Computational Geometry --

Where Did it Come From,

What is it,

What is it Good For

Comp Geom--Where did it come from?

Personal recollections

Computational Geometry as a research discipline started in the early 1970's in the math programming, theory/algorithms and CAD communities.

Earliest papers,

Chand+Kapur -- convex hull by gift wrapping
motivated by math programming

Graham -- convex hull by Graham scan
motivated by statistics problem

DPD + Lipton -- multidimensional searching
motivated by NPcompleteness,
open problem in Knuth, vol 3.

Shamos -- geometric computing
roots in theory, numerical analysis, OR

Shamos + Hoey-- Voronoi diagram
solved open problem in previous paper

Forrest -- computational geometry
motivated by CAD/CAM work

Comp Geom -- How did it grow?

Separate conferences, journals start in 1985

- SCG 1985
- DCG
- CCCG
- CGTA
- IJCGA

Various workshops

- Santa Cruz
- Oberwohlfach
- Dagstuhl
- Manufacturing + CG, CG Software, ...
- now, 10 year retrospective ...

Currently, 500-600 papers per year

Experiments with the computational geometry bibliography (version of February 1, 1996).
ftp.cs.usask.ca:pub/geometry/geombib.tar.Z

(with enormous thanks!!)

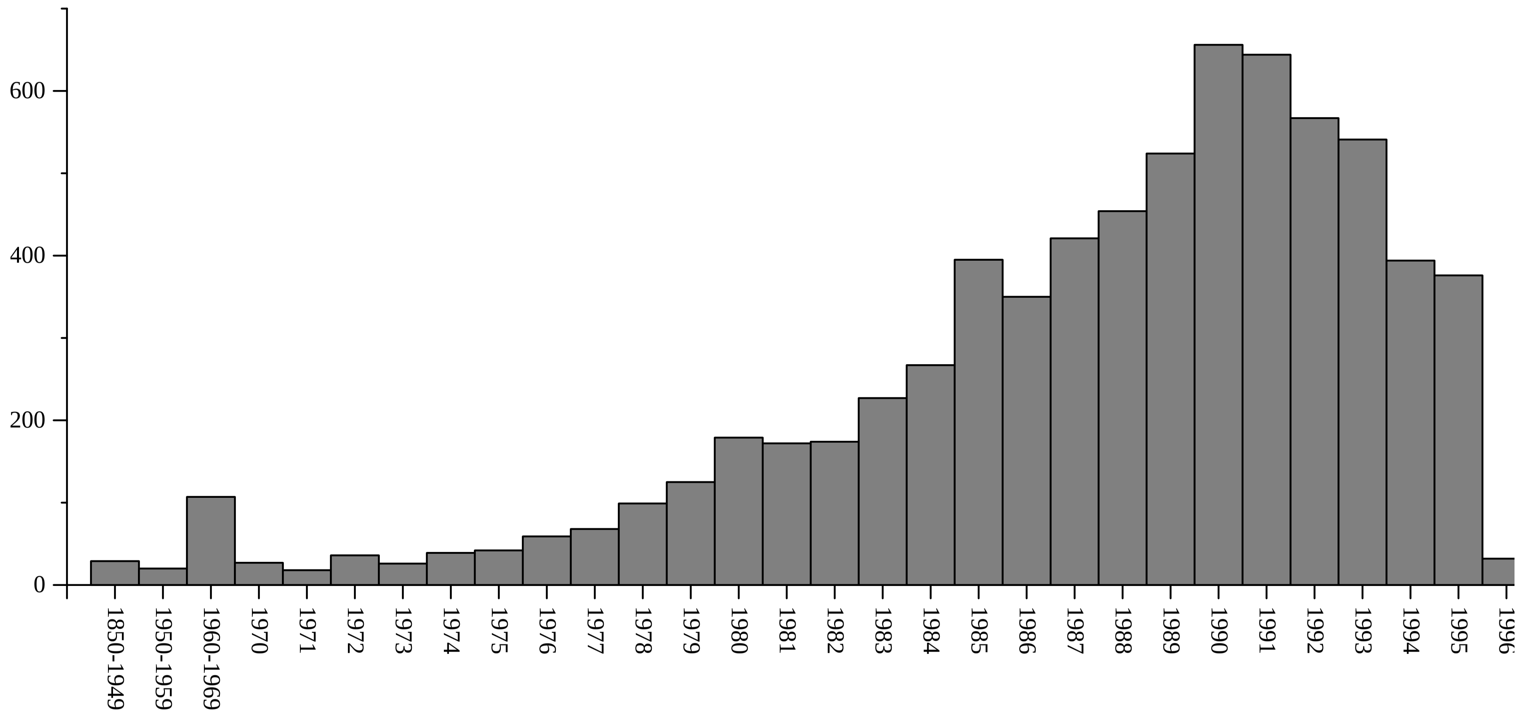
The Bibliography -- What goes in?

