Implementing Geometric Algorithms
LEDA and CGAL

Susan Hert Kurt Mehlhorn

Max-Planck-Institute for Computer Science

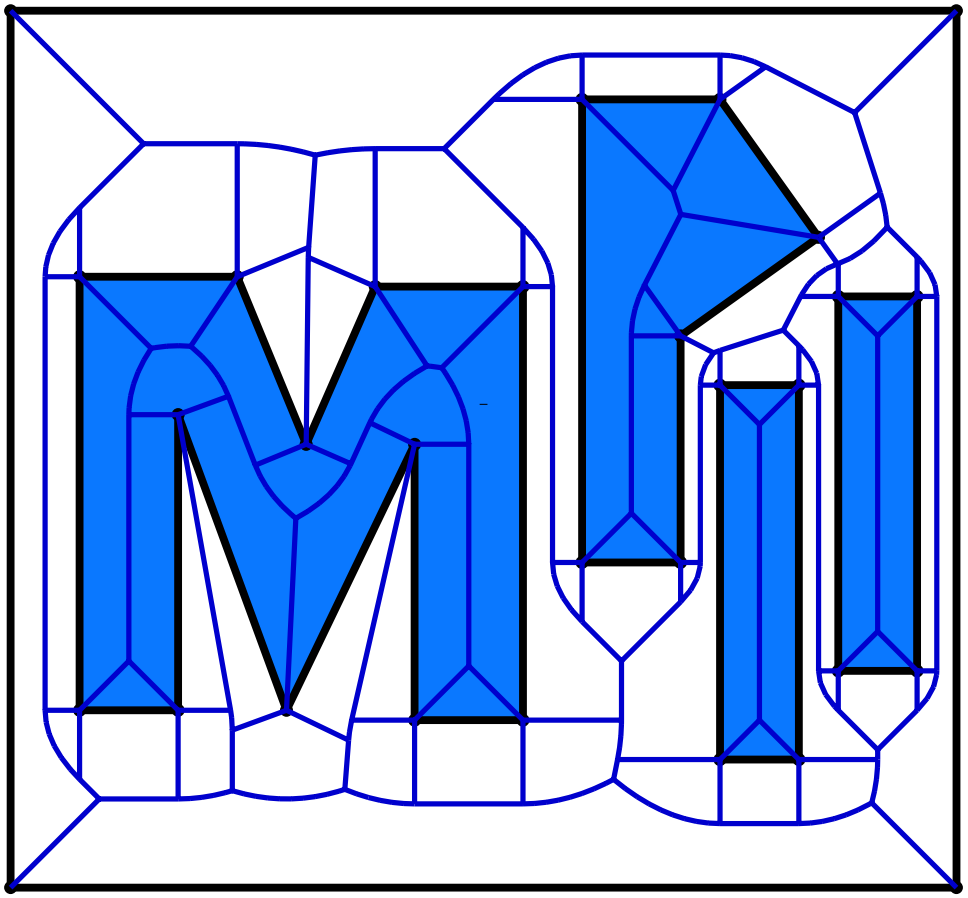
Saarbrücken

Germany

Overview

- Introduction (20 min)
- Systems I: LEDA and CGAL (20 min)
- Demos (30 min)
- Arithmetic and Exact Kernels (60 min)
- Degeneracy and Algorithmic Issues (75 min)
- Simple Algorithms (randomized algs) (45 min)
- Result Checking (30 min)
- Systems II and Summary

