December 30, 1994

SRC Research Report **133a**

The Third Annual Video Review of Computational Geometry

Edited by Marc H. Brown and John Hershberger

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Systems Research Center

The charter of SRC is to advance both the state of knowledge and the state of the art in computer systems. From our establishment in 1984, we have performed basic and applied research to support Digital's business objectives. Our current work includes exploring distributed personal computing on multiple platforms, networking, programming technology, system modelling and management techniques, and selected applications.

Our strategy is to test the technical and practical value of our ideas by building hardware and software prototypes and using them as daily tools. Interesting systems are too complex to be evaluated solely in the abstract; extended use allows us to investigate their properties in depth. This experience is useful in the short term in refining our designs, and invaluable in the long term in advancing our knowledge. Most of the major advances in information systems have come through this strategy, including personal computing, distributed systems, and the Internet.

We also perform complementary work of a more mathematical flavor. Some of it is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. Other work explores new ground motivated by problems that arise in our systems research.

We have a strong commitment to communicating our results; exposing and testing our ideas in the research and development communities leads to improved understanding. Our research report series supplements publication in professional journals and conferences. We seek users for our prototype systems among those with whom we have common interests, and we encourage collaboration with university researchers.

Robert W. Taylor, Director

The Third Annual Video Review of Computational Geometry

Edited by Marc H. Brown and John Hershberger

December 30, 1994

Author Affiliation

John Hershberger is currently employed by Mentor Graphics. He can be reached at `jeh@wv.mentorg.com`.

©Digital Equipment Corporation 1994

The copyright of each section in this report is held by the author of the section. Similarly, the copyright of each segment of the accompanying videotape is held by the contributor of the segment. All other material in the report is copyrighted ©Digital Equipment Corporation 1994. This work may not be copied or reproduced in whole or in part for any commercial purpose without permission of the authors and individual contributors.

Abstract

Computational geometry concepts are often easiest to understand visually, in terms of the geometric objects they manipulate. Indeed, most papers in the field rely on diagrams to communicate the intuition behind their results. However, static figures are not always adequate.

The accompanying videotape showcases advances in the use of algorithm animation, visualization, and interactive computing in the study of computational geometry. This report contains brief descriptions of all the segments of the videotape.

This report is the third annual Video Review of Computational Geometry to be published as a SRC Research Report. The '92 Video Review is SRC Research Report 87, and the '93 Video Review is SRC Research Report 101. Future editions will be published by the ACM, as part of the proceedings of the annual ACM Symposium on Computational Geometry.

Preface

Computational geometry concepts are often easiest to understand visually. Indeed, most papers in computational geometry rely on diagrams to communicate the intuition behind their results. However, static figures are not always adequate to describe geometric algorithms, since algorithms are inherently dynamic. The accompanying videotape showcases advances in the use of algorithm animation, visualization, and interactive computing in the study of computational geometry.

This report contains short descriptions of the eight segments in the videotape. The first three segments are animations of two-dimensional algorithms: a polygonal approximation algorithm from cartography, an algorithm for computing the rectangle discrepancy, and a fixed-radius neighbor-searching algorithm. The fourth segment presents the GASP system for animating geometric algorithms in two or three dimensions; GASP is used by two other segments in the review. The remaining segments illustrate algorithms and concepts in three or more dimensions: the fifth segment explores the relationship between penumbral shadows in three dimensions and four-dimensional polytopes; the final three segments show three-dimensional algorithms for interactive collision detection, polyhedral separators, and interactive display of alpha hulls.

This report and the accompanying videotape comprise the Third Annual Video Review of Computational Geometry. In the three years since the Video Review was introduced, it has become an established part of the annual ACM Symposium on Computational Geometry. The 1995 Video Review and future editions will be distributed by the ACM as part of the conference proceedings.

We thank Bob Taylor, director of the Systems Research Center, for his support in establishing the annual Video Review. The '92 Video Review is available as SRC Research Report 87, and the '93 Video Review as SRC Research Report 101. These reports (written guides and videotapes) are available by sending electronic mail to `src-report@pa.dec.com`.

Finally, we thank the members of the 1994 Video Program Committee for their help in evaluating the entries. The members of the committee were Marshall Bern (Xerox PARC), Marc Brown (Digital Equipment Corporation's Systems Research Center), Leo Guibas (Stanford University), John Hershberger (Mentor Graphics), Jim Ruppert (NASA Ames), and Jenny Zhao (Silicon Graphics).

Marc H. Brown
John Hershberger

Video Review Co-Chairs

Table of Contents

An $O(n \log n)$ Implementation Of The Douglas-Peucker Algorithm For Line Simplification

John Hershberger

Mentor Graphics
1001 Ridder Park Drive
San Jose, CA 95131

Jack Snoeyink

Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z2 Canada

1

The "line simplification problem" is as follows: Given a sequence of vertices that make up a polygonal chain, find a subsequence of the vertices that approximates the chain. The accompanying videotape shows two algorithms for solving the line simplification problem.

Introduction

An important task of the cartographer's art is to extract features from detailed data and represent them on a simple and readable map. As computers become increasingly involved in automated cartography, efficient algorithms are needed for the tasks of extraction and simplification. Cartographers [1, 6] have identified the *line simplification problem* as an important part of representing linear features. Given a *polygonal chain*, a sequence of vertices $V = \{v_0, v_1, \ldots, v_n\}$, the problem is to find a subsequence of vertices that approximates the chain V. We assume that the input chain V has no self-intersections (but not the output).

In 1973, Douglas and Peucker (now Poiker) [3] published a simple recursive simplification method in the cartography literature that has become the favorite of many [5, 10]. This method has been independently proposed in other contexts such

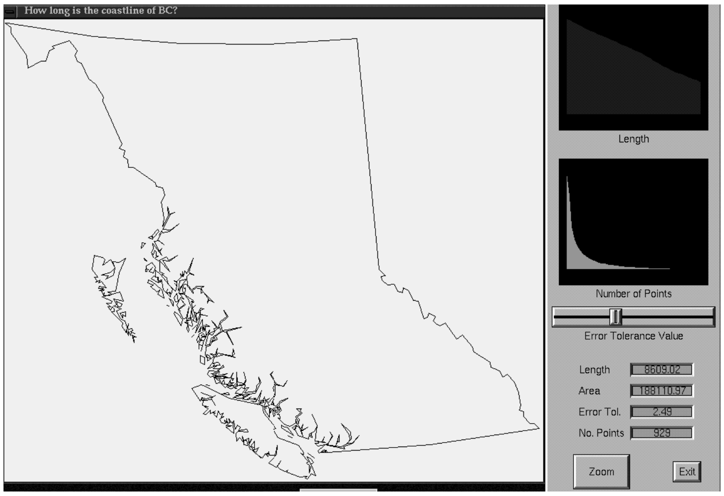


Figure 1: Simplifying the polygonal chain representing the coastline of British Columbia.

as vision [8] and computational geometry [9]. The accompanying videotape animates two different implementations of this simplification method—the straightforward implementation suggested by Douglas and Peucker and the implementation of Hershberger and Snoeyink that improves the worst-case running time.

Method

Given a polygonal chain $V = \{v_0, v_1, \ldots, v_n\}$, the recursive line simplification method proceeds as follows: Initially, approximate V by the line segment $\overline{v_0 v_n}$. Determine the farthest vertex v_f from the line $\overleftrightarrow{v_0 v_n}$. If its distance $d(v_f, \overleftrightarrow{v_0 v_n})$ is at most a given tolerance $\varepsilon \geq 0$, then accept the segment $\overline{v_0 v_n}$ as a good approximation to V. Otherwise, break V at v_f and recursively approximate the subchain $\{v_0, v_1, \ldots, v_f\}$ and $\{v_f, \ldots, v_n\}$.

