Some of the Theory behind LEDA

Kurt Mehlhorn
MPI Informatik, Saarbrücken

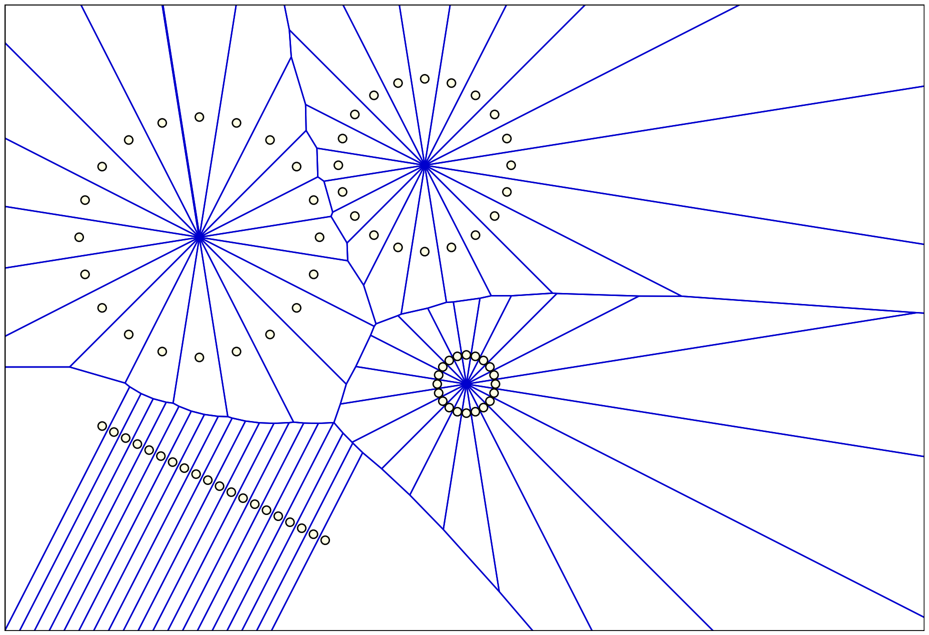
Stefan Näher
Universität Trier

Christian Uhrig
Algorithmic Solutions GmbH, Saarbrücken

Christoph Burnikel, Michael Seel, Stefan Schirra, Oliver Zlotowski, Mathias Bäsken, Joachim Ziegler, Guido Schäfer, Sven Thiel, Ernst Althaus, David Alberts, Ulrike Bartuschka, Ulrich Finkler, Stefan Funke, Evelyn Haak, Andreas Luleich, Jochen Könemann, Mathias Metzler, Michael Müller, Michael Muth, Markus Neukirch, Markus Paul, Christian Schwarz, Michael Wenzel, Thomas Ziegler, and many others

I ran a number of LEDA demos during my talk. You will get more out of the slides, if you run the demos on the side. I will insert slides with screen dumps at the appropriate places. As a screen saver I showed the demo

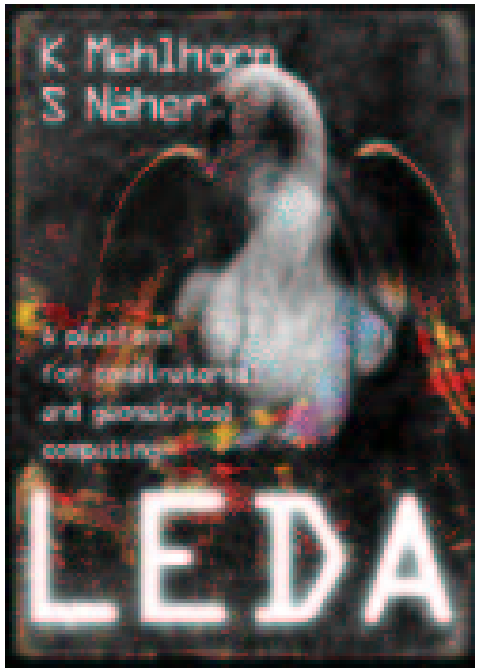
`demo/geowin/geowin_voro`



I pointed out later in the talk that the picture is so beautiful because the input is highly degenerate.

Outline of Talk

- **Introduction**
  - what is LEDA?
  - who uses LEDA and what for?
- **The Theoretical Basis of LEDA**
  - Algorithms
  - Correctness
  - Efficiency
  - System Architecture
- **Summary**



What is LEDA?

- LEDA = **L**ibrary of **E**fficient **D**ata Types and **A**lgorithms
  - covers a large part of combinatorial and geometric computing (AHU, CLR, Mehlh, PS, BKOS)
  - easy to use
  - extendible
  - correct
  - efficient
- provides algorithmic intelligence for applications in GIS, VLSI-design, scheduling, traffic planning, graphics, facility planning, computational biology, . . .
- a platform on which to build applications
- a tool for teaching algorithms and algorithm engineering
- extended by AGD, LEDA-SM, CGAL

Modules in LEDA

- Basic Data Types: random source, stack, queue, map, list, set, dictionary, priority queue, . . .
- Advanced Data Types: partition, sorted sequence, pq-trees, dynamic trees, range trees, interval trees, segment trees, . . .
- Numbers: integer, rational, bigfloat, real, linear algebra
- Graphs and Graph Algorithms: graphs, node and edge arrays, iterators, shortest paths, maximum flow, min cost flow, matching (weighted, unweighted, bipartite, general), assignment, components and connectivity, planarity, layout, . . .
- Geometry: points, lines, circles, inexact and exact geometry kernels, convex hulls, Delaunay diagrams, Voronoi diagrams, sweep-line method, polygons and boolean operations, . . .
- Visualization and I/O: graphic windows, graph editor, persistence

LEDA Users

- academic users
  - 1500 installations in more than 50 countries
  - $\leq$ 50% of our users are in CS
  - $\leq$ 20% of our users are in algorithms
- commercial users

  MCI, Siemens AG, Ford, Lufthansa Systems, E-Plus Mobilfunk, Silicon Graphics, Sony Corporation, Deutsche System Technik, France Telecom, Digital Equipment, Chevron Petroleum, Sun Microsystems, Commerz Financial Products, Daimler Benz, IBM, Mentor Graphics, Canon Research Center, General Motors, CSIRO, Leica, Hewlett Packard, VTT Information, InterHDL, Bosch Telecom, Deutsche Telekom, Minolta, NEC, ESA, Isys Software, Daimler Benz Aerospace, Electro Optical Systems, TerraGlyph, Bioreason, Mitsubishi, ST Microelectronics, Eurodecision, Real-Time Innovations, Source One Network, Viewscape3D, Lion Bioscience AG, Bioinformatics, Celera, HIS, SmithKline Beecham, Biomax, Shared Earth, General Motors, MIP, Credit Suisse, Infineon, Motorola Russia, Rational, . . .

  and about 300 others

Sample Applications

Application	Algorithmic Intelligence
traffic planning (Daimler-Chrysler)	graph algorithms, max flow, shortest path
geographic information systems (MUS)	intersection of polygons, point location, overlay of planar maps
data mining (Silicon Graphics)	data structures, graph algorithms
VLSI-design (Ford)	graph algorithms, Steiner trees, scan-line algorithms
Human Genom Sequencing (Celera Genomics)	graph algorithms, data structures, visualization

The Theoretical Basis of LEDA

1. **Algorithms**
2. **System Architecture**
3. **Correctness of Geometrical and Network Algorithms**
4. **Correctness of Implementations**
5. **Efficiency**

I am not claiming that we were the first in any of the topics above,
I do claim, however, that we made significant progress in items 2, 3, 4, and 5.

