shapes including necklaces. This approach reduces computational time since it suffices to test collisions among bounding volumes.

Definition 2 (Hierarchies) *For a necklace $\mathcal{N}$, let $T(\mathcal{N})$ denote the balanced binary tree defined recursively so that, for an internal node v, its left subtree contains $\lfloor m/2 \rfloor$ leaves where m is the number of leaves below v. Both wrapped and layered hierarchies have a* cage *$C(v)$ associated with a node v of $T(\mathcal{N})$ and the cages of leaves correspond to the beads, $C(v) = B_i$ for i-th leaf v where $i = 1, 2, \ldots, n$. For an internal node v, the cage is defined differently for two hierarchies:*

- Wrapped hierarchy. *The cage $C(v)$ is the smallest enclosing ball of beads corresponding to the leaves below v.*
- Layered hierarchy. *The cage $C(v)$ is the smallest enclosing ball of the cages corresponding to the children of v.*

The layered hierarchy has many advantages over the wrapped counterpart. For a necklace size n, the layered hierarchy can be constructed in linear time since the cage of each node can be computed in $O(1)$ time. The wrapped hierarchy can be constructed in $O(n \log n)$ time by using linear-time algorithm [4] for computing the minimum enclosing sphere MES. A simpler algorithm for computing MES in expected $O(n)$ time can be used. Note that the algorithm for the layered hierarchy is even simpler.

3 Necklace Deformation with Layered Hierarchy

Modeling conformation changes is required in protein docking, robotics and computer graphics. One

*Department of Computer Science, University of Texas at Dallas, Box 830688, Richardson, TX 75083, USA.

[1] We do not require that any two consecutive beads along the necklace have a point in common.

type of local deformation - *rigid-body conformational change* [5] - can be defined as follows.

Definition 3 (Conformational Change) *Let $\mathcal{N} = \langle B_1, B_2, \ldots, B_n \rangle$ be a necklace and let i be an index from 1 to $n-1$. Let M be a rigid motion in $\mathbb{R}^3$ (the composition of a translation and a rotation). If $\mathcal{N}' = \langle B_1, \ldots, B_i, M(B_{i+1}), \ldots, M(B_n) \rangle$ is a necklace then $\mathcal{N}'$ is a rigid-body conformational change of $\mathcal{N}$.*

We show how to modify the layered hierarchy so that a rigid-body conformational change can be done efficiently. We store a rigid motion $M(v)$ associated with a vertex v of $T(\mathcal{N})$. Each rigid motion $M(v)$ is a composition of a 3D rotation $R(v)$ and a translation $T(v)$. The rotation can be represented as a quaternion or a rotation matrix. We assume that the inverse rigid motion $M^{-1}(v)$ can be computed in $O(1)$ time. We call the hierarchy augmented with rigid motions as *augmented layered hierarchy*.

The augmented layered hierarchy defines the position of a bead B_i in the space as follows. Let $v_1 = v_{\mathrm{root}}, v_2, \ldots, v_k$ be the path from the root of the layered hierarchy to the vertex with the bead B_i and let c_i be the center of the bead stored in v_k. Then the real position of B_i is $M_1(M_2(\ldots M_k(c_i)\ldots))$ where $M_j = M(v_j), j = 1, \ldots, k$. Let I denote the *identity transformation*, i.e. $I(p) = p$ for any $p \in \mathbb{R}^3$.

Theorem 1 *The augmented layered hierarchy can be maintained in $O(\log n)$ time if a rigid-body conformational change is applied.*

4 Cages of Layered Hierarchy

Although the layered hierarchy can support rigid-body conformational changes efficiently (if augmented as in previous section), the wrapped hierarchy occupies smaller space since its cages are always no larger that the corresponding cages of the layered hierarchy. Despite this evidence we show that the cages of the layered hierarchy have the same upper bound as the corresponding cages of the wrapped hierarchy.

Let T_u denote the subtree rooted at a vertex u of $T(\mathcal{N})$ and let $n_l(u)$ be the number of leaves in T_u.

Theorem 2 *The radius of any cage $C(v)$ of layered hierarchy $T(\mathcal{N})$ is at most $\delta(n_l(v)-1)/2 + \rho_{\max}$, where $n_l(v)$ is the number of leaves for a tree is rooted at v.*

5 Collision Detection for Layered Hierarchy

An useful tool for the collision testing is a *separating family* [2].

Definition 4 (Separating Family) *A* separating family $\Sigma = \{(u, v)\}$ *is a set of pairs of nodes of a volume bounding hierarchy satisfying the following properties:*

1. *If $(u, v) \in \Sigma$ then $C(u)$ and $C(v)$ are disjointed.*
2. *For any two non-adjacent beads $B_p, B_m \in N$, there is a pair $(u, v) \in \Sigma$ such that $B_p \subseteq C(u)$ and $B_m \subseteq C(v)$.*

To derive a bound for the layered hierarchy we analyze a separating family Σ built by the following algorithm for collision detection. The algorithm starts with the pair $Q = \{(root, root)\}$ and, for a pair $(u, v) \in Q$ such that $C(u) \cap C(v) \neq \emptyset$, replaced it by at most 4 pairs of children of u and v. Our analysis is based on the analysis of the wrapped hierarchy by Guibas *et al.* [2].

Lemma 3 *If $(C(u), C(v)) \in \Sigma$ then the cage $C(v)$ is contained in $K - C(u)$ where K is the ball concentric with $C(u)$ and of radius $4(\delta n_l(u) + \rho_{\max})$.*

Theorem 4 *Let $\mathcal{H}_L$ be the layered hierarchy for a necklace $\mathcal{N}$ in $\mathbb{R}^d, d \geq 3$ with n beads. Algorithm 2 builds the unique separating family Σ for $\mathcal{N}$ of size $O(n^{2-2/d})$. This bound is asymptotically tight in the worst case.*

References

[1] Y.-E. A. Ban, H. Edelsbrunner, and J. Rudolph. Interface surfaces for protein-protein complexes. In *Proc. of the 8th Annual Internat. Conf. on Research in Computational Molecular Biology (RECOMB'04)*, pp. 205–212, 2004.

[2] L. J. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. *Comput. Geom. Theory Appl.*, 28(2-3):137–163, 2004.

[3] D. Halperin and M. H. Ovemars. Spheres, molecules, and hidden surface removal. *Comput. Geom. Theory Appl.*, 11(2):83–102, 1998.

[4] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pp. 329–338, 1982.

[5] P. Sompornpisut, Y. Liu, and E. Perozo. Calculation of rigid-body conformational changes using restraint-driven cartesian transformations. *Biophys Journal*, 81(5):2530–2546, 2001.

Author Index