The Voronoi Diagram of Line Segments



Voronoi diagram = points with at least two nearest neighbors

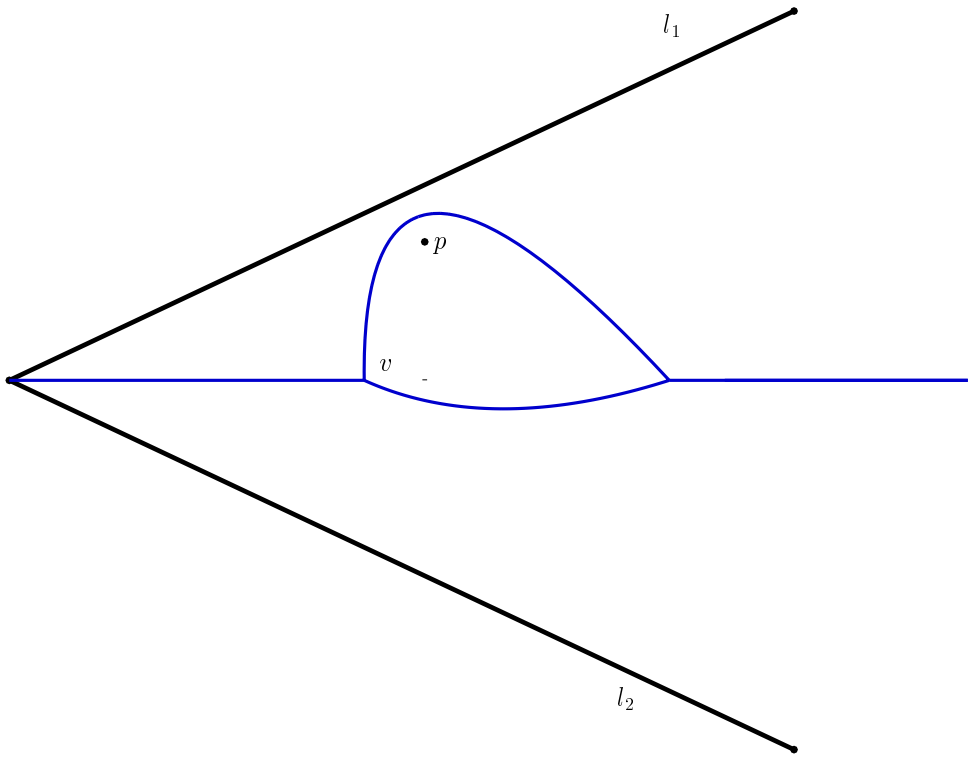
VD consists of parts of

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

Some Remarks

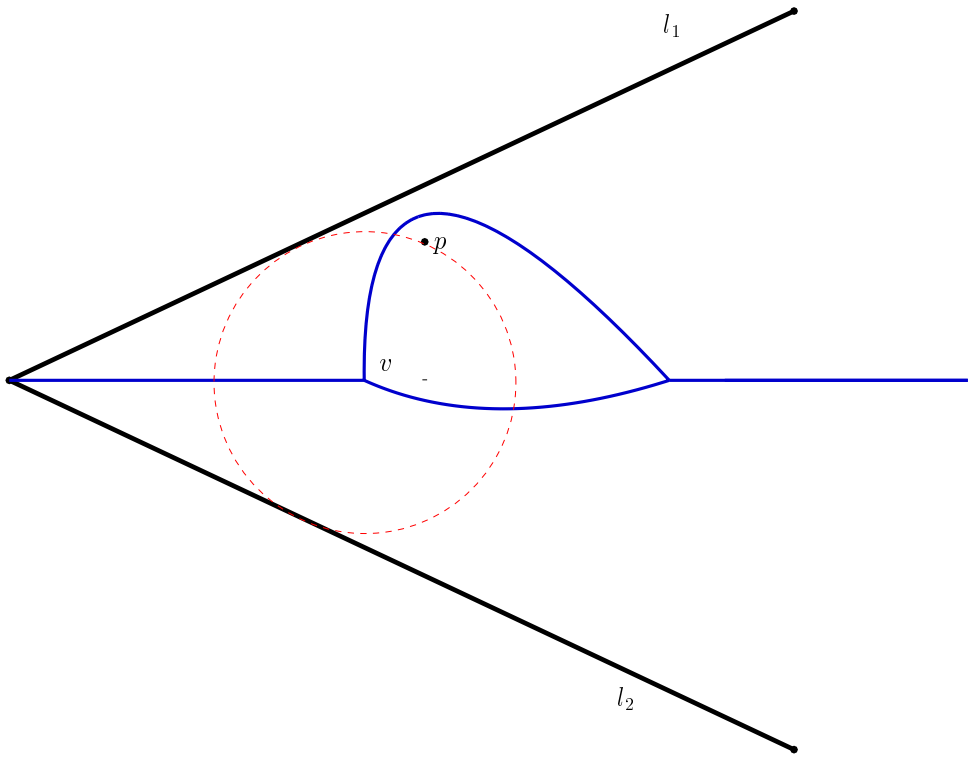
- About 10 years ago, I asked a student to implement an algorithm for Voronoi diagrams of line segments
  - he was a good student, has a PhD by now
- we found several algs in the literature
  - divide and conquer
  - sweep
  - randomized incremental
- all algs use a certain geometric primitive: the incircle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