The Theoretical Basis of LEDA

1. **Algorithms**
   - when we started, we believed, that algorithm theory would suffice as a basis and
   - that it would require sweat, but no theoretical insights, to build the system.
   - we quickly learned differently
   - we had to learn about specification methods, programming paradigms, programming patterns, system architecture, numerical analysis, algebra, . . . , and
   - we had to develop new algorithms to get a correct and efficient system
2. **System Architecture**
3. **Correctness of Geometrical and Network Algorithms**
4. **Correctness of Implementations**
5. **Efficiency**

The Theoretical Basis of LEDA

1. **Algorithms**
2. **System Architecture**
   - a loose collection of programs does not make an easy-to-use, coherent, and extendible system
   - coherent design with a small number of basic concepts
   - abstract data types including abstract treatment of pointers
   - I will not go into this aspect of the work $\Longrightarrow$ try the system out
3. **Correctness of Geometrical and Network Algorithms**
4. **Correctness of Implementations**
5. **Efficiency**

The Theoretical Basis of LEDA

1. **Algorithms**
2. **System Architecture**
3. **Correctness of Geometrical and Network Algorithms**
    - algs are designed for a Real RAM, but machines provide *int*s and *double*s.
    - geometric algs are designed for non-degenerate inputs
    - design and implementation of efficient exact number types and geometry kernels (illusion of a Real RAM)
    - reformulation of geometric algs for general inputs
    - analysis of arithmetic demand
4. **Correctness of Implementations**
5. **Efficiency**

The Theoretical Basis of LEDA

1. **Algorithms**
2. **System Architecture**
3. **Correctness of Geometrical and Network Algorithms**
4. **Correctness of Implementations**
   - implementors make mistakes and we are no exception to this rule
   - program checking
5. **Efficiency**

The Theoretical Basis of LEDA

1. **Algorithms**
2. **System Architecture**
3. **Correctness of Geometrical and Network Algorithms**
4. **Correctness of Implementations**
5. **Efficiency**
   - library use incurs significant overhead
   - algs are designed to prove big-Oh statements, but for programs constant factors matter
   - low (near-zero) overhead library design
     - careful design of basic data structures
     - programming techniques that ease the task of optimizing compilers
   - libraries open opportunities: complex algs
   - design choices, heuristics, and analysis

Floating Point Arithmetic Destroys Geometry

- given two lines $\ell_1 : a_1 x + b_1 y = c_1$ and $\ell_2 : a_2 x + b_2 y = c_2$
- compute their intersection point $p = (x_p, y_p)$.

$$x_p = \frac{c_1 b_2 + c_2 b_1}{a_1 b_2 - a_2 b_1} \qquad y_p = \frac{a_1 c_2 + a_2 c_1}{a_1 b_2 - a_2 b_1}$$

- check whether p lies on ℓ_1, i.e. check whether

$$a_1 x_p + b_1 y_p == c_1$$

- with floating point arithmetic, $a_i, b_i, c_i \in [0\,..\,10000]$ uniformly at random, the check is going to fail in about one-third of the cases.

Floating Point Arithmetic Destroys Geometry

- given two lines $\ell_1 : a_1x + b_1y = c_1$ and $\ell_2 : a_2x + b_2y = c_2$
- compute their intersection point $p = (x_p, y_p)$.

$$x_p = \frac{c_1b_2 + c_2b_1}{a_1b_2 - a_2b_1} \qquad y_p = \frac{a_1c_2 + a_2c_1}{a_1b_2 - a_2b_1}$$

- check whether p lies on ℓ_1, i.e. check whether

$$a_1x_p + b_1y_p == c_1$$

- with floating point arithmetic, $a_i, b_i, c_i \in [0..10000]$ uniformly at random, the check is going to fail in about one-third of the cases.

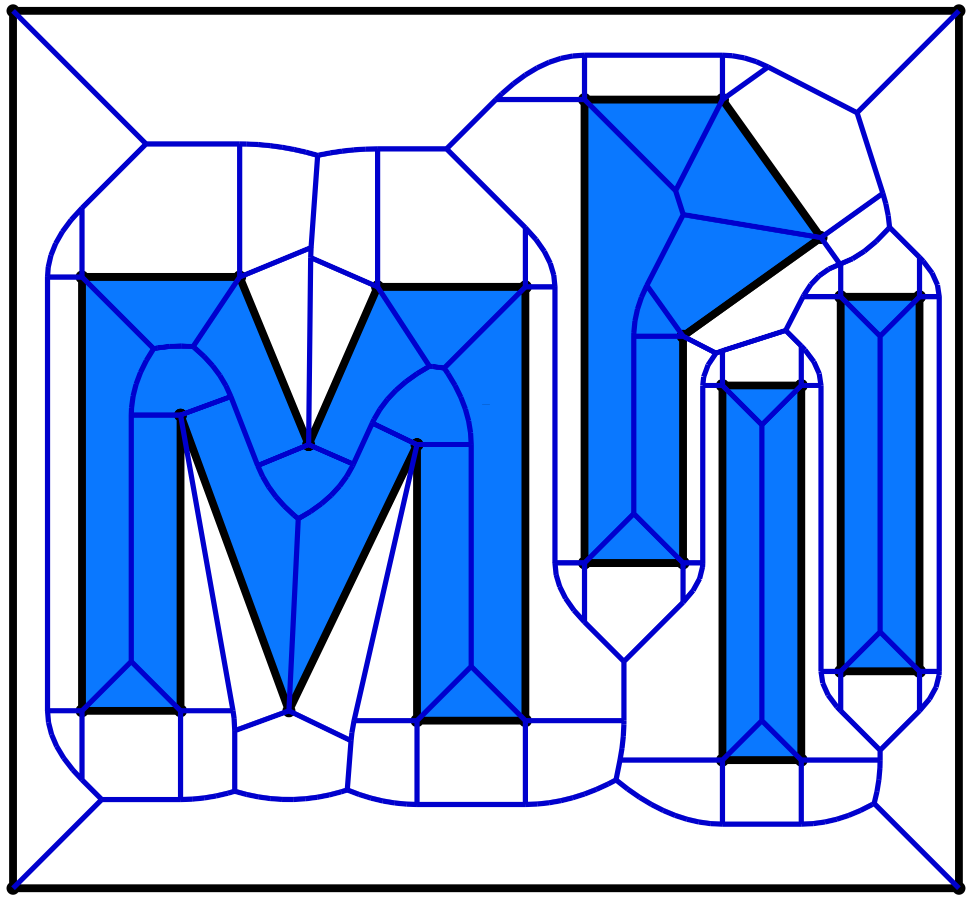
- in the first release of LEDA none of the geometric algorithms worked as claimed.
- **today, all of them do due to exact number types and geometry kernels.**

Academic Problem??? Boolean Operations on Polygons

- construct a regular n-gon P, (or cylinder)
- obtain Q from P by a rotation by α degrees about its center,
- compute the union of P and Q (= a $4n$ gon).