Papers relevant to computational geometry, which for us means the study of the computational complexity of well-defined geometric problems. Thus we are talking algorithms, data structures, analysis of time and storage, lower and upper bounds, but also geometric objects, geometric operations, and combinatorial complexity of geometric structures.

We interpret relevance in a rather broad sense, although we prefer that references from cognate areas (such as discrete geometry, solid modeling, image processing, computer graphics, and robotics) emphasize books or survey articles rather than individual papers.

The earliest papers

1850 Dirichlet (first Voronoi paper)
1857 Sylvester quadratic forms
1860 Sylvester
1885 min circle enclosing n points in plane
1888 line arrangements
1907 by Voronoi on quadratic forms
1908 original Voronoi paper
1909 by Voronoi
1911a) Thiessen **"Precipitation Averages for Large Areas"**
1911b) Lennes
Theorems on the simple finite polygon and polyhedron
1928 Ackermann function
1930 Helly
1932 van Kampen; Flores cell complexes
1934a) Steinitz and Rademacher -- **polyhedra**
1934b) Delaunay -- Delaunay triangulation
1935a) Erdos and Szekeres,
"A combinatorial problem in geometry"
1935b) Alexandroff and Hopf, Topologie
1936a) Santalo, integral geometry (2)
1936b) Wagner
(any triangulation can be transformed into any other by flips)
1948 Fary **(straight line embeddings of graphs)**
1951 Tarski, ...
1955 Hadwiger
1957 Prim, MST
1959 Dijkstra, shortest path



Publications over time

What is computational geometry?

Nature of the work

theoretical -- combinatorial insights into the
nature of geometric structures

algorithmic -- new and improved methods of
doing computations geometric

applications -- using the new techniques to
understand, then solve problems of
practice.

Sample areas

Computer Graphics
Robotics
GIS
Computer Vision
Information Retrieval
CAD/CAM
Manufacturing
Computational Biology
Medicine
Machine Learning

What is it good for?

Possible answers

1. intellectual sustenance

   these are difficult problems whose solutions are interesting

2. seed corn

   if we understand these concepts, we will have fuel with which to solve problems

3. direct solutions to problems

   there are real world problems requiring these techniques for their solution

4. provoker of problems

   by solving these problems, we will find new problems that are interesting

5. creation of tools (meta-solutions)

   the tools we develop here are the basis of many problem solutions

6. combining many of the above

How to find the answer

1. Prove hard and interesting theorems

2. Develop ideas and then look for a problem that needs these ideas.

3. Build environments in which it is easier to prove theorems.

4. Consider a problem from an application and look for methods that can be used to solve it.

5. Implement an algorithm, publicize your code and ask people how they use it.

6. Build environments in which it is easier to implement algorithms.

Case Studies

1. **qhull --** an implementation of practical importance that arose from theoretical considerations.

2. **sampling problems of computer graphics --** a problem that arose in practice, admitted a theoretical solution, led to theoretical extensions that found practical application in machine learning.

3. **polyhedral hierarchies --** finding an application to the progressive refinement of meshes many years later.

Case studies not covered --

4. Doing empirical studies of algorithms on data sets of practical import.

5. Algorithm animation and debugging tools.

6. Libraries, programming environments and a usable software base.

1. qhull

Where it came from?

Brad Barber proposal -- I want to build the
ideal programming system.

DPD -- here's an application.
robust geometric computation.

Step 1 -- problem to consider
Convex hull in arbitrary dimension.