A Typical Test: The Incircle Test



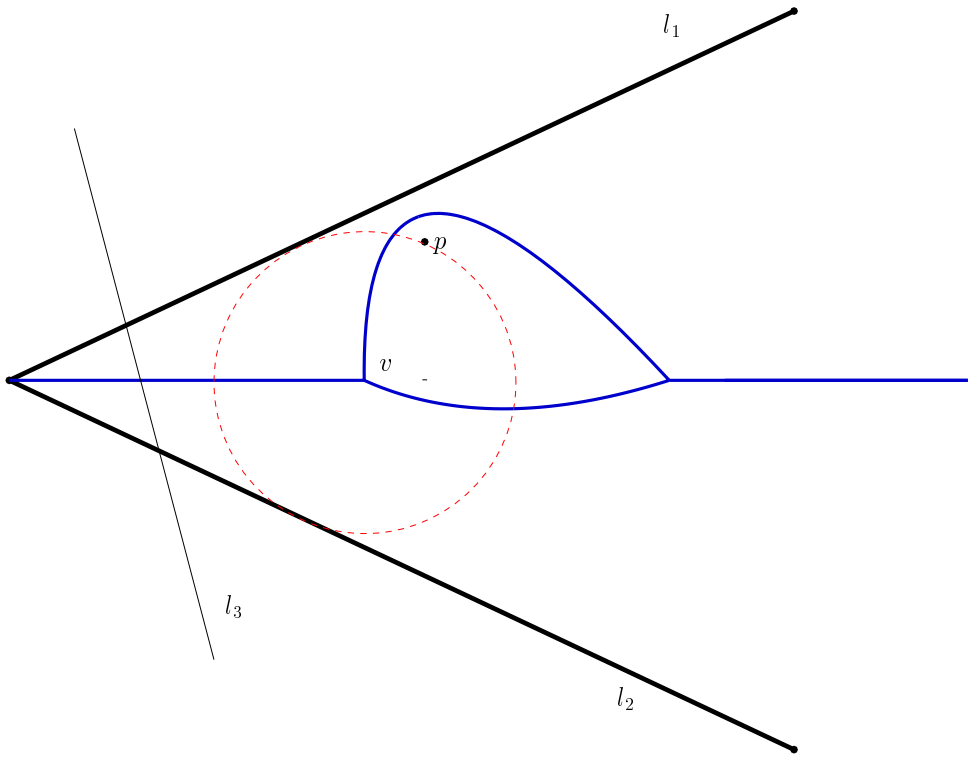
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$

A Typical Test: The Incircle Test



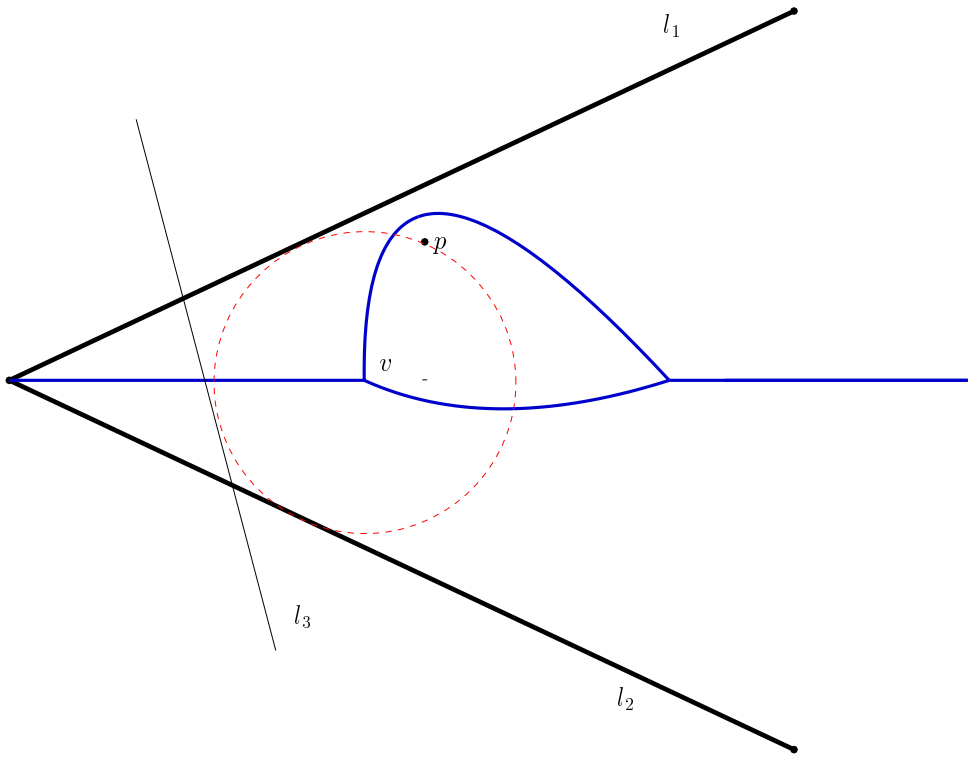
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$