Algorithms

The two implementations differ in the way they find v_f, the farthest vertex from $\overleftrightarrow{v_0 v_n}$.

The straightforward approach measures distance from $\overleftrightarrow{v_0 v_n}$ for each vertex and takes the maximum. Thus, the running time of this algorithm will satisfy a quicksort-like recurrence, with best case $\Theta(n \log n)$. Since geometry determines the farthest vertex, the worst-case running time is $\Theta(n^2)$. One cannot use linear-time median finding or randomization to improve the running time. (If one assumes that the input data is such that farthest vertices are found near the middle of chains, then one expects $O(n \log n)$ behavior.)

The second implementation uses a *path hull data structure* [2, 4] to maintain a dynamic convex hull of the polygonal chain V. Using Melkman's convex hull algorithm [7], we compute two convex hulls from the middle of the chain outward. We can find a farthest vertex v_f from a line by locating two extreme points on each hull using binary search. When we split V at vertex v_f, we undo one of the hull computations to obtain the hull from the "middle" to v_f, recursively approximate the subchain containing the middle, then build hulls for the other subchain and recursively approximate it. One can use a simple credit argument to show that the total time taken by the algorithm is $O(n \log n)$. The best case can even be linear, if all the hulls are small and splits occur at the ends.

Unfortunately for us, the statistical properties of cartographic data usually mean that the straightforward implementation is slightly faster than the path hull implementation. Since the variance of the running time of the path hull implementation is smaller, it may be interesting for parallel or interactive applications.

Technical Details

The programs were run on a Silicon Graphics Crimson in the GraFiC lab at UBC and output was recorded to video in real time.

Acknowledgments

Scott Andrews extracted the data of the coastline of British Columbia from the Digital Chart of the World. NSERC and the B.C. Advanced Systems Institute supported the work on this video. DEC Systems Research Center has also provided partial support for the research.

1

References

[1] B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.

[2] D. Dobkin, L. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Algorithmica*, 10:1–23, 1993.

[3] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[4] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Intl. Symp. Spatial Data Handling*, pages 134–143. IGU Commission on GIS, 1992.

[5] R. B. McMaster. A statistical analysis of mathematical measures for linear simplification. *Amer. Cartog.*, 13:103–116, 1986.

[6] R. B. McMaster. Automated line generalization. *Cartographica*, 24(2):74–111, 1987.

[7] A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Info. Proc. Let.*, 25:11–12, 1987.

[8] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Comp. Vis. Graph. Image Proc.*, 1:244–256, 1972.

[9] G. Rote. Quadratic convergence of the sandwich algorithm for approximating convex functions and convex figures in the plane. In *Proc. 2nd Can. Conf. Comp. Geom.*, pages 120–124, Ottawa, Ontario, 1990.

[10] E. R. White. Assessment of line-generalization algorithms using characteristic points. *Amer. Cartog.*, 12(1):17–27, 1985.

Computing the Rectangle Discrepancy

David P. Dobkin and Dimitrios Gunopulos

Department of Computer Science
Princeton University
Princeton, NJ 08544

2

Supersampling is one of the most general ways of attacking the problem of antialiasing in computer graphics. In this approach, a picture is sampled at a very high rate, and then resampled by averaging supersamples within the pixel area to compute the pixel value. Ray tracing, a common technique used to produce realistic images of computer modeled scenes, is another instance of supersampling. Discrepancy theory provides a measure of the quality of the sampling patterns. The "rectangle discrepancy" in particular gives a good measure of how well a sample pattern, when applied to the area of one or more pixels, captures small details in the picture. The accompanying videotape shows an animation of an algorithm we have developed [2] for computing the "rectangle discrepancy" of a two-dimensional point set. In particular, this algorithm can be used to find sampling sets with very low rectangle discrepancy. Interestingly the problem of computing the rectangle discrepancy also arises in the context of learning theory [4].

Definitions

Let us consider a point set $\mathcal{X}$ in the unit square. Let $\mathcal{F}$ be the set of all rectangles in the unit square with edges parallel to the two axes. For an axis-oriented rectangle A ($A \in \mathcal{F}$), we define the function

$$In(A) = |A \cap \mathcal{X}|$$

2

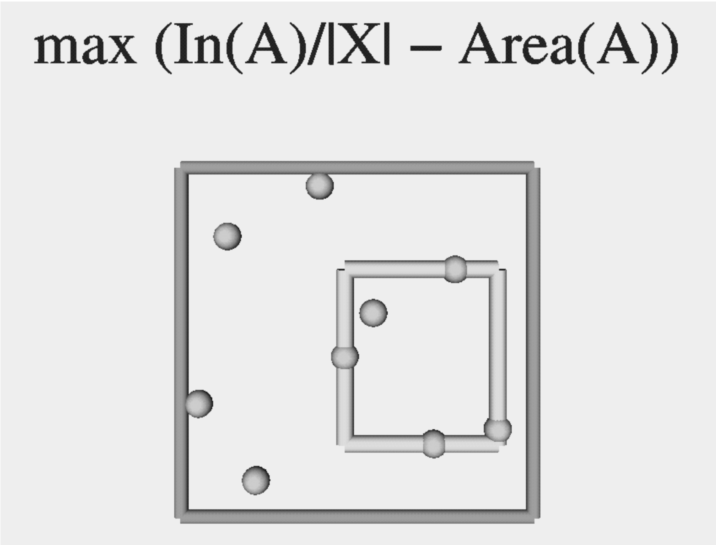


Figure 2: The rectangle that maximizes $\mathcal{D}_l$.

The *rectangle discrepancy* of a rectangle $A \in \mathcal{F}$ is

$$\mathcal{D}(A) = |In(A)/|\mathcal{X}| - Area(A)|$$

The *maximum rectangle discrepancy* of the set $\mathcal{X}$ is

$$Max\mathcal{D}(\mathcal{X}) = max_{A \in \mathcal{F}}(\mathcal{D}(A))$$

We also define the following simpler functions for $A \in \mathcal{F}$

$$\mathcal{D}_l(A) = In(A)/|\mathcal{X}| - Area(A)$$

and

$$\mathcal{D}_o(A) = Area(A) - In(A)/|\mathcal{X}|$$

The algorithm we present in this animation computes

$$Max\mathcal{D}_l(\mathcal{X}) = max_{A \in \mathcal{F}}(\mathcal{D}_l(A))$$

for a given input set $\mathcal{X}$, in time $O(|\mathcal{X}|^2 \log |\mathcal{X}|)$. The same techniques however can be used to maximize $\mathcal{D}_o$, and so this problem is equivalent to finding $Max\mathcal{D}(\mathcal{X})$.

2

Videotape Contents

We explain the main idea of the algorithm in a series of simple lemmata. First we show that we have to consider at most $O(|\mathcal{X}|^4)$ rectangles, more specifically the ones that have all their edges pass through a point in $\mathcal{X}$.

This allows us to study pairs of points in in $\mathcal{X}$. For each such pair we find the rectangle that maximizes $\mathcal{D}_l$ over all rectangles with horizontal edges that pass through the points of the given pair.