System	n	α	time	output
ACIS	1000	1.0e-4	5 min	correct
ACIS	1000	1.0e-5	4.5 min	correct
ACIS	1000	1.0e-6	30 sec	problem too difficult
Microstation95	100	1.0e-2	2 sec	correct
Microstation95	100	0.5e-2	3 sec	incorrect answer
Rhino3D	200	1.0e-2	15sec	correct
Rhino3D	400	1.0e-2	–	CRASH
CGAL/LEDA	5000	6.175e-06	30 sec	correct
CGAL/LEDA	5000	1.581e-09	34 sec	correct
CGAL/LEDA	20000	9.88e-07	141 sec	correct

The Voronoi Diagram of Line Segments



Voronoi diagram = points with at least two nearest neighbors

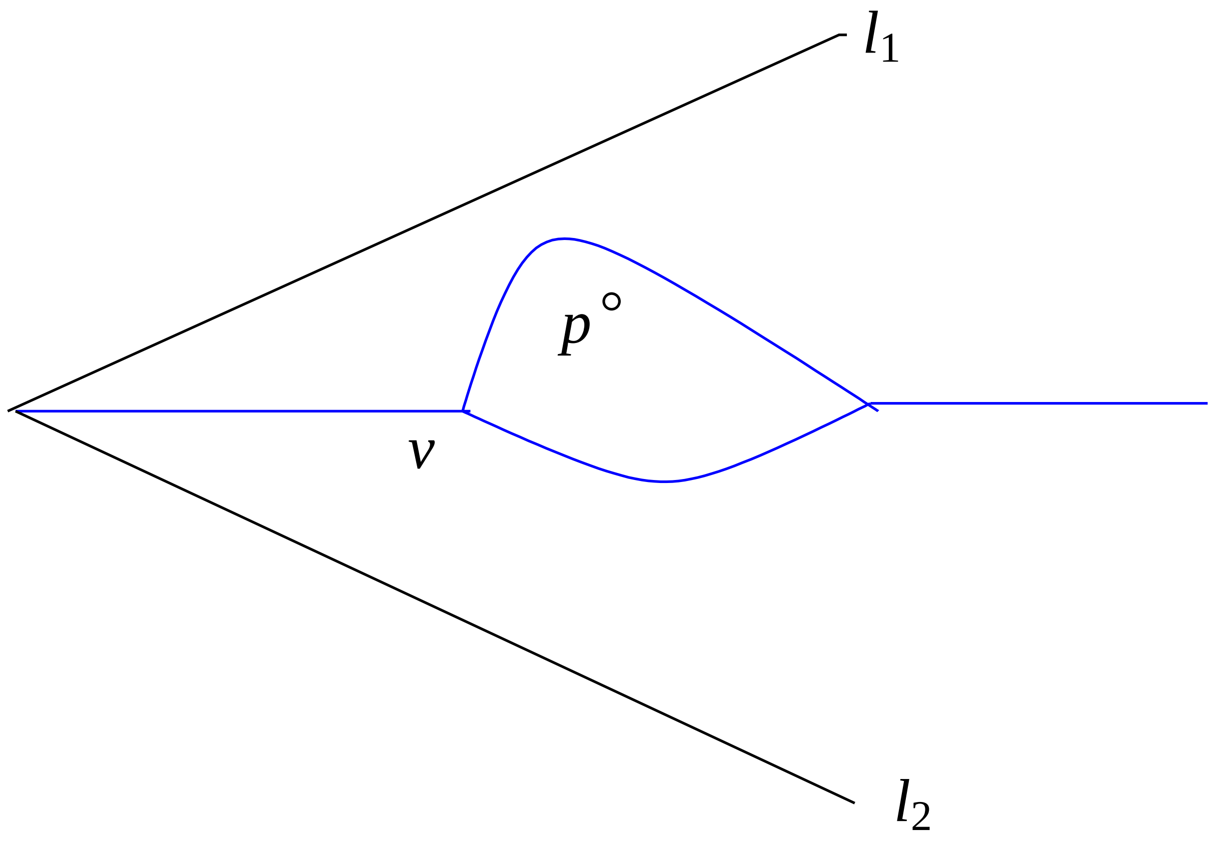
VD consists of parts of

- perpendicular bisectors of points, and
- angular bisectors of lines, and
- parabolas

Some Remarks

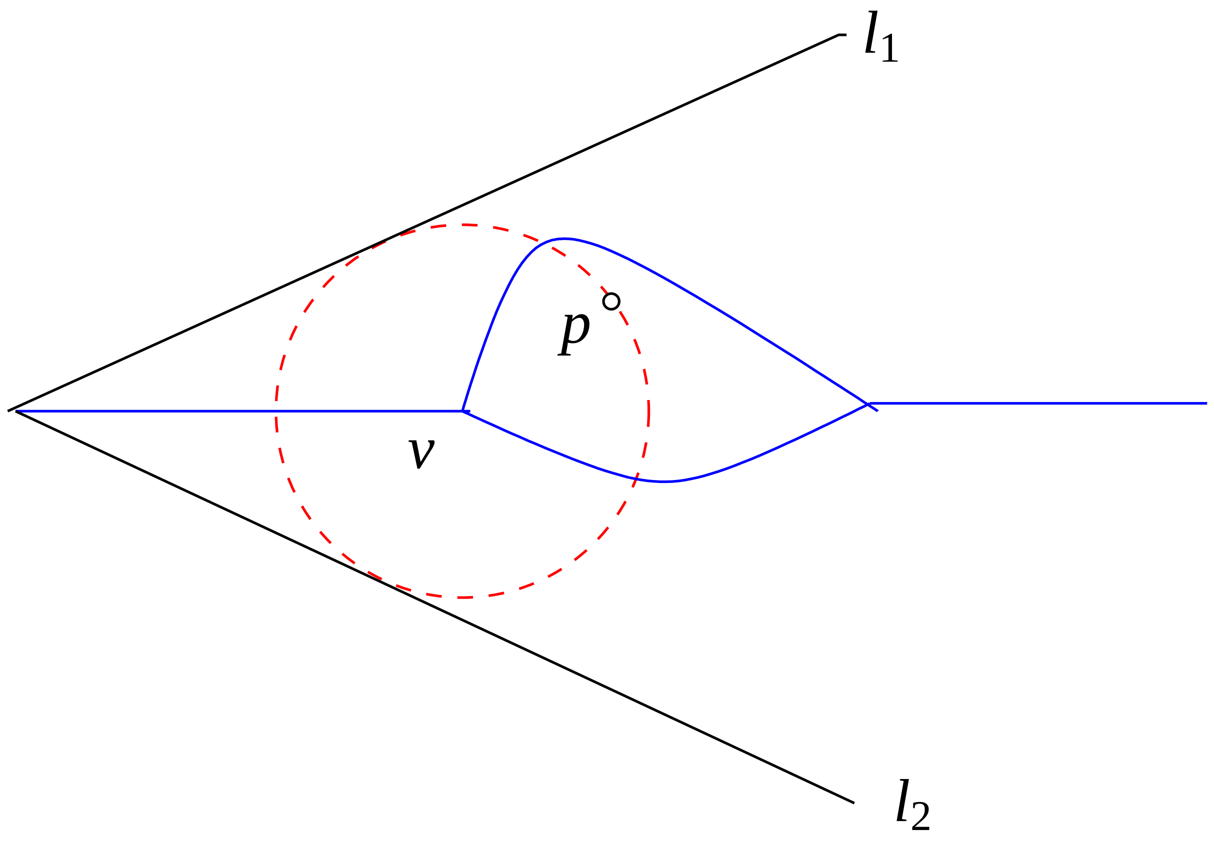
- about 10 years ago, I asked a student to implement an algorithm for Voronoi diagrams of line segments
  - he was a good student, has a PhD by now
- we found several algs in the literature
  - divide and conquer
  - sweep
  - randomized incremental
- all algs use a certain geometric primitive: the *incircle test*