Step 2 -- split time between
proving theorems and
doing experiments to
understand the problem

Step 3 -- write paper, publish code

Step 4 -- iterate with referees re paper,
iterate with users re code
add author Hannu Huhdanpaa

Step 5 -- measure impact

A few observations...

People don't really want the hull,
they want a structure to use.

So, develop an algorithm that generates a structure which when queried with points returns

INSIDE
OUTSIDE
CANNOT DETERMINE

Avis et al (posting to compgeom mailing list)

CH1: Find all faces of all dimensions of P.

CH2: Construct the Hasse diagram of the face lattice of P.

CH3: Find all facets of P.

CH4: Construct a triangulation of the boundary of P.

Besides roundoff error, input data may not be precise. Points are really balls.

This leads to convex hulls where facets have thickness, and where the convex hull **may not be convex.**

Modes of operation

Measuring running times

Theoretical

If the following conditions hold, the algorithms runs in time ....

Worst case vs. average case

What you can prove vs. what you observe

Practical

Benchmarking against the best known algorithms on datasets from practice.

Times observed by users after the code is released.

Modes of operation

Complexity of doing the work

Proving theorems is a difficult task; we all understand this.

Implementing code that people (who are in vastly different communities) will want to use is a more difficult task; we don't understand this process as well.

Reward system

Papers are judged to be correct and important by referees who are experts in the field. The key thing is that the proofs are correct and the results are valuable.

Programs are judged to be correct when bugs stop being reported. Their importance is measured empirically by ftp counts, breadth of application, ...

Demographics of convex hull programs

Who uses them?

[Qhull] has been used for
support structures in layered manufacturing \cite{all–dut95},
classification of molecules by their biological activity,
vibration control,
geographic information systems,
neighbors of the origin in the \R{8} lattice,
stress analysis,
stability of robot grasps \cite{bel–ras95},
spectrometry \cite{boar93},
constrained control allocation \cite{bor–dur95},
robot navigation \cite{cuc–net95},
micromagnetic modeling \cite{por–gla96},
and invariant sets of delta–sigma modulators \cite{zha–goo94}.

Demographics of convex hull programs

How many people use them? (Data as of 1/96)

ftp counts for copies of qhull

3071	qhull.tar.Z	
655	qhull.sit.hqx	(mac version)
443	qhull-2.01.tar.Z	
424	qhull-1.0.tar.Z	
296	qhull-2.1.tar.Z	
184	qhull-2.02.tar.Z	
118	qhull-2.0.tar.Z	
87	qhull-2.2.tar.Z	
19	qhull-2.1b.tar.Z	
1	qhull.tar.gz	

5298	total	

(from Steve Fortune)

I counted about 3000 distinct requests for my voronoi code over the life of netlib. For 1995 only, Ken's [Clarkson] convex hull code was requested 924 times. My guess is anybody who gets one of the programs gets them all.

Lessons learned

Convex Hull/Voronoi/Delaunay code is most desired by those in other areas.

Next highest is ~ 1000 ftp hits.

but still not as popular as other tools

geomview, mpeg_play ~ 30K ftp hits

Software that's out there will be applied, often to problems we don't know and in directions we'd have never guessed.

Writing good code is significantly more time consuming than writing a good paper.

the rewards are higher

broad application possible, reaches a much wider community.

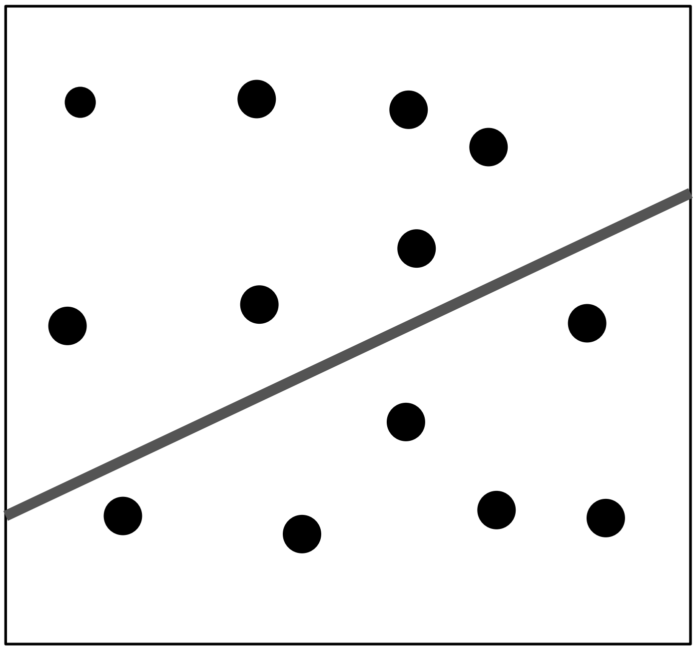
the rewards are lower

within the community we give little recognition to experimental work.