This problem is much easier now because we know the y coordinates of the rectangle. Assuming that the two points are p_b, p_t ($p_b(y) < p_t(y)$), and the height of the rectangle is dy, ($dy = p_t(y) - p_b(y)$) we have to find the following maximum

$$max_{A \in \mathcal{F}_{ij}}(In(A)/|\mathcal{X}| \; - \; dx_A \; dy)$$

where $\mathcal{F}_{ij}$ is the set of rectangles that pass through points p_b and p_t, and dx_A is the width of A. If we eliminate the points in $\mathcal{X}$ that either lie above the top y coordinate of the rectangle or below the lower y coordinate, this becomes a one-dimensional problem. We project the points on the x axis and obtain a new set $\mathcal{X}'$ of one-dimensional points. We can equivalently maximize the following function

$$max_{A \in [0,1], p_b(x) \in A}(In(A)/|\mathcal{X}'| \; - \; Length(A))$$

We can reduce this problem to the problem of computing the halfspace discrepancy of one-dimensional point sets, a simpler problem studied in [1]. Here we want to find the anchored interval that maximizes

$$max_{A=[0,z_i]}(i/|\mathcal{Z}| \; - \; z_i)$$

for a given one-dimensional set $\mathcal{Z} = \{z_1 \ldots z_n\}$ with $0 \leq z_i < z_{i+1} \leq 1$. This is a linear function of the rank and the coordinate of the point. To solve it we find the convex hull of the set $\{(z_1, 1) \ldots (z_n, n)\}$ and find the maximum with a binary search. From this maximum we obtain the solution to the original maximization problem.

In this way we compute the maximum rectangle for all pairs of points. Clearly the rectangle that maximizes $\mathcal{D}_l$ is the maximum of these $O(|\mathcal{X}|^2)$ rectangles.

Finally, we show how to use the dynamic data structure of [5] to solve a sequence of one-dimensional problems efficiently. With the use of this technique we can maintain the convex hull at a cost of $O(\log^2 |\mathcal{X}|)$ time per insertion. This in turn allows us to find the maximum rectangle for each pair of points in $O(\log^2 |\mathcal{X}|)$ time, and yields the final running time of $O(|\mathcal{X}|^2 \log^2 |\mathcal{X}|)$.

2

Technical Details

The accompanying videotape was prepared in the Computer Science Department at Princeton University. The implementation of the algorithm was written in C, to run under UNIX on a Silicon Graphics Iris workstation. The animation was created with the animation system GASP that Ayellet Tal and David Dobkin are developing in this department [3]. Recording was done at the Interactive Computer Graphics Lab at Princeton and editing was done with the assistance of the Department of Media Services at Princeton.

Acknowledgments

This work supported in part by the National Science Foundation under Grant Number CCR93-01254 and by The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

References

[1] D.P. Dobkin and D. Eppstein, Computing the Discrepancy. *Proceedings of the Ninth Annual Symposium on Computational Geometry,* (1993).

[2] D.P. Dobkin and D. Gunopulos, Computing the Rectangle Discrepancy. *Princeton University Technical Report 443-94.*

[3] D.P. Dobkin and A. Tal, GASP – A System to Facilitate Animating Geometric Algorithms. *Third Annual Video Review of Computational Geometry, to appear,* (1994)

[4] W. Maass, Efficient Agnostic PAC-Learning with Simple Hypotheses. *Submitted to COLT '94.*

[5] M. Overmars and J. van Leeuwen, Maintenance of Configurations in the Plane. *J. Comput. Sys. Sci.* 23 (1981) 166-204.

An Animation of a Fixed-Radius All-Nearest Neighbors Algorithm

Hans-Peter Lenhof and Michiel Smid

Max-Planck-Institut für Informatik
D-66123 Saarbrücken, Germany

3

A fixed-radius all-nearest neighbors problem takes as input a set S of n points and a real number δ, and reports all pairs of points that are at distance at most δ. The accompanying videotape shows an animation of an algorithm, developed by the authors [2], for solving the fixed-radius all-nearest neighbors problem.

The Algorithm

The algorithm depicted in the videotape works in two stages. In the first stage, a grid is computed. Instead of a standard grid, we use a so-called degraded grid that is easier to construct by means of a simple sweep algorithm. This degraded grid consists of boxes with sides of length at least δ. If a box contains points of S, then its sides are of length exactly δ. In the second stage, this grid is used for the actual enumeration. For each non-empty box, it suffices to compare each of its points with all points in the same box or in one of the neighboring boxes. Although the algorithm may compare many pairs having distance more than δ, it can be shown that the total number of pairs considered is proportional to the number of pairs that are at most δ apart. As a result, the total running time of the algorithm is proportional to $n \log n$ plus the number of pairs that are at distance at most δ.

As an application, we give an animation of the algorithm of Heiden et al. [1] for triangulating the contact surface of a molecule. In a first step, points on this surface are computed. Given these points, we compute all pairs that are at distance at most

δ for a suitable choice of δ. This gives for each point a neighbor list containing all points that are at distance at most δ from it. Given these lists, we can triangulate the surface incrementally: An edge of the current triangulation is called an outer edge, if it belongs to only one triangle. All outer edges are maintained in a queue. With each edge, we store a so-called common neighbor list, which is the intersection of the two neighbor lists of the points belonging to this edge. We start with any triangle. Its edges and their common neighbor lists are inserted into the queue. As long as the queue is non-empty, we do the following: We take any edge from the queue and look at all points in its common neighbor list. Each such point defines a triangle with the current edge. We take that point defining the smallest triangle. This triangle becomes part of the triangulation and we update the queue appropriately.

Technical Details

The video was produced at the Max-Planck-Institute for Computer Science, using Silicon Graphics Indy and Indigo^2 machines, and using Showcase and Geomview-1.1 software. We thank Roland Berberich, Jack Hazboun, Dirk Louis and Peter Müller for their help in producing this video.

Acknowledgments

This work was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

References

[1] W. Heiden, M. Schlenkrich and J. Brickmann. *Triangulation algorithms for the representation of molecular surface properties.* J. Comp. Aided Mol. Des. **4** (1990), pp. 255-269.

[2] H.P. Lenhof and M. Smid. *Enumerating the k closest pairs optimally.* Proceedings FOCS 1992, pp. 380-386.

GASP - A System to Facilitate Animating Geometric Algorithms

Ayellet Tal and David Dobkin

Department of Computer Science
Princeton University
Princeton, NJ 08544

4

The accompanying videotape shows five algorithm animations that have been developed using GASP. The GASP system is noteworthy because it is very easy for a user without any knowledge of computer graphics to create an animation of a geometric algorithm quickly.

Overview

The GASP system helps in the creation and viewing of animations for geometric algorithms. The user need not have any knowledge of computer graphics in order to quickly generate an animation. GASP allows the fast prototyping of algorithm animations. A typical animation can be produced in a matter of days or even hours. Even highly complex geometric algorithms can be animated with ease. The system is also intended to facilitate the task of implementing and debugging geometric algorithms.