A Typical Test: The Incircle Test



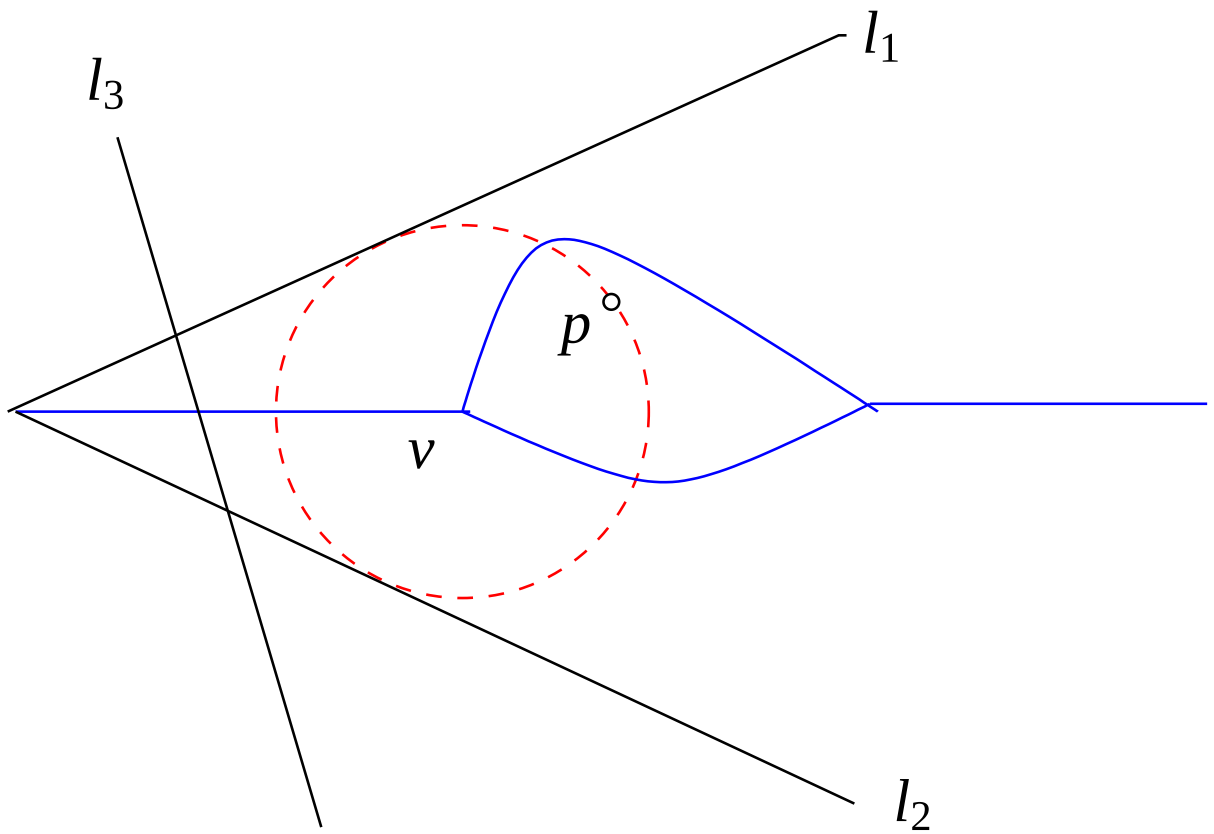
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$

A Typical Test: The Incircle Test



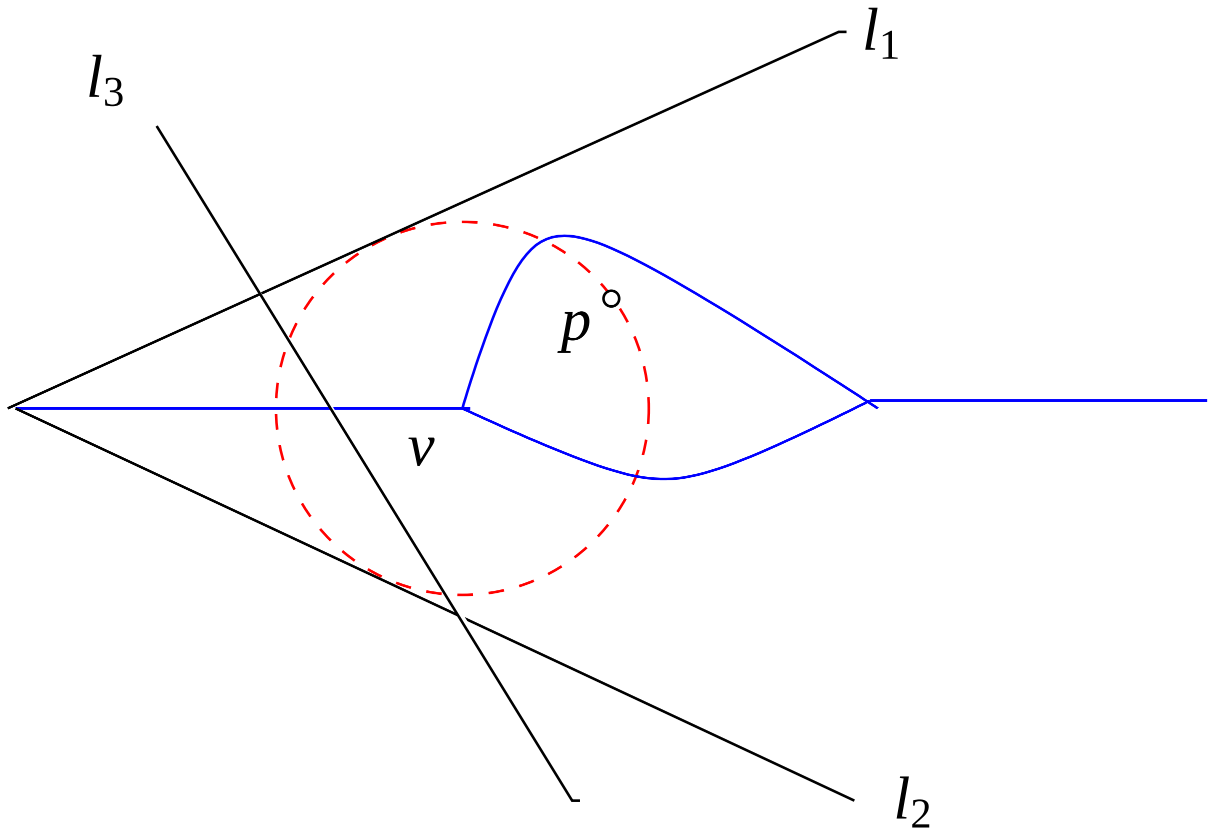
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$