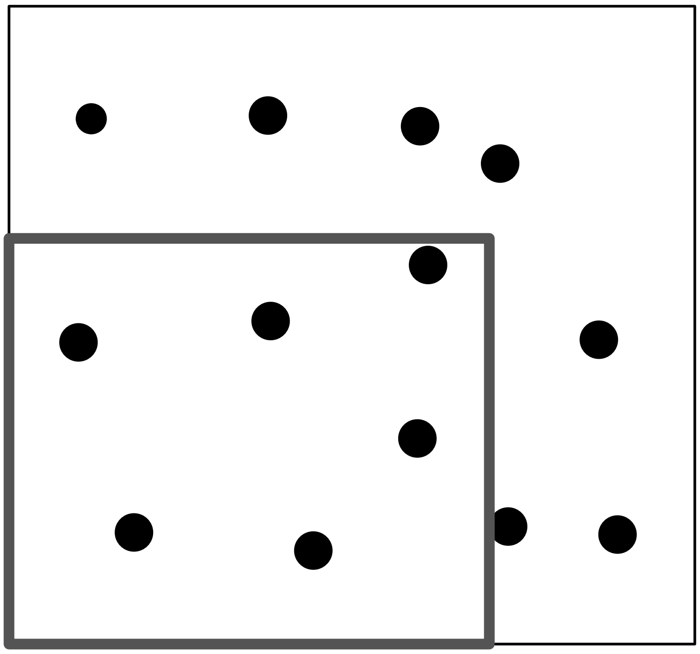
2. Sampling Problem

Initial Motivation

Mitchell -- Need for good sampling sets



line cuts



(anchored) box cuts

Good ==> the fraction of sample points in a region approximates the area of the region.

Measure of goodness -- maximum (L_∞) or weighted average (L_2).

Discrepancy is the right measure

$$D(X) = \max \{ | \mathrm{Area}(R) - |(R \cap X)|/|X| \, | \}$$

The Underlying data structures question

Our data are real numbers in [0,1].

Our operations are:

INSERT(x): Add x to the data set,
increase the number of data by 1.

After each insertion assume that the data are sorted as

$$x_1, x_2, \ldots, x_n$$

QUERY: Find the value of i such that
$x_i - i/n$ is maximized or minimized.

Difficulty:

$$x_i - i/n > x_{i+1} - (i+1)/n$$

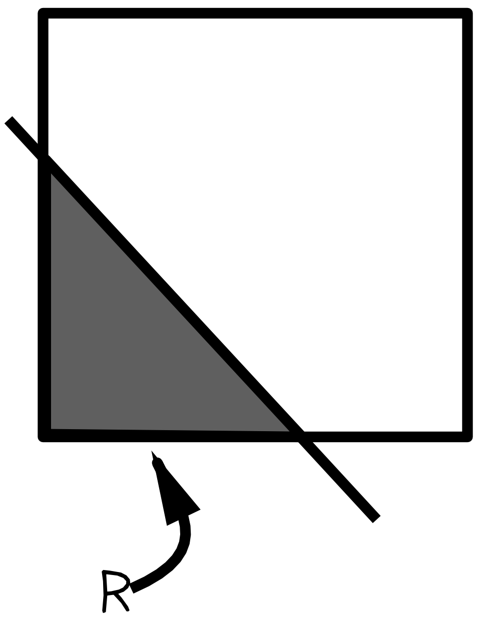
But,

$$x_i - i/(n+1) < x_{i+1} - (i+1)/(n+1)$$

Enter David Eppstein...

Discrepancy of Arbitrary Edges

The typical graphics situation has an edge clipped within a pixel.



We want to be sure to sample the correct proportion of samples in each part of the pixel.