The application's code specifies only the structure of the animation (which building blocks are included), and the structure of each scene (e.g., the position of the objects). The user need not be concerned with the way the animation appears on the screen. If the user wishes to influence the look of the animation and not only its structure, he can edit an ASCII *"Style File"* which controls viewing aspects of the animation. Even when the user changes the style file, the user needs no specific

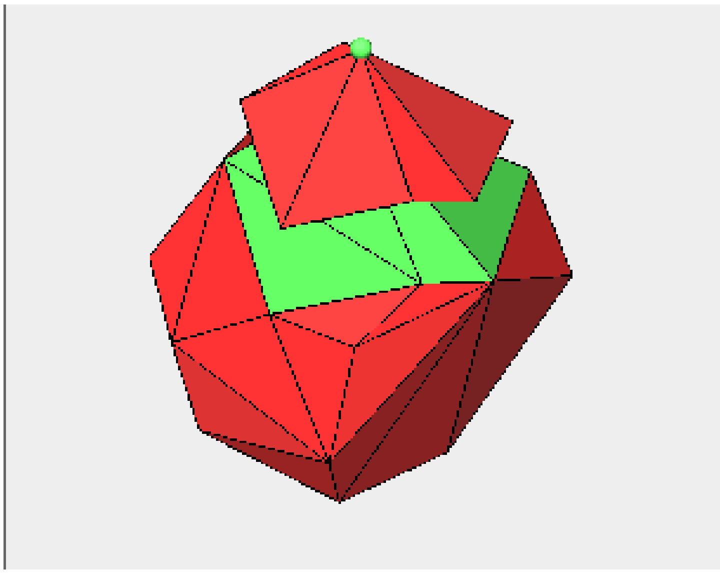


Figure 3: Polyhedral Hierarchies

knowledge of computer graphics. The animation is generated automatically by the system. But a different animation will be generated if the style file is modified.

The GASP environment allows the viewer to control the execution of the algorithm in an easy way. The animation can be run at varying speeds: fast($>>$), slow($>$) or step by step ($> \ |$). The analogous $<$, $<<$ and $| \ <$ push buttons run the algorithm in reverse. The viewer can *PAUSE* at any time to suspend the execution of the algorithm or can *EJECT* the movie. The environment is invaluable for education and debugging.

Videotape Contents

This video shows five examples of GASP in action. These examples were chosen to give a sense of the range of possible applications of GASP. In addition, two other videos [1], [3] that use our system appear in this collection of animations.

4

Clip 1: Polyhedral Hierarchies

To begin, we show a clip from [6]. This is the hierarchical data structure for representing convex polyhedra [4, 5], and the animation was an aid in debugging the algorithm for detecting plane-polyhedral intersection. We explain a single pluck and then show how the hierarchy progresses from level to level. To create the animation for explaining a single pluck the user need only write the following short code:

```
explain_pluck(poly_vert_no, poly_vertices,
  poly_face_no, poly_faces, poly_names,
  vert_no, vertices, face_no, faces)
{
  /* create the polyhedron */
  Begin_atomic("poly");
  Create_polyhedron("P0",
    poly_vert_no, poly_face_no, poly_vertices, poly_faces);
  Rotate_world();
  End_atomic();

  /* remove vertices and cones */
  Begin_atomic("pluck");
  Split_polyhedron(poly_names, "P0", vert_no, vertices);
  End_atomic();

  /* add new faces */
  Begin_atomic("add_faces");
  Add_faces(poly_names[0], face_no, faces);
  End_atomic();

  /* undo plucking */
  Undo(2);
}
```

4

Clip 2: Objects That Cannot Be Taken Apart With Two Hands

The second animation, based on [10], is a remake of [9]. The animation shows a configuration of six tetrahedra that cannot be taken apart by translation with two hands. Then, it presents a configuration of thirty objects that cannot be taken apart by applying an isometry to any proper subset. The purpose of the animation is to illustrate the use of GASP as an illustration tool for geometric configurations. It took us far less than a day to generate that animation. The animation was created by the following function:

```
hands(){
  for (i = 0; i < stick_no; i++){
    /* stick i */
    get_polyhedron(&points,
      &indices, &nmax, &fmax, &atomic_name, &stick_name);

    Begin_atomic(atomic_name);
    Create_polyhedron(stick_name, nmax, fmax, points, indices);
    End_atomic();
  }

  Begin_atomic("Rotate");
  Rotate_world();
  End_atomic();
}
```

The following is part of the style file for the above animation. The style file determines the following aspects of the animation. The background color is light gray. The colors to be chosen by GASP are colors which fit the creation of a video (rather than the screen). Each atomic unit spans 10 frames. Rotation of the world is done 380 degrees around the Y axis, and it is executed over 160 frames, instead of over 10. The color of the stick which fades in during atomic unit `stick0` is red. (Similarly, the style file includes colors for the other sticks.)

```
begin_global_style
  background = 0.9 0.9 0.9;
  color = VIDEO;
  frames = 10;
end_global_style

begin_unit_style Rotate
  frames = 160;
  rotation_world = Y 380.0;
end_unit_style

begin_unit_style stick0
  color = 0.625 0.273475 0.273475;
end_unit_style
```

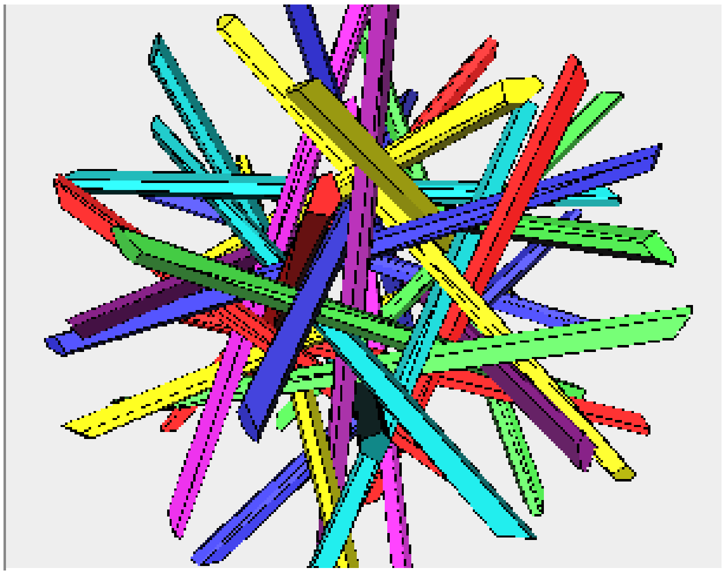


Figure 4: Objects That Cannot Be Taken Apart With Two Hands

Clip 3: Motion Planning

Next we show an animation of a motion planning algorithm [7]. It is a repeat of the first part of [8]. The object to be moved is a disc and the obstacles are simple disjoint polygons. The goal of the animation is to show the use of GASP in creating two-dimensional animations.

Clip 4: Line Segment Intersection

The fourth animation, which is based on [2], shows a line segment intersection algorithm in action and illustrates its most important features. The animation consists of three phases. The first phase presents the initial line segments and the visibility map that needs to be built. The second phase demonstrates that the visibility map is being constructed by operating in a sweepline fashion, scanning the segments from left to right, and maintaining the vertical visibility map of the region swept along the way. The third phase demonstrates that the cross section along the sweepline is maintained in a lazy fashion. The animation is used as an aid in explaining a highly complex algorithm and to allow students to interact with the algorithm. A student can choose the initial line segments by editing an ASCII file and view how the animation is changing.

Clip 5: Heapsort

The last clip animates heapsort. The colored rods have lengths to be sorted. We first display the initial creation of the heap. Next, simple motions are used to control the repeated operations of removing the largest element, and reheaping. Heapsort is given as an example of using GASP to facilitate the animation of any algorithm that involves the display of three dimensional geometries.

Technical Details

This video was prepared in the Computer Science Department at Princeton University. The programs were written in C to run under UNIX on a Silicon Graphics Iris. Recording was done at the Interactive Computer Graphics Lab at Princeton and editing was done with the assistance of the Princeton Department of Media Services.

Acknowledgments

We thank Kirk Alexander and Mike Mills for their help in producing the final video. This work supported in part by the National Science Foundation under Grant Number CCR93-01254 and by The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

References

[1] H. Bronnimann. Almost optimal polyhedral separators. In *The Third Annual Video Review of Computational Geometry*, 1994.