A Typical Test: The Incircle Test



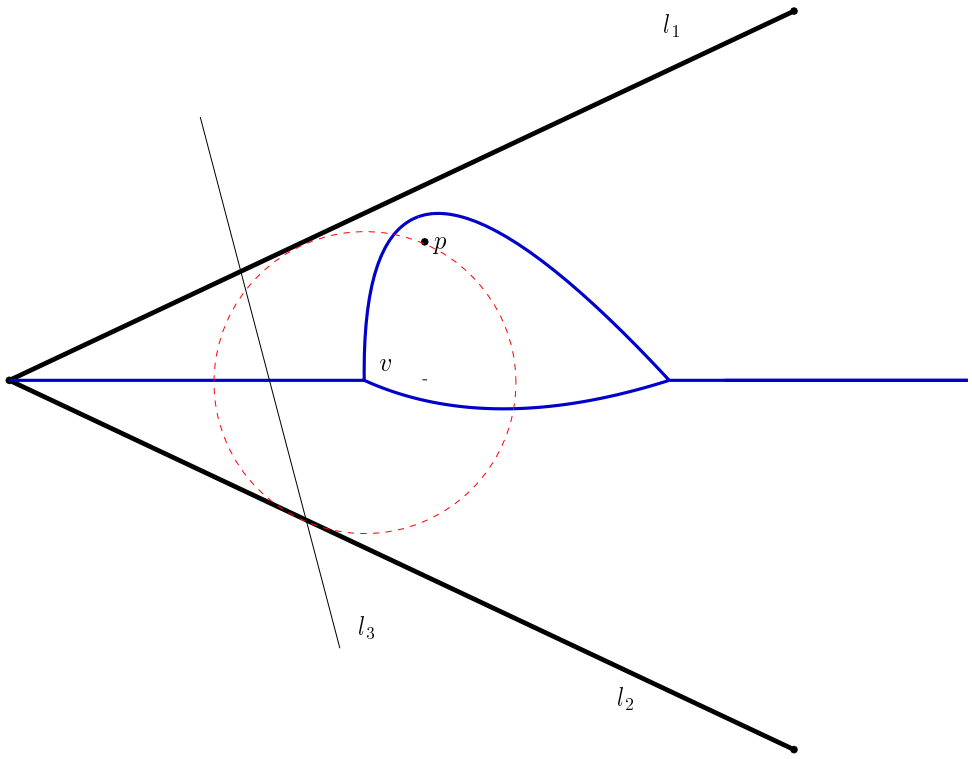
- v is defined by l_1, l_2, and p, i.e., $dist(v,p) = dist(v,l_1) = dist(v,l_2)$
- Add l_3. Is v still a Voronoi vertex?
- If $dist(v,p) < dist(v,l_3)$, **YES**

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 $dist(v,p) = dist(v,l_3)$, **YES, BUT...**

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**

Experiences

- None of the papers discussed the case **YES, BUT...**
  - They all started with: *We assume our input to be in general position.*
  - We discuss this issue in the section on Degeneracy
- None of the papers mentioned that it might be difficult to make the case distinction, i.e., to

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

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

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

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

Experiences

- None of the papers discussed the case **YES, BUT...**
- None of the papers mentioned that it might be difficult to make the case distinction, i.e., to

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

  - we implemented the algorithm using floating point arithmetic
  - and found a few examples where the program worked
  - cf. section on arithmetic and exact kernels
- None of the papers mentioned that alg is complex
  - cf. section on simple algorithms
- None of the papers mentioned that implementors make mistakes
  - cf. section on result checking
- We had nothing to build on
  - cf. section on systems (LEDA and CGAL).

Overview

- Introduction (20 min)
- Systems I: LEDA and CGAL (20 min)
- Demos (30 min)
- Arithmetic and Exact Kernels (60 min)
- Degeneracy and Algorithmic Issues (75 min)
- Simple Algorithms (randomized algs) (45 min)
- Result Checking (30 min)
- Systems II and Summary

Message of Tutorial

- Computational Geometry has addressed implementation issues over the past 10 years and
- has found (partial) solutions

CGAL and LEDA make the solutions available to YOU

What is LEDA?