S is a point set S
the Edge Discrepancy of S is

$$\max\left(\mathrm{Area}(R) - |R \cap S|/|S|\right)$$

the max is over all possible shaded regions R.

Computing the Edge Discrepancy

1. Observe that there are a limited # of possible ``shaded regions''

2. Notice that we can move to dual space and walk the line arrangement of the points in S.

3. Analyze this situation in primal space.

   There are n^2 edges that matter plus a few boundary effects.

4. Implementation is more easily done by an $O(n^2 \log n)$ algorithm which is faster than the naive $O(n^3)$ algorithm.

5. There are extensions to higher dimensions...

Running Times...

Points	Fast	Slow
100	1.7	3.2
200	6.8	23.9
400	28.7	178.7
800	120.2	1399.5
1600	513.2	11272.3
3200	2258.9	89351.3
6400	9554.8	738041.4

Who cares?

The discrepancy gives us a sense of how well
we are approximating an integral via
a Monte Carlo procedure.

Lower discrepancy ==> faster convergence
==> fewer samples ==> faster rendering

Process	16 points	256 points
Zaremba	0.184	0.0345
jittered	0.183	0.0296
Dart-Throwing	0.180	0.0339
Poisson	0.299	0.0791
On-Line MC	0.169	0.0281
Off-Line MC	0.106	0.0215

2-Dimensional arbitrary-edge L^∞ discrepancies.

(average over 100 trials for random ones)
(on line add a point at a time)
(off line find n points and move to improve)

N	Poisson	Jittered	Uniform	Annealed
4	0.46	0.41	0.25	0.25
9	0.38	0.25	0.17	0.14
16	0.35	0.21	0.12	0.094
25	0.15	0.12	0.10	0.069
36	0.23	0.11	0.083	0.055
49	0.19	0.09	0.071	0.045
64	0.12	0.07	0.062	0.038

Arbitrary-edge discrepancies for some patterns

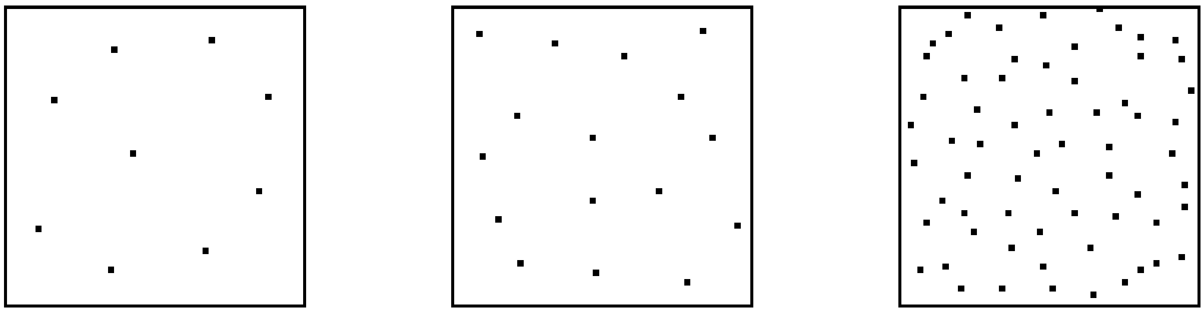
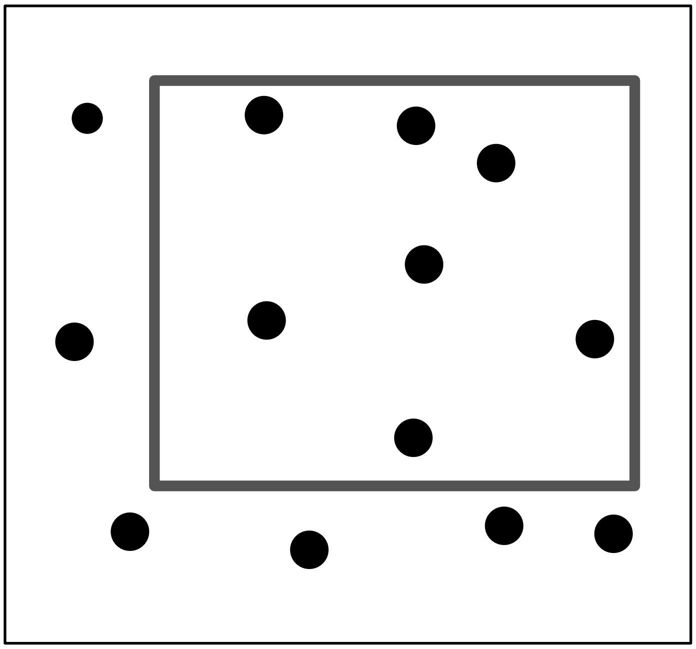