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

[3] D. Dobkin and D. Gunopulos. Computing the rectangle discrepancy. In *The Third Annual Video Review of Computational Geometry*, 1994.

[4] D. Dobkin and D. Kirkpatrick. Fast detection of polyhedral intersections. *Journal of Algorithms*, 6:381–392, 1985.

[5] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. *ICALP*, pages 400–413, 1990.

[6] D. Dobkin and A. Tal. Building and using polyhedral hierarchies (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, page 394, May 1993.

[7] H. Rohnert. Moving a disc between polygons. *Algorithmica*, 6:182–191, 1991.

[8] S. Schirra. Moving a disc between polygons (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, pages 395–396, May 1993.

[9] J. Snoeyink. Objects that cannot be taken apart with two hands (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, page 405, May 1993.

[10] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. In *The Ninth Annual ACM Symposium on Computational Geometry*, pages 247–256, May 1993.

Penumbral Shadows

Adrian Mariano
Center for Applied Math
Cornell University
Ithaca, NY 14853–3801

Linus Upson
NeXT
900 Chesapeake Drive
Redwood City, CA 94063

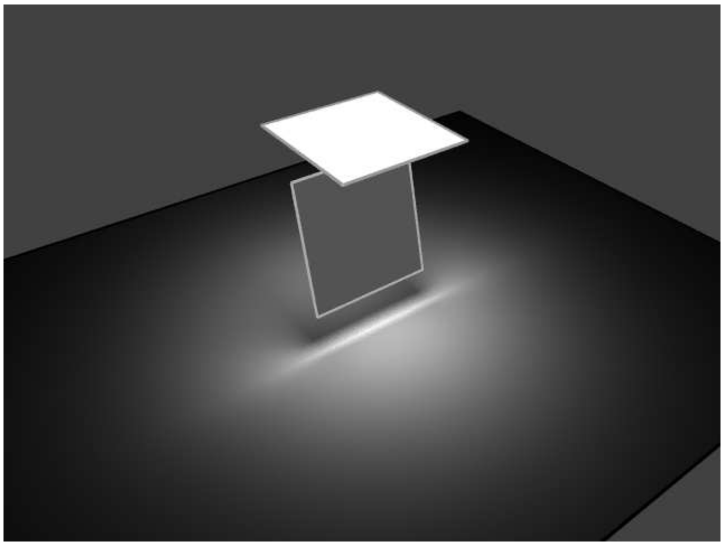
5

The shadows cast by polygonal occluding objects illuminated by polygonal light sources are complicated. Unlike shadows cast by point sources, they are divided into regions where the shadow intensity changes at different rates. The boundaries between these different regions form the *shadow diagram*.

The accompanying videotape depicts shadows cast by a square light source with a square occluding object. We have included two types of images: flat shadow scenes and perspective views. The flat shadow images were calculated using the exact formula for radiative energy transfer. This calculation depends only on the boundary of the visible light source. For polygons, it can be reduced to a simple summation [1].

Finding the boundary of the visible portion of the illuminating polygon when it is occluded by a second polygon requires computing the intersection of two polygons. General purpose algorithms for computing intersections of arbitrary polygons in the plane are quite complicated, so we used the Sutherland–Hodgman algorithm, which cannot handle concave polygons, but which is much simpler than the fully general methods [2]. We created the three-dimensional perspective views from the flat images using the texture mapping capabilities of *Rayshade*.

The shadow diagrams can be obtained by projecting the edges or vertices of the light source through vertices or edges of the occluder onto the plane of the shadow. The video illustrates that when a square occluding object is illuminated by a square light source, the shadow diagram is a projection of a four-dimensional hypercube.



In general, the shadow diagrams for an arbitrary light source and occluder can be seen as projections of four-dimensional polytopes by considering the lines which intersect the plane of the shadow. Let S be the plane of the shadow and let P be a different plane parallel to S. If a line intersects S at the point (x_s, y_s) and it intersects P at the point (x_p, y_p) then the line can be mapped to a point (x_s, y_s, x_p, y_p) in $\mathbf{R}^4$. Now let D be the set of all lines which intersect both the light source and the occluder. When D is mapped into $\mathbf{R}^4$, the result is a four-dimensional polytope. If this polytope is projected into $\mathbf{R}^2$ by discarding the last two coordinates of each point, the shadow diagram results. Because P can be changed, this construction describes a particular shadow diagram as the projections of any one of a family of polytopes.

References

[1] Hottel and Sarofim. *Radiative Transfer.* McGraw Hill, New York. 1967.

[2] Vatti, Bala. "A Generic Solution to Polygon Clipping." *Communications of the ACM*, July 1992.

[3] Teller, Seth. "Computing the Antipenumbra of an Area Light Source." SIGGRAPH '92 Conference Proceedings, July 1992.

Exact Collision Detection for Interactive Environments

Jonathan D. Cohen
Dinesh Manocha
Madhav K. Ponamgi

Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175

Ming C. Lin

Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

6

The accompanying videotape demonstrates algorithms for collision detection in large-scaled, interactive environments. Such environments are characterized by the number of objects undergoing rigid motion and the complexity of the models. The algorithms do not assume that the motions of the objects are expressible as closed form functions of time, nor do they assume any upper bounds on their velocities.

Background

Over the last few years, a great deal of interest has been generated in *virtual or synthetic* environments, such as architectural walkthroughs, visual simulation systems for designing the shapes of mechanical parts, training systems, etc. Interactive, large-scaled virtual environments pose new challenges to the collision detection problem, because they require performance at interactive rates for thousands of pairwise intersection tests [1].

Our algorithms use a two-level hierarchical detection test for each model to *selectively* compute the *precise* contact between objects, achieving real-time performance without compromising accuracy. At the top level, we use a *dimension reduc-*

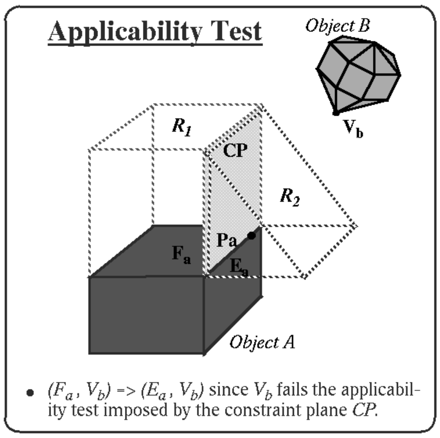


Figure 5: An example of collision detection for convex polyhedra

tion approach (based on a technique termed as *sweep and prune*) and the *temporal and geometric coherence* that exists between successive frames to overcome the bottleneck of $O(n^2)$ pairwise comparison tests in a large environment of n moving objects [1]. After we eliminate the object pairs that are not in the close vicinity of each other, we employ an exact collision detection scheme. The exact convex polytope collision detection algorithm is based upon [2]: locality (captured by walking from one Voronoi region of the candidate feature to one of its neighboring features' Voronoi regions according to the constraints failed) and coherence (used to reduce computation efforts in a dynamic environment). We can deal with non-convex objects or articulated bodies by using a sub-part hierarchical tree representation.

Exact Collision Detection

The Voronoi regions form a partition of space outside the polytope according to the closest feature. The collection of Voronoi regions of each polytope is the generalized Voronoi diagram of the polytope. Note that the generalized Voronoi diagram of a convex polytope has linear size and consists of polyhedral regions. Each Voronoi region is bounded by a set of constraint planes with pointers to the neighboring cells (which each share a constraint plane with it).