A Typical Test: The Incircle Test



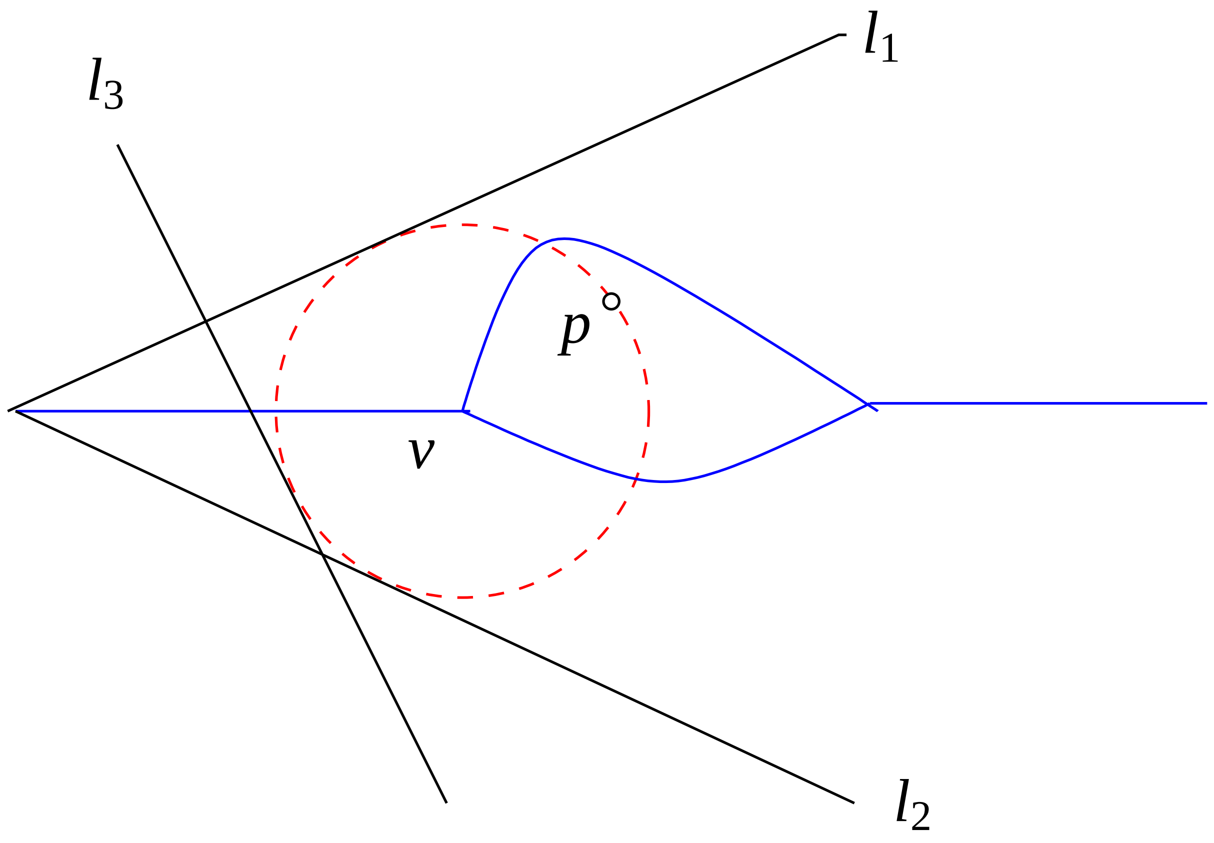
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$
- Add l_3. Is v still a Voronoi vertex?
- If $dist(v,p) < dist(v,l_3)$, **YES**

A Typical Test: The Incircle Test



- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$
- Add l_3. Is v still a Voronoi vertex?
- If $dist(v,p) > dist(v,l_3)$, **NO**

A Typical Test: The Incircle Test



- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$
- Add l_3. Is v still a Voronoi vertex?
- If $dist(v,p) = dist(v,l_3)$, **YES, BUT the diagram changes in the vicinity of v.**

Two Difficulties

- none of the papers discussed the case **YES, BUT...**
  - they all started with: *We assume our input to be in general position.*
  - we reformulated our geometric algs to make them work for all inputs.

Two Difficulties

- none of the papers discussed the case **YES, BUT...**
  - they all started with: *We assume our input to be in general position.*
  - we reformulated our geometric algs to make them work for all inputs.

- none of the papers mentioned that it might be difficult to make the case distinction, i.e., to

$$\text{compare} \quad dist(v,p) \quad \text{and} \quad dist(v,l_3)\ .$$

- floating point arithmetic (even arbitrary precision) does not suffice
- we designed efficient and exact number types for algebraic numbers.
- the exact geometry kernels of LEDA and CGAL encapsulate them and make them easily accessible.

Algebraic Formulation

- l_i: $a_i \cdot x + b_i \cdot y + c_i = 0$, $1 \le i \le 3$
- $p = (0,0)$

$$x_v = \frac{int + \sqrt{2c_1c_2(\sqrt{N}+C)}}{\sqrt{N} - (a_1a_2 + b_1b_2)} \qquad y_v = \frac{int + \sqrt{2c_1c_2(\sqrt{N}-C)}}{\sqrt{N} - (a_1a_2 + b_1b_2)}$$

where

$$N = N_1 \cdot N_2 \qquad N_i = a_i^2 + b_i^2 \qquad C = a_1a_2 - b_1b_2$$

$$x_v^2 + y_v^2 \ ? \ \frac{(a_3 \cdot x_v + b_3 \cdot y_v + c_3)^2}{a_3^2 + b_3^2}$$

A Separation Bound for Division-Free Expressions

Let E be an expression with integer operands and operators $+$, $-$, $*$ and $\sqrt{\ }$. Define

- $u(E) =$ value of E after replacing $-$ by $+$.
- $k(E) =$ number of distinct square roots in E.

Then (BFMS, BFMSS)

$$E = 0 \quad \text{or} \quad |E| \geq \frac{1}{u(E)^{2^{k(E)}-1}}$$

Theorem allows us to determine signs of algebraic expressions by numerical computation with precision $(2^{k(E)} - 1)\log u(E)$.

related work: Mignotte, Canny, Dube/Yap, Li/Yap, Scheinermann

extensions: division, higher-order roots, roots of univariate polynomials

Discussion I

How small can $A - B\sqrt{C}$ be, if non-zero? $\quad A, B, C \in \mathbb{N}$.

$$|A - B\sqrt{C}| = \left| \frac{(A - B\sqrt{C})(A + B\sqrt{C})}{A + B\sqrt{C}} \right| = \frac{|A^2 - B^2 C|}{|A + B\sqrt{C}|} \geq \frac{1}{|A + B\sqrt{C}|} \geq \frac{1}{|A| + |B|\sqrt{C}}$$

This is a special case of the theorem

- $u(E) = |A| + |B|\sqrt{C}$
- $k = 1$

Discussion II

- Consider $E = (\sqrt{x+1} + \sqrt{x}) \cdot (\sqrt{x+1} - \sqrt{x}) - 1$ where x is an arbitrary integer.
- Observe $E = 0$.
- $u(E) \approx 4x + 1 \approx 4x$ and $k(E) = 2$.
- Thus

$$E = 0 \quad \text{or} \quad |E| \geq \frac{1}{u(E)^{2^{k(E)}-1}} \approx \frac{1}{(4x)^3}$$

- It suffices to evaluate E with precision $3\log(4x) = 3\log x + 6$.