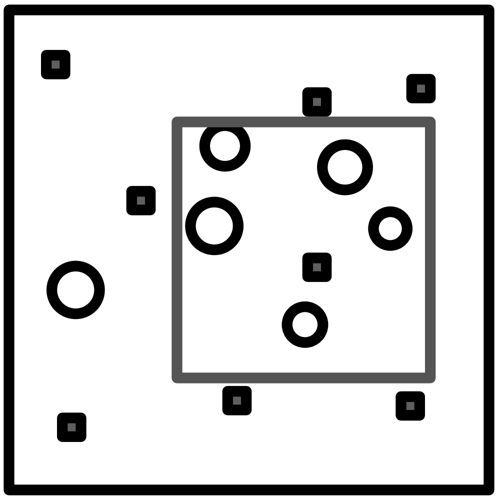
Figure 4. Low-discrepancy patterns for 9, 16, and 64 points.

One last discrepancy model

Generalize the rooted rectangle to arbitrary axis-oriented rectangles.



Other extension, consider bichromatic points



Discrepancy = 5-1 = 4

Applicable to problems of concept learning.

e.g. Appendicitis data, other data sets in the machine learning community

Lessons learned

Practice and theory can cross-fertilize.

Doing ``theoretical generalizations'' may lead to ``practical applications'' in some cases.

An asymptotically faster algorithm may run faster in practice

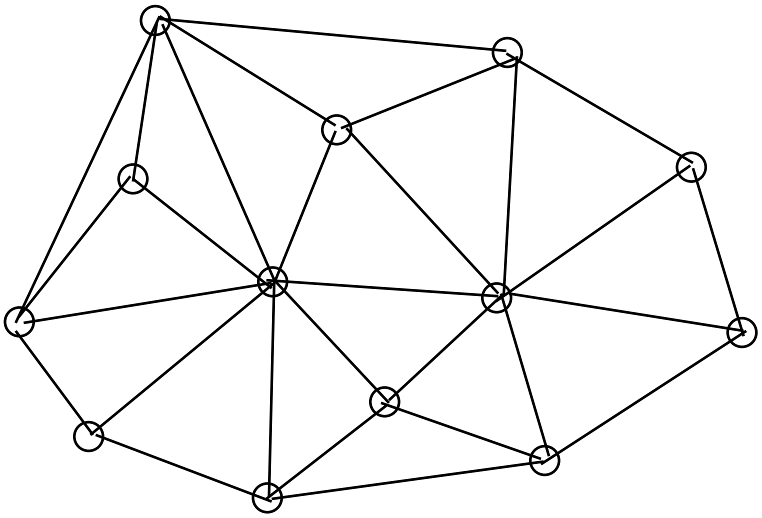
Sample data sets are useful in experimental work

This work may lead to publication in unexpected places

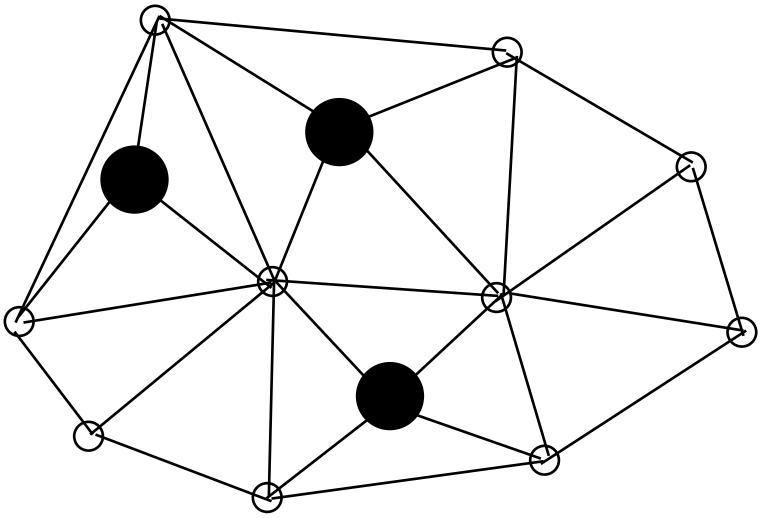
DPD + Eppstein --> SCG
DPD + Mitchell --> GI
DPD + Eppstein + Mitchell --> TOGS