If a point lies on a constraint plane, then it is equidistant from the two features which share this constraint plane in their Voronoi regions. If a point P on object A lies inside the Voronoi region of the feature f_B on object B, then f_B is a closest feature to the point P.

Our method is straightforward. We start with a candidate pair of features, one from each polytope, and check whether the closest points lie on these features. Since the polytopes and their faces are convex, this is a local test involving only the neighboring features of the current candidate features. If either feature fails the test, we step to a neighboring feature of one or both candidates, and try again. With some simple preprocessing, we can guarantee that every feature has a constant number of neighboring features. This is how we can verify or update the closest feature pair in *expected constant* time. Fig. 5 shows how this algorithm works on a pair of simple convex polytopes.

Sweep and Prune

The exact polytope collision detection algorithm is fast enough to perform collision detection at real-time, in a multi-body environment of moderate size. However, the $O(n^2)$ pairwise detection tests become a bottleneck for large n. By exploiting the spatial and temporal coherence, we use a method whose running time is a linear function of the number of objects and the number of pairs of objects in close vicinity of each other.

For each object in the environment, we compute a smallest fitting, axis-aligned bounding box, which is dynamically resized using a hill-climbing algorithm based on the properties of convex polytopes and the principle of coherence.

These bounding boxes are used to prune down the number of pairwise collision detection tests. We take the orthogonal projections of the bounding boxes onto the x, y, and z axes and sort the intervals in three separate lists. If the projections overlap in *all three* dimensions, we activate the exact collision detection algorithm. At each frame, we sweep over the sorted lists of intervals, updating them and culling out the pairs of non-overlapping bounding boxes.

Each polytope in the environment now has a bounding box. We only call the exact collision detection algorithm between two objects when their bounding boxes overlap. Due to coherence, the sorted lists of intervals, constructed from the projections of the bounding boxes, change little from frame to frame, so we use a bubble or insertion sort to update these lists in expected linear time. Without the bounding box hierarchy, we would have to perform a quadratic number of pairwise exact detection tests.

6

Video Contents

We have successfully demonstrated the algorithm in an architectural walkthrough environment and a multi-body simulation system. For simulations consisting of thousands of models, the overall algorithm has exhibited linear time performance in practice.

We first demonstrate how the polytope collision detection algorithm [2] tracks the closest features between two convex polytopes in real time. We maintain and update the closest features using the Voronoi regions of the polytopes. Since the polytopes move only slightly between the frames, the temporal and geometric coherence is well preserved. Thus, the expected running time of this algorithm is constant, independent of the complexity of the polytopes.

This is followed by demonstrating the algorithm on an articulated body, namely a hand, represented by the union of multiple convex polytopes. To demonstrate the efficiency of the algorithm and effectiveness of the coherence, we keep track of all pairs of closest features between each finger and the nearby objects. However, using a subpart hierarchical tree representation as in [2] can eliminate the unnecessary computation.

The over all collision detection algorithm is tested on a walkthrough and a multi-body dynamic simulation environment. The video footage of walkthrough was taken from a user's **actual walkthrough experience**. The collision detection algorithm performs very well in large environments composed of hundreds of polytopes. The kitchen, like most realistic environments, is fairly static; therefore, we can sort the bounding boxes of the polytopes in sub-linear time at each instance (this requires a modified sorting algorithm which sorts only the moving objects). We see very little degradation of frame rate when we add the capability of collision detection to our walkthrough environment.

We further demonstrate a multi-body simulation with several complex objects in a moderately dense environment. The objects are placed in a bounded 3-dimensional simulation volume whose walls act as barriers. Unlike the walkthrough environment, *all* the objects move independently. The dynamics of multi-body simulation assumes elastic collision. In order to show the asymptotic time bound on the algorithm, we increase the number of possible interactions quadratically, but the frame time using our collision detection algorithm increases only linearly.

Acknowledgments

We are grateful to Dr. Fred Brooks and the walkthrough group at UNC Chapel Hill for the model of the kitchen.

D. Manocha was supported in part by a Junior Faculty Award and ARPA Contract #DAEA 18-90-C-0044.

References

[1] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. Interactive and exact collision detection for large-scaled environments. Technical Report TR94-005, Department of Computer Science, University of North Carolina, 1994.

[2] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California at Berkeley, December 1993. Department of Electrical Engineering and Computer Science.

Almost Optimal Polyhedral Separators

Hervé Brönnimann

Department of Computer Science
Princeton University
Princeton, NJ 08544

7

The accompanying videotape illustrates two deterministic polynomial time methods for finding a separator for two nested convex polyhedra in 3-dimensions.

1 Introduction

Let $Q \subseteq P$ be two convex polyhedra in $\mathbb{R}^3$. The problem of finding a separating convex polyhedron between P and Q with as few facets as possible is believed to be *NP*-hard [2], but one can find good approximations for it. Mitchell and Suri [4] cast the problem as a set cover problem. Let $\mathcal{H} = \mathcal{H}(Q)$ denote the set of supporting hyperplanes of Q. Let the *cover set system* be $(\mathcal{H}, \{\mathcal{H}_p : p \in \partial P\})$, where, for any point p, $\mathcal{H}_p$ consists of those hyperplanes in $\mathcal{H}$ for which p and Q lie on opposite sides. They show that a hitting set for the cover set system gives a subset of facets of Q whose supporting halfspaces define a polytope whose boundary lies between Q and P.

A polytope Q' between Q and P such that each facet of Q' is supported by a facet of Q is called *canonical*. The smallest (with respect to the number of facets) canonical polytope contained in P can be obtained from a minimal hitting set of this set system and is within 3 times the optimal separating polyhedron (in the number of faces). Therefore, in this video, we just show how to obtain an optimal canonical separator.

2 Video Contents

We first familiarize the viewer with the two polyhedra $Q \subseteq P$; Q is seen inside by means of transparency (see Fig. 6, top). We explain how to obtain canonical separators, and how to construct the set system. The portion of the arrangement $\mathcal{A}$ of $\mathcal{H}(Q)$ on the boundary of P is visualized, and some rows of the set system are then shown sequentially, one corresponding to each facet of this arrangement.

To explain the greedy method on this set system, an approach which is proposed by Mitchell and Suri [4], we show how the choice of a facet removes a portion of the boundary of P, and how the greedy method proceeds sequentially by choosing the facet which covers the biggest remaining portion (measured by the number of cells of $\mathcal{A}$ it separates from Q). After four iterations, we skip to the final greedy separator, which has 14 facets. In general, the greedy separator is guaranteed to have at most $s_{opt} \log |Q|$ facets, where s_{opt} is the size of an optimal canonical separator.

Another algorithm is then animated. It is the weighting method described by Goodrich and the author in [1]. We show how putting a weight on the faces of Q induces a weight for the cells of the arrangement $\mathcal{A}$, and how keeping only the heavy cells yields a separator which is nothing else than a $(1/2c)$-net for the planes $\mathcal{H}(Q)$ supporting faces of Q (see Fig. 6, bottom). We then explain the reweighting strategy in this context. Eventually, a separator is found that has 13 facets. In general, the weighted separator is guaranteed to have $O(s_{opt})$ facets.