- A library of combinatorial and geometric data types and algorithms
  - covers a large part of combinatorial and geometric computing
    (AHU, CLR, Mehlh, PS, Ko, Ki, BKOS, O'Rourke)
  - easy to use
  - extendible
  - correct
  - efficient
- provides algorithmic intelligence for applications in GIS, VLSI-design, scheduling, traffic planning, solid modelling, 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: bigints, bigfloat, rationals, algebraic numbers, 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, ...
- Exact Geometry Kernels: points, lines, circles, ...
- Geometric Algorithms: polygons and boolean operations on polygons (2d), convex hulls (dd), Delaunay diagrams (dd), Voronoi diagrams, sweep-line method (2d), curve reconstruction ...
- Visualization and I/O: windows, graphwin, geowin

Sample Applications

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

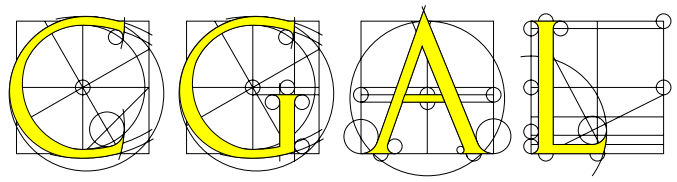
LEDA users

- academic users (= users that pay little)
  - 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, USA; CAD Design, USA; Comptel, Finnland; France Telecom, Frankreich; E-Plus Mobilfunk, Düsseldorf; Dolphin Software, Holland; ATR Telecommunications, Japan; Sun Microsystems, USA; Mentor Graphics, USA; Siemens AG; Lufthansa Systems, Kelsterbach; Daimler Benz, Berlin; Ford, USA; Aerospace, Muenchen; Silicon Graphics, USA; Digital Equipment, USA; Sony Corporation, Japan; Dainippon Screen, Japan; Minolta, Japan; Shinko, Japan; Chevron Petroleum, USA; Prediction Company, USA; Commerz Financial Products, Frankfurt; IBM, Japan; and many others

Demos

- 3d-convex hulls: handles one- and two-dimensional input sets
- boolean operations on polygons
- Delaunay triangulations, Voronoi diagrams, curve reconstruction
- range queries
- additional demos: depending on time and audience



Computational Geometry Algorithms Library

http://www.cgal.org

- ETH Zürich
- Freie Universität Berlin
- INRIA Sophia-Antipolis
- MPI Saarbrücken
- Tel Aviv University
- Universität Trier
- Utrecht University

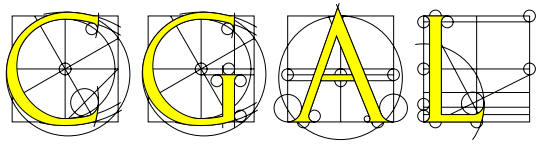
ESPRIT projects CGAL and GALIA.

Design Goals

- Efficiency
- Ease of use

```
Segment_2 s1(Point_2(0,0), Point_2(1,2));
Segment_2 s2(Point_2(1,0), Point_2(0,1));
if (do_intersect(s1, s2)) ...
```

- Correctness
  - produce the specified result
  - produce results robustly
- Flexibility
  - more than one algorithm
  - more than one kind of input and/or output



Geometry Kernels

Provide:

- classes for elementary geometric objects in 2D, 3D, (and dD soon):

  `Point_2, Direction_2, Isocuboid_3, Sphere_3`, ...

- predicates:

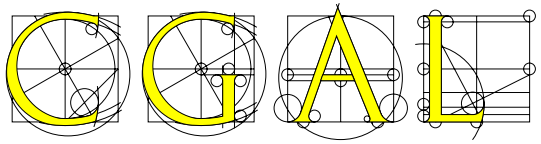
  `orientation_2, compare_x, side_of_bounded_sphere`, ...