Numerical Sign Computation

```
$sep(E) \leftarrow u(E)^{1-2^{k(E)}}$;                                  // bound from previous slide
$k \leftarrow 1$;
while (true)
{ compute an approximation $\widetilde{E}$ with $|E - \widetilde{E}| < 2^{-k}$;
  if ( $|\widetilde{E}| \geq 2^{-k}$ )          return $sign(\widetilde{E})$;
  if ( $2^{-k} < sep(E)/2$ )  return “sign is zero”;     // since $|E| \leq 2^{-k} + 2^{-k} < sep(E)$
  $k \leftarrow 2 \cdot k$;                                                   // double precision
}
```

- $\widetilde{E}$ is computed by numerical methods
- worst case complexity is determined by separation bound:
  maximal precision required is logarithm of separation bound
- easy cases are decided quickly (a **big** plus of the separation bound approach)
- strategy above is basis for sign test in LEDA *reals*.

The LEDA Number Type *REAL*

The theorem is packaged in the LEDA data type *real*. It provides exact arithmetic for arithmetic expressions involving roots.

```
real x = ...  some integer ...;
real sx = sqrt(x);
real sxp = sqrt(x+1);
real A = (sxp + sx) * (sxp - sx);  // = 1
real B = A - 1;                    // = 0
cout << A.sign();                  // 1
cout << B.sign();                  // 0
```

If x has 100 binary places this takes less than .1 seconds. Run demo.

Reals are used in many geometric programs, e.g., Voronoi diagrams of line segments, boolean operations on curved polygons, arrangements of ellipsoids, …

Demo: Real Numbers

please start `demo_cout/Numbers/real_demo1`

```
mehlhorn@mpino1119:/usr/local/LEDA-4.3.1/demo_cout/Numbers real_demo1

We demonstrate the LEDA number type real which gives you exact computation
with expressions involving roots.

Consider expressions A = (sqrt(x+5)+sqrt(x))*(sqrt(x+5)-sqrt(x))
and                  B = A - 5

The value of A is 5 and the value of B is 0.

The demo asks you for an integer L, chooses a random
integer with L decimal digits and computes the signs
of A and B.

L = 1000

The sign of A is  1. This took  0.01  seconds.

The sign of B is  0. This took  0.04  seconds.
```

Are Our Programs Correct?

- we start from correct algorithms
- we have a sound basis: exact geometry kernels and exact number types
- we document and test, and our large user community tests
- we use program checking (Blum, Kannan, Wassermann, Rubinfeld)
- **many LEDA programs provide proof that their output is correct**

  and

  checkers check proof.

- **Example:** is $A \cdot x = b$ solvable?

Output:	yes	no
Witness:	x_0 such that $A \cdot x_0 = b$	c such that $c^T \cdot A = 0$
		$c^T \cdot b \neq 0$

Standard Planarity Test

Input: a graph G

Output:

yes	if G is planar
no	if G is non-planar

Story

Standard Planarity Test

Input:	a graph G	
Output:	yes	if G is planar
	no	if G is non-planar

Story

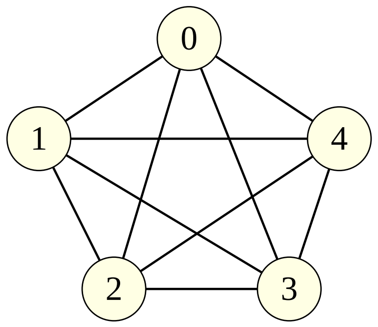
Yes, the implementor made a mistake, but the specification is to blame.

A Convincing Planarity Test

Input: a graph G

Output:

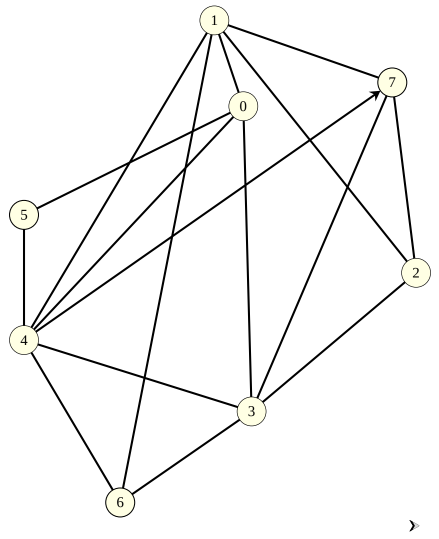
- a planar embedding, if G is planar
- a Kuratowski subgraph K_5 or $K_{3,3}$, if G is non-planar.

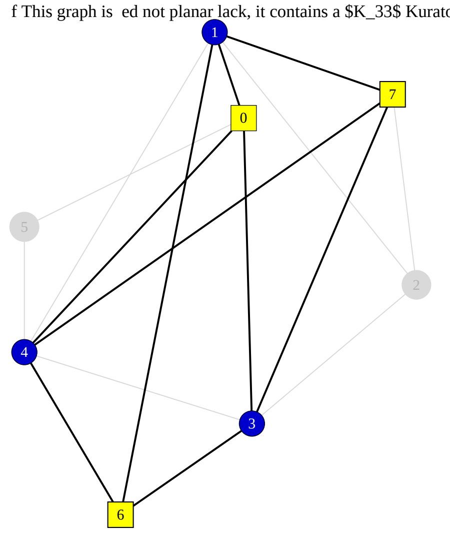


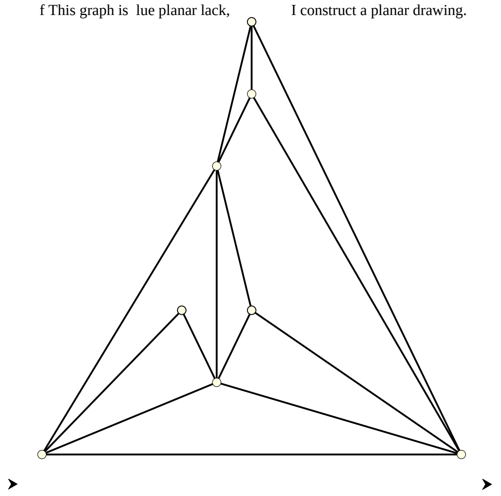
- it is easy to check the witness
  - checking the embedding: check Euler's relation: $n - m + f = 2$.
  - checking Kuratowski subgraph: check disjointness of paths
- finding witnesses is non-trivial. We use Lempel/Even/Cederbaum + Booth/Luecker for the test, Chiba/Nishizeki/Saito for the embedding and M/Näher/Hundack for the Kuratowski subgraphs; all linear time
- run planarity demo, planarity time, and, if time permits, matching demo.