DPD + Gunopolous --> COLT
... + Maass --> JCSS

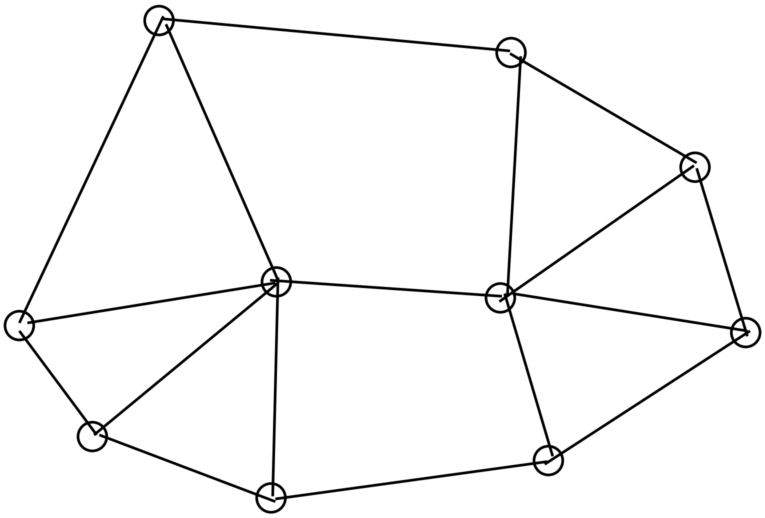
3. Hierarchical searching



Kirkpatrick 1979

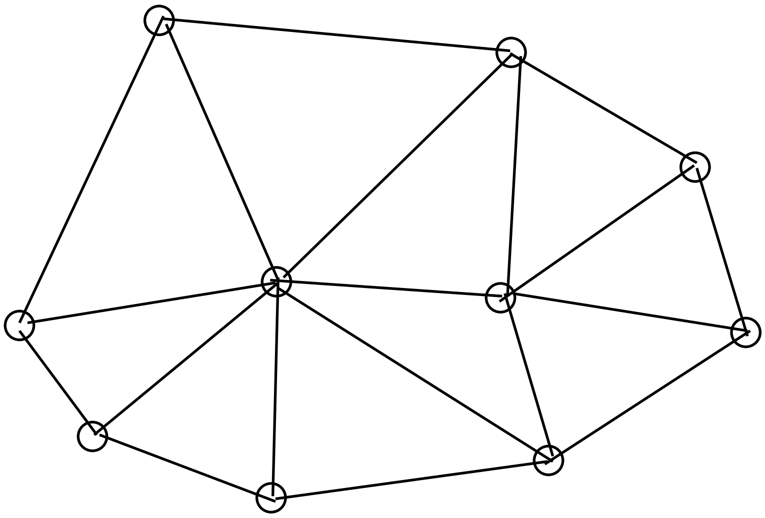


Pick an independent set of vertices.