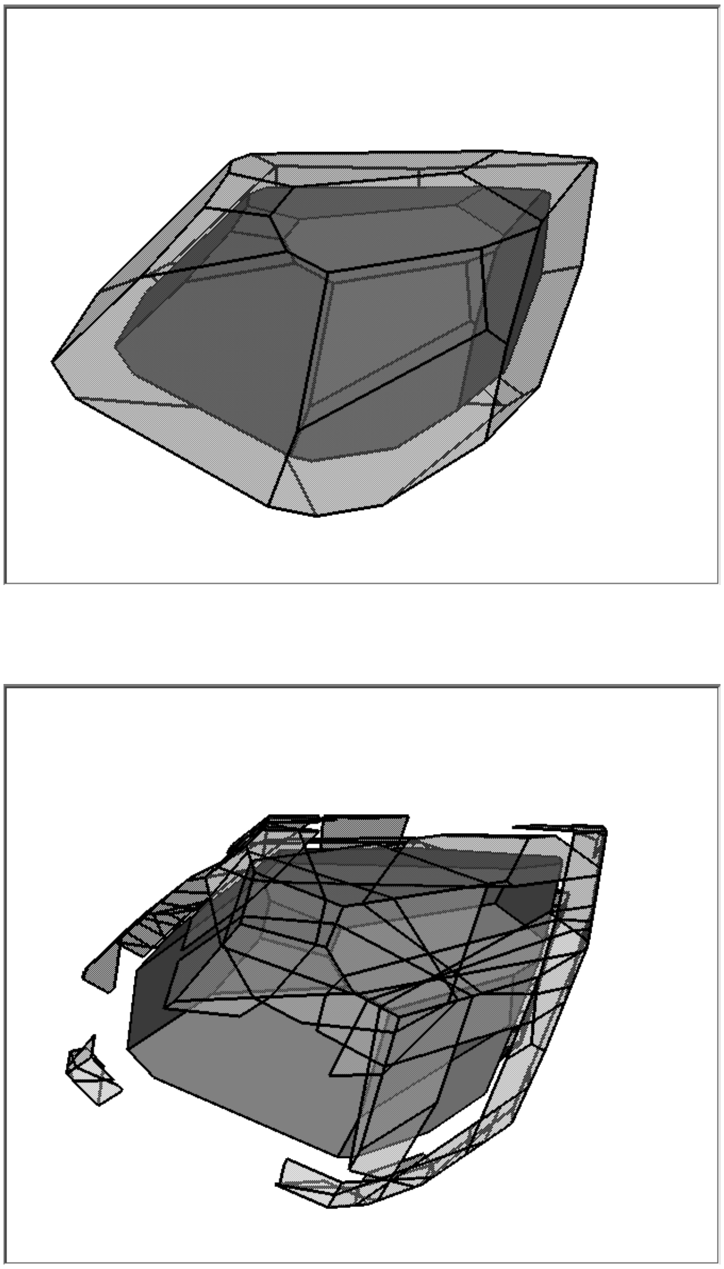


Figure 6: The top image shows two nested polyhedra. The bottom image shows the weighted algorithm in action. Specifically, the heavy cells are displayed in yellow on P, and the cover for those cells is a $(1/2c)$-net for the planes with weight, displayed in green on the boundary of Q.

3 Technical Details

Both algorithms are implemented and animated using GASP, a Geometric Animation System created at Princeton University by Ayellet Tal and David Dobkin [3]. The implementation is written in C and runs on a Silicon Graphics Iris workstation. The algorithm is fully animated, in the sense that it produces an animation for any given data P and Q. The data used for the video were created with a random generator of planes tangent to the sphere, then intersected to obtain Q, which was blown up to yield P.

Acknowledgments

We would like to thank Ayellet Tal and David Dobkin for the customized GASP environment and their invaluable help. Thanks also go to Kirk Alexander and Kevin Perry, of Interactive Computer Graphics Lab at Princeton University and Mike Mills, of the Princeton Department of Media Services.

The author was supported in part by NSF Grant CCR-93-01254, the Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology Inc., and Ecole Normale Supérieure of Paris.

References

[1] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. In *Proc. 10th ACM Sympos. Computational Geometry*, pages 293–302, 1994.

[2] G. Das. *Approximation schemes in computational geometry*. Ph.D. thesis, University of Wisconsin, 1990.

[3] D. P. Dobkin and A. Tal. GASP–a system to facilitate animating geometric algorithms (Video). These proceedings.

[4] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 296–306, 1992.

Interactive Visualization of Weighted Three-dimensional Alpha Hulls

Amitabh Varshney Frederick P. Brooks, Jr.
William V. Wright

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

8

The accompanying videotape shows an interactive visualization of weighted three-dimensional α-hulls for static and dynamic spheres. The α-hull is analytically computed and represented by a triangulated mesh. The entire surface is computed and displayed in real-time at interactive rates. The weighted three-dimensional α-hulls are equivalent to smooth molecular surfaces of biochemistry. Biochemistry applications of interactive computation and display of α-hulls or smooth molecular surfaces are outlined.

Introduction

The α-hull has been defined as a generalization of the convex hull of point-sets by Edelsbrunner, Kirkpatrick, and Seidel [3, 4]. Given a set of points P, a ball b of radius α is defined as an *empty α-ball* if $b \cap P = \phi$. For $0 \leq \alpha \leq \infty$, the *$\alpha$-hull* of P is defined as the complement of the union of all empty α-balls [3]. It has been shown that it is possible to compute the α-hull of a set of points P from the Voronoi diagram of P. For $\alpha = \infty$ the α-hull over the set of points P is the same as their convex hull. Edelsbrunner and Mücke [5] have defined the *α-shape* over P to be the polytope that approximates the α-hull over P by replacing circular arcs of the α-hull by straight edges and spherical caps by triangles. An α-shape of a set of points P is a subset of the Delaunay triangulation of P. Edelsbrunner [3] has extended the concept of α-shapes to deal with weighted points (i.e. spheres with possibly unequal radii) in three dimensions. An α-shape of a set of weighted points P_w is a subset of the regular triangulation of P_w.

The smooth molecular surface of a molecule is defined as the surface which an exterior probe-sphere touches as it is rolled over the spherical atoms of that molecule. This definition of a molecular surface was first proposed by Richards [7]. This surface is useful in studying the structure and interactions of proteins, in particular for attacking the protein-substrate docking problem. The analytic computation of the molecular surface was first done by Connolly [2].

Looking at the above definitions, one can see that the weighted α-hulls for three dimensions and the smooth molecular surface of a molecule with a probe-radius α have the same definitions. In this video we present the visualization of weighted α-hulls as used to define molecular surfaces.

Our Approach

Computation of α-hulls and α-shapes has traditionally been done by first constructing power diagrams or regular triangulations. Since these methods involve computing the entire diagram or triangulation first and then culling away the parts that are not required, their complexity is $O(n^2)$ in time, where n is the number of points. This is worst-case optimal, since an α-shape in three dimensions could have a complexity of $\Omega(n^2)$.

However, when α-hulls are used as molecular surfaces, one can do better. Let $M = \{S_1, \ldots, S_n\}$, be a set of spheres, where each sphere, S_i, is expressed as a pair (c_i, r_i), c_i being the center of the sphere and r_i being the radius of the sphere. Collections of spheres representing molecules have two interesting properties: (i) the minimum distance d_{ij} between any two centers c_i and c_j is greater than or equal to a positive constant l_{min} — the smallest bond-length in the molecule and (ii) the values of all the radii can be bounded from above and below by strictly positive values, $0 < r_{min} \leq r_i \leq r_{max}$. We take advantage of the first property to arrive at better running times for our algorithm. Stated simply, the first property says that the number of neighboring atoms within a fixed distance from any atom i, is always bounded from above by a constant k_{max} that depends on the minimum spacing between any two atoms. We refer to two atoms i and j as *neighboring* if it is possible to place a probe sphere such that it is contact with both S_i and S_j.