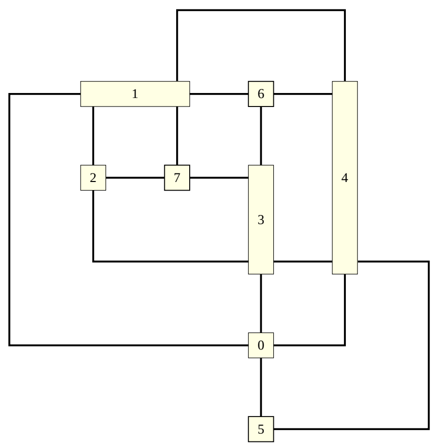
Demo: Planarity I

please start `demo/graphwin/graphwin` and construct the leftmost graph shown below (use the help button if you do not know how to construct a graph). Open the menu `Graph-Test` and press `planar`. The message “graph is not planar” should appear. Press the proof button and you should get the second picture from the left. It shows you a K_{33} subgraph. Press the done-button and you should be back to the picture on the left. Delete the only edge of G that is shown as a directed arc. Press the done-button once more and you should be back to the first graph. Go to menu `Layout-PlanarLayouts` and press Orthogonal.




f This graph is ed not planar lack, it contains a K_33 Kuratowski subdivison.


f This graph is lue planar lack, I construct a planar drawing.



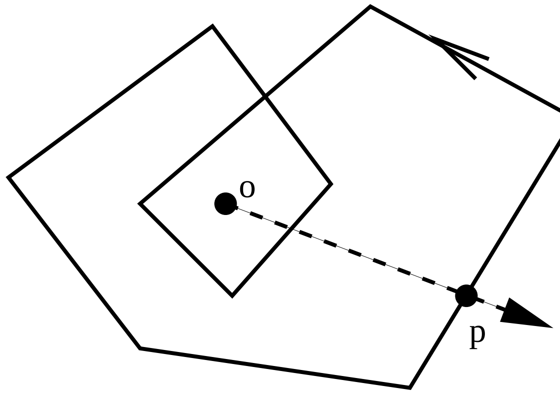
Demo: Planarity II

start `demo_cout/Graph/planarity_time`.

```
We illustrate the speed of algorithms related to planarity. There are two
planarity tests in LEDA, one based on Lempel, Even, Cederbaum and one
based on Hopcroft and Tarjan. The former is the fa-
ster and uses less space.

please type 1 if you also want to see the Hopcroft and Tarjan algorithm.1
Please start with n and m around ten-thousand.
n = 10000
m = 30000

Planar Graph
time for generation of graph =  0.22
time for planarity test, BL_PLANAR(G):  0.27
time for planarity test + embedding or Kuratowski, BL_PLANAR(G,K,true):  0.42
time for check, Genus(G) == 0:  0.05
time for planarity test, HT_PLANAR(G):  0.76
time for planarity test + embedding, HT_PLANAR(G,K,true):  1.25
Press any key to proceed.
```

Checking Convex Hulls (MNSSSS)

Given a simplicial, piecewise linear closed hyper-surface F in d-space decide whether F is the surface of a convex polytope.



F is convex iff it passes the following three tests

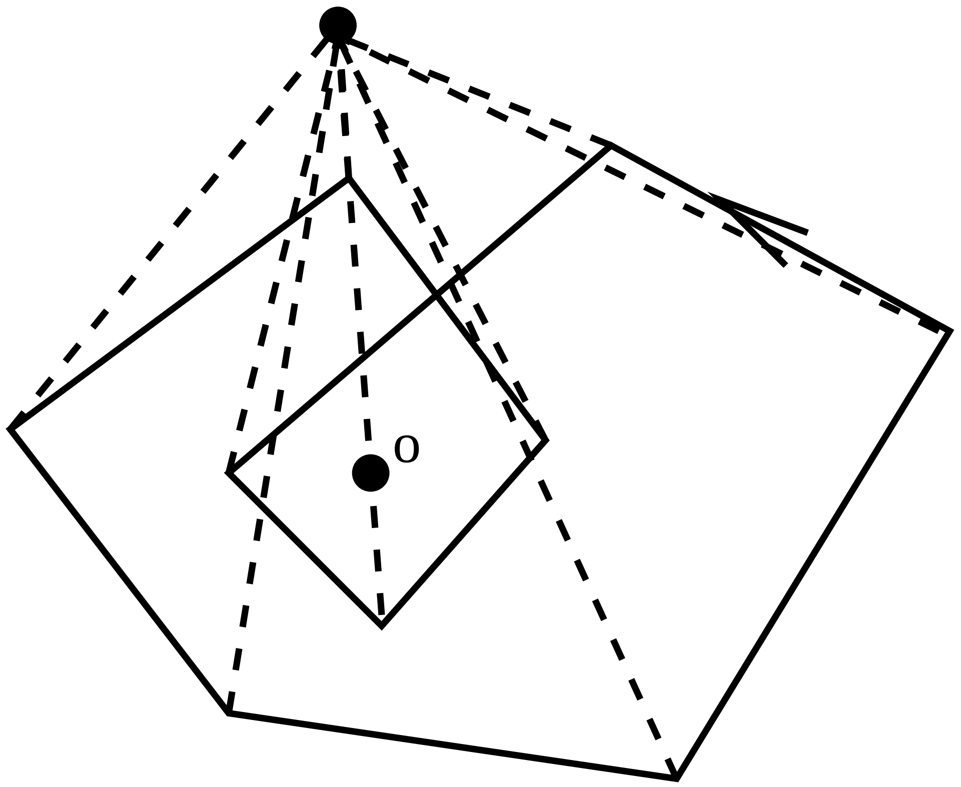
1. check local convexity at every ridge
2. $0 =$ center of gravity of all vertices

   check whether 0 is on the negative side of all facets
3. $p =$ center of gravity of vertices of some facet f

   check whether ray $\vec{0p}$ intersects closure of facet different from f

Sufficiency of Test is Non-Trivial Claim

- ray for third test cannot be chosen arbitrarily, since in R^d, $d \geq 3$, ray may “escape” through lower-dimensional feature.



Efficiency

- library use incurs significant overhead
- algs are designed to prove big-Oh statements, but for programs constant factors matter
- low (near-zero) overhead library design
  - careful design of basic data structures
  - programming techniques that ease the task of optimizing compilers
- libraries open opportunities: complex algs
- design choices, heuristics, and analysis

Disclaimer: We do not have the best implementation for all problems.