Remove them

Retriangulate



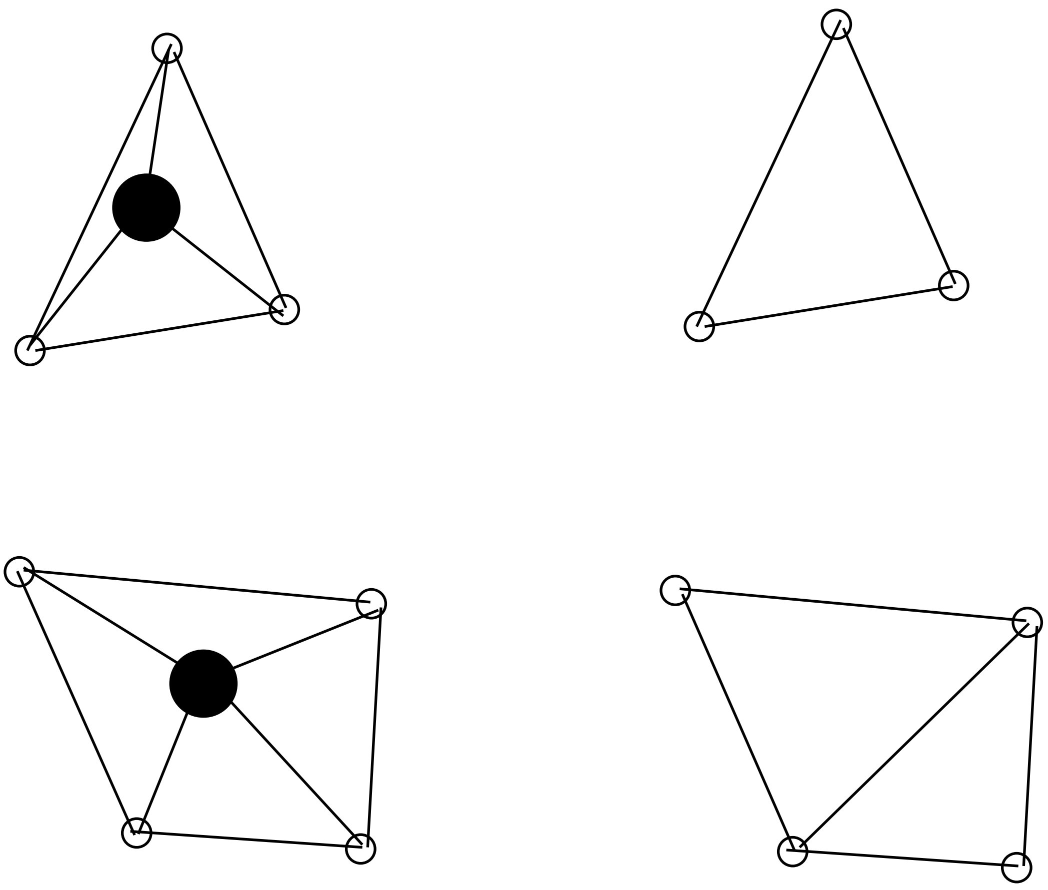
Key observations

1. You always make progress
   i.e. remove a constant fraction of vertices

2. Coarse --> fine follows from fine --> coarse

3. Planar subdivision searching becomes easy

4. Extends to polyhedral intersection problems
   can detect intersection faster than
   you can read the input (DPD+Kirkpatrick)



Questions

1. Is it practical?
   Asymptotic provable constants are large,
   In practice, it works fair

2. Can it be implemented?
   600 lines of *theoretician's* C
   of which 150 are *debugging*
   Builds on pre-existing solid modeling tools

3. What can we learn from an implementation?
   real behavior
   experiments
      e.g. does the shape stay symmetric
         can we bias removal
         what are average shapes
         how robust is it

4. Does it have further applications?
   decimation
   sub-sampling (e.g. for wavelets)

Application

Large meshes need to be presented at various levels of simplification.

This simplification is used for progressive rendering of a shape. (simple example GIF-89)

This simplification is also used for solving PDE's (simple example multigrid techniques)

Modeling the simplification can be done by wavelet techniques. Even points remain, odd points are removed. Reconstruction is done by adding back in odd points. Empty regions are filled from information stored with the even points and then updated by the odd points.

Implementation done
Application in process
``pretty pictures beginning to result''
DPD + Schroeder + Sweldens

Health Report

Computational Geometry

Conferences and Journals continue to thrive
Field has leveled off but appears stable

Applied Computational Geometry

There's a lot out there waiting to blossom
Conference program is very good
rejected good rather than accepting bad

Experimental Computational Geometry

Field needs some encouragement
Reward system needs to be changed
Tools for software sharing must be improved
Sample data sets are needed, ...

Conclusion

Computational Geometry as we know won't be around in 10 years (and shouldn't be).

A prescription for the future

- Keep the good theory happening
- Encourage applications
- Reward experimental work
- Make experimental work easier to do
- Keep the discussion going
  - task force report,
  - workshops at different directions,
  - compgeom mailing list,
  - dialogues with other fields