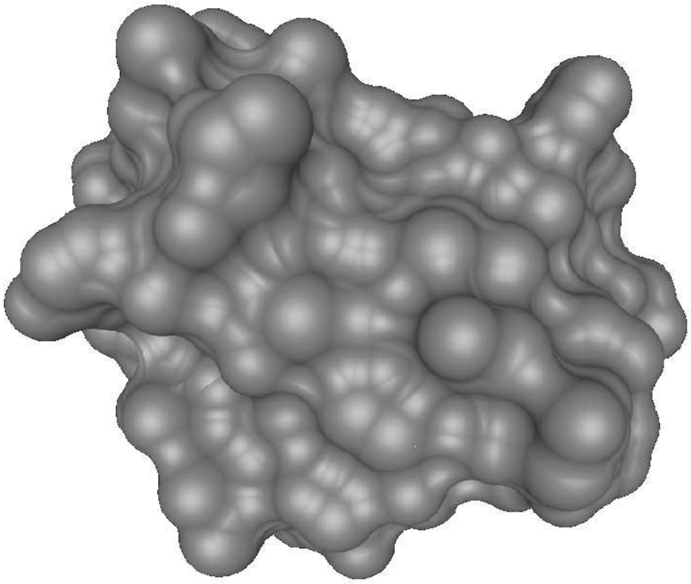


Figure 7: The molecular surface (alpha-hull) for crambin at probe-sphere radii (alpha value) of 1.4 Ångstroms.

A power cell (as described by Aurenhammer [1]) for a given sphere S_i with respect to M can be computed as the intersection of $n - 1$ halfspaces, each halfspace contributed by the pair $(S_i, S_j), 1 \le j \le n, j \neq i$. If the average number of neighbors for an atom is k, then for our purposes it is sufficient to just compute an approximation to the power cell, which we call a *feasible cell*, as an intersection of k halfspaces (one halfspace contributed by each neighbor). This can be done in deterministic time O($k \log k$). For n atoms, this task can be parallelized over n processors, each processor computing the feasible cell for one atom. To check if a feasible cell is non-null, we use a randomized linear programming algorithm that has linear expected time complexity and is quite fast in practice [8]. For details of our approach the interested reader can refer to [9].

We have used ideas from the theory of packing and covering of spheres to estimate the value for k, the average number of neighboring atoms, for an atom in protein molecules. We can prove that for proteins, $k < 140$ for a probe-sphere radius of 1.4Å– the radius of a water molecule. Details about this result can be found in [10]. In practice we have found that for protein molecules and for a probe-radius of 1.4Å, k is around 45.

In the algorithms for computing the convex hull of a set of points, it is assumed that the points are in a general position, i.e. no more than d points lie on the same $d-1$ dimensional hyperplane. In reality this assumption often fails to hold, leading to problems. For example, planar benzene rings occur often in proteins, causing six carbon and six hydrogen atoms to be all coplanar. One of the recent approaches to solving this problem has been to perturb the input point set slightly to avoid these degeneracies. The approach of Emiris and Canny [6] perturbs the j^{th} dimension of the i^{th} point as:

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon(i^j \bmod q) 1 \leq i \leq n, 1 \leq j \leq d \tag{1}$$

where ϵ is a symbolic infinitesimal and q is the smallest prime greater than n. Instead of performing exact integer arithmetic, we just perturb the centers of the spheres by the above scheme and that has worked quite well for us in practice.

With no preprocessing, we can compute the molecular surface for a 396 atom Crambin and a probe-radius of 1.4Å, using 40 Intel i860 processors in 0.2 seconds. We have implemented α-hulls on Pixel-Planes 5, though the method is general enough to be easily implemented on any other parallel architecture. Our approach is analytically exact. The only approximation is the degree of tessellation of spherical and toroidal patches.

Applications in Biochemistry

Interactive computation and display of molecular surfaces should benefit biochemists in three important ways. First, the ability to change the probe-radius interactively helps one study the surface. Second, it helps in visualizing the changing surface of a molecule as its atom positions are changed. These changes in atom positions could be due to user-defined forces as the user attempts to modify a molecular model on a computer. Third, it assists in incorporating the effects of the solvent into the overall potential energy computations during the interactive modifications of a molecule on a computer.

Scope for further work

At present we are not using any incremental temporal information in constructing these surfaces. Thus, if the atoms move slightly from their positions, the whole sur-

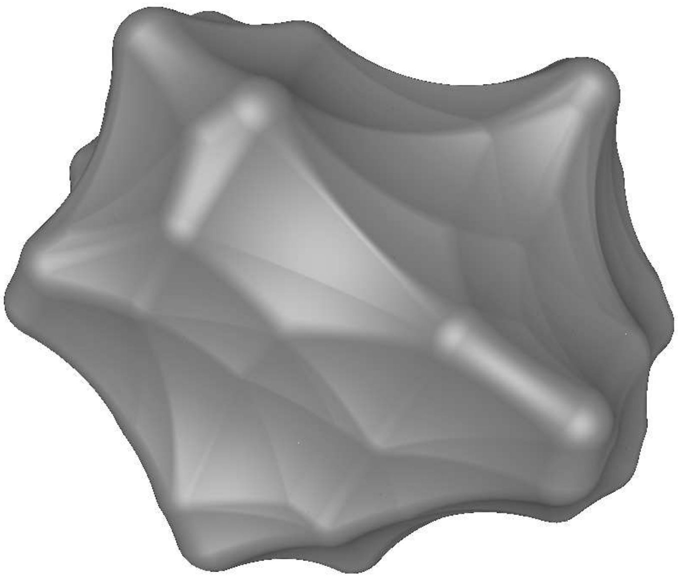


Figure 8: The molecular surface (alpha-hull) for crambin at probe-sphere radii (alpha value) of 10 Ångstroms.

face is recomputed from the beginning, although one could conceivably store the feasible cell description with each atom. Assuming the atoms of the molecule move along continuous trajectories, it should be possible to compute such surfaces (and indeed α-hulls and α-shapes) incrementally and efficiently by using the information from previous time steps.

Acknowledgments

We would like to acknowledge the valuable discussions we had with Pankaj K. Agarwal, Dinesh Manocha, Jan F. Prins, and David C. Richardson during various stages of this work. We would also like to acknowledge Michael E. Pique and Victoria Roberts at the Scripps Clinic for the molecular dynamics data. We would like to thank the anonymous referees for their suggestions which have led to improvements in the presentation of this extended abstract as well as the video. This work was supported by NIH National Center for Research Resources grant number 5-P41-RR02170.

8

References

[1] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal of Computing*, 16(1):78–96, 1987.

[2] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16:548–558, 1983.

[3] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1740, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[4] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):551–559, 1983.

[5] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1), 1994.

[6] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Eighth Annual Symposium on Computational Geometry*, pages 74–82, Berlin, Germany, June 1992. ACM Press.

[7] F. M. Richards. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioengg.*, 6:151–176, 1977.

[8] R. Seidel. Linear programming and convex hulls made easy. In *Sixth Annual ACM Symposium on Computational Geometry*, pages 211–215, Berkeley, California, June 1990. ACM Press.

[9] A. Varshney and F. P. Brooks, Jr. Fast analytical computation of Richards's smooth molecular surface. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 300–307, October 1993.

[10] A. Varshney, W. V. Wright, and F. P. Brooks, Jr. Bounding the number of unit spheres inside a larger sphere. Technical Report UNC-CS-TR-93-039, Department of Computer Science, University of North Carolina at Chapel Hill, 1993.