- intersection computation and detection

  `do_intersect, intersect`

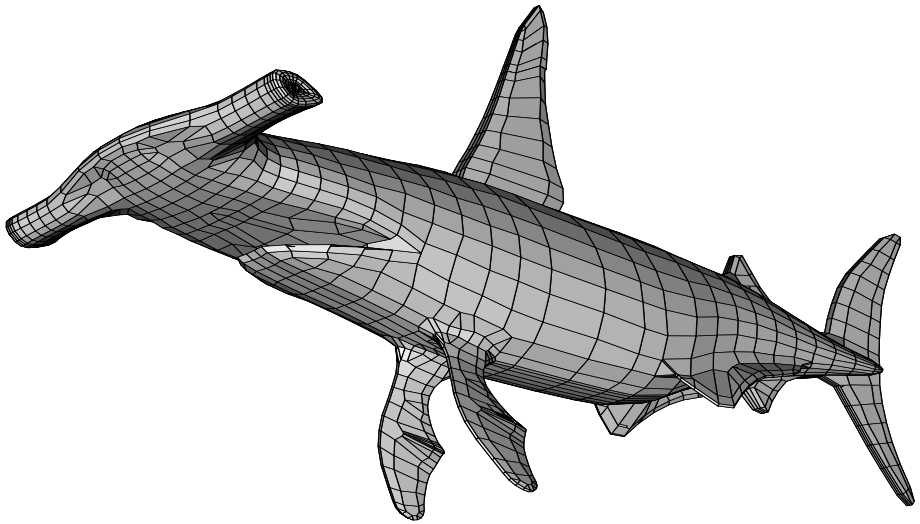
- distance computation and comparison

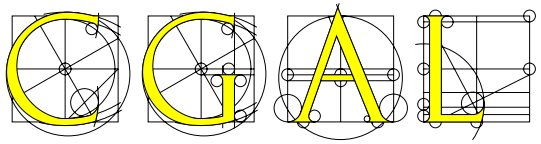
  `squared_distance, cmp_signed_dist_to_plane`, ...



Geometric Data Structures

- Search Structures
- Planar Maps
- Triangulations
- Halfedge Data Structure
- Polyhedral Surfaces
- ...

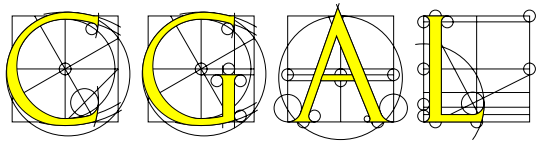




Geometric Algorithms

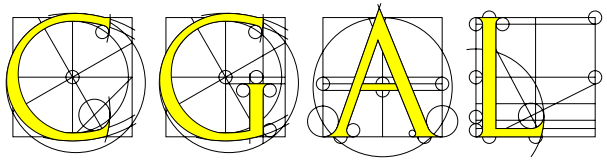
- Convex Hulls
- Voronoi Diagrams
- Point Location
- Optimization
- ...

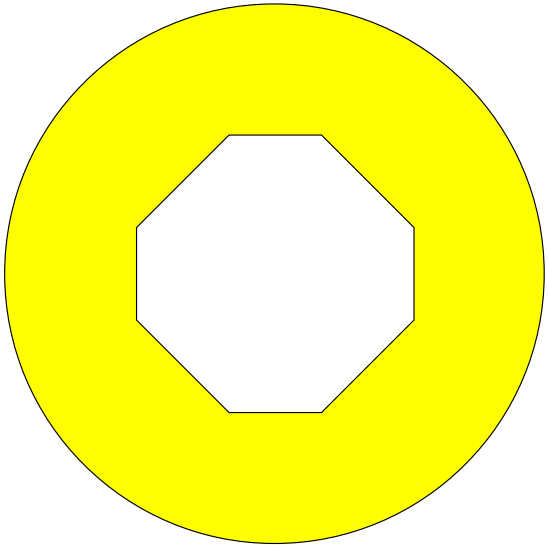
- Alpha Shapes
- Delaunay Triangulations
- Nearest Neighbor Queries
- Polygon Partitioning
- ...

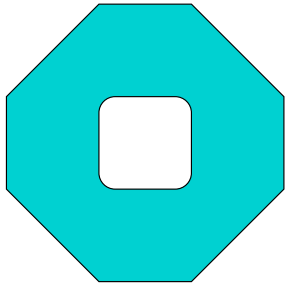


Support Library

- **Number Types**: `Interval_arithmetic`, `Quotient`, `Filtered_exact`, ...
- **Visualization**: Geomview, LEDA Windows, GeoWin, ...
- **Generators**: polygons, convex sets, points in sphere, points on sphere, ...
- **STL Extensions**: circulators, creators, in-place lists, ...
- ...







Traits
(Kernel + ...)