Carefully Designed Basic Data Structures I

STL versus LEDA

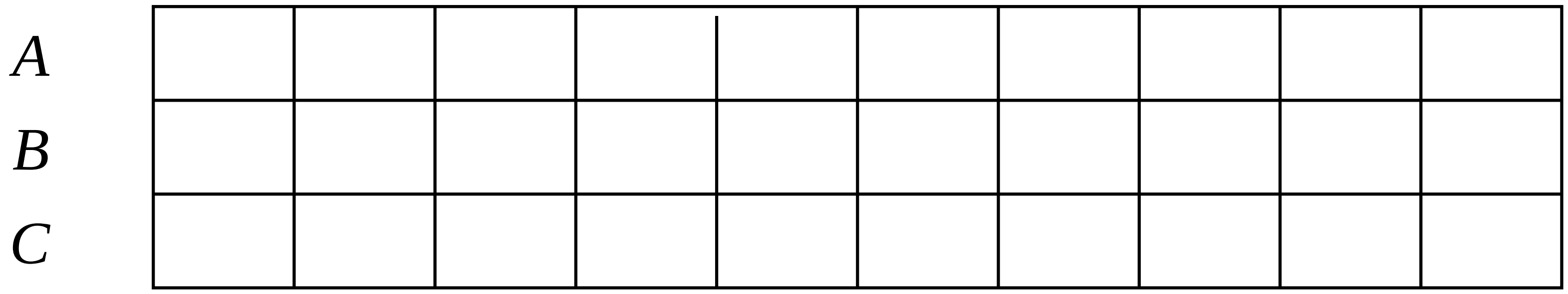
```
Number of entries: 1000000

leda_map : stl_hash_map    LEDA       STL
-----------------------------------------------
ordered insertions         0.300 sec  0.520 sec
ordered lookups            0.050 sec  0.220 sec
random  insertions         0.100 sec  0.220 sec
random  lookups            0.070 sec  0.220 sec
-----------------------------------------------
total                      0.520 sec  1.180 sec
```

- improved method for recognizing empty table positions

in `demo_cout/stl` you can find more comparisons.

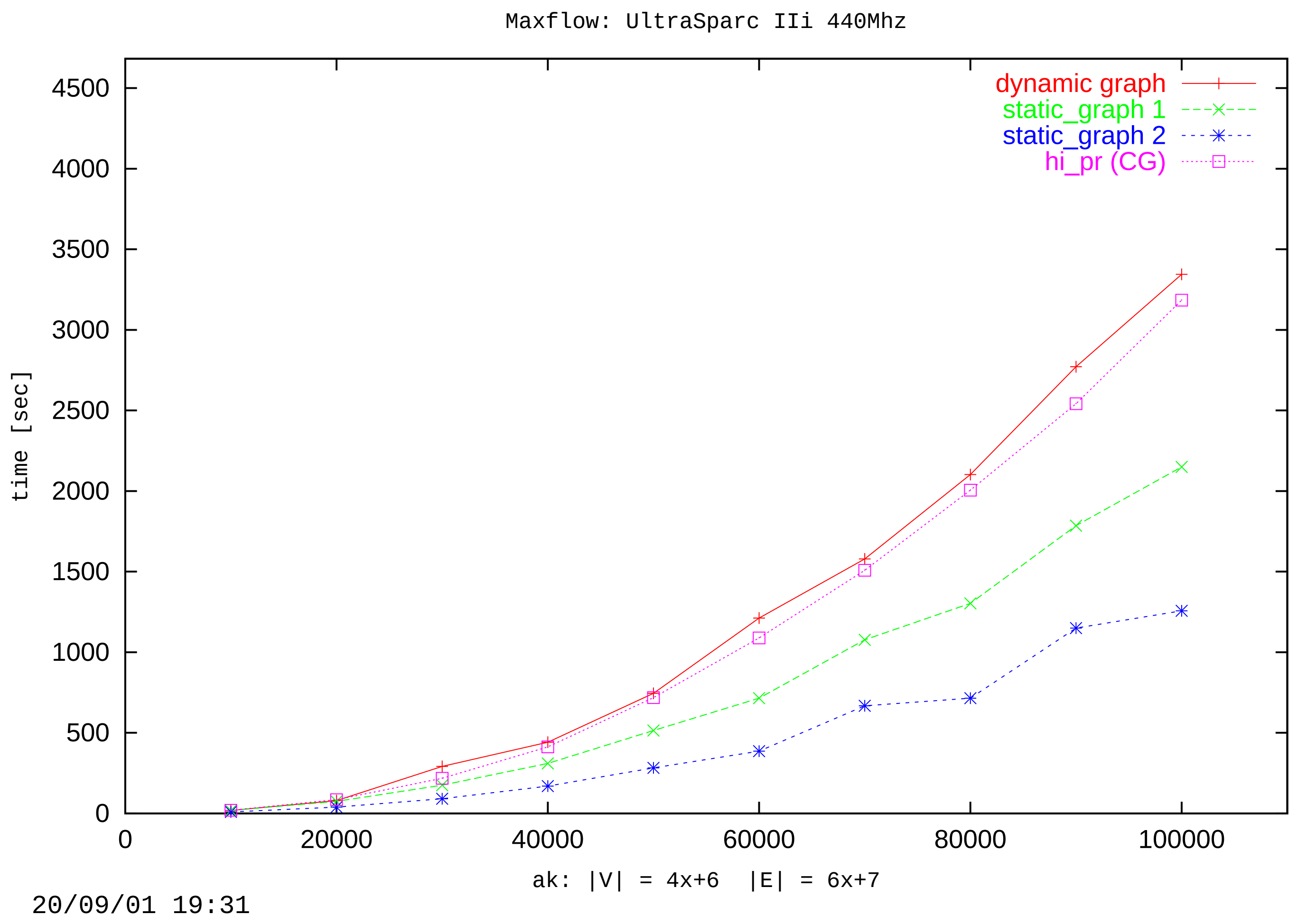
Near-Zero Overhead Library Design: Graphs (Bäsken, Näher, Zlotowski)

- LEDA offers three graph types: dynamic, static, static2 (not yet released)
- new graph data type: static2
  - compact representation of edges $v \oplus w$ instead of (v, w)
  - vertical and horizontal allocation of arrays



  - programming techniques that ease the task of optimizing compilers
  - interface slightly less convenient than for dynamic graphs
  - maybe the best graph data structure around
- all graph algs using LEDA will profit from redesign

Near-Zero Overhead Library Design II



- dynamic graph, static1, static2 use *same* LEDA-program, but different graph implementations
- CG is Cherkassky/Goldberg

Libraries Open Up Opportunities: Complex Algorithms

- weighed matching in graphs: Edmonds' algorithm

Instance	Blossom IV	LEDA
Delaunay, $n = 40,000$	32	24
Random, $n = 40,000$, $m = 6n$	205	29
Difficult Instance, $n = 151,780$, $m = 881,317$	200,019	2,799

- Blossom IV (Applegate/Cook and Cook/Rohe)
  - $O(n^3)$ version of Edmonds' algorithm due to Lawler and Gabow +
  - many clever heuristics
- LEDA (M/Schäfer)
  - $O(nm\log n)$ version due to Ghalil, Micali, and Gabow +
  - complex data structures (e.g., concatenable queues) +
  - heuristics (e.g., to construct initial matching)
- we did not have to start from scratch

Summary

- Over the past 10 years we built

  LEDA: Library of Efficient Data Types and Algorithms

  CGAL: Computational Geometry Algorithms Library

- Main Difficulties

  Correctness

  – Algs are designed for *Real RAMs*, but real machines have *ints* and *doubles*.
  $\Longrightarrow$ *exact arithmetic*

  – Programmers make mistakes $\Longrightarrow$ *program checking*

  Efficiency

  – Library Overhead $\Longrightarrow$ *careful design*

  – Algs are designed to prove a big O statement $\Longrightarrow$ *design choices*

  – Complex algs require infrastructure $\Longrightarrow$ *libraries*

  System Architecture

  – ease of use, flexibility, extendibility $\Longrightarrow$ *try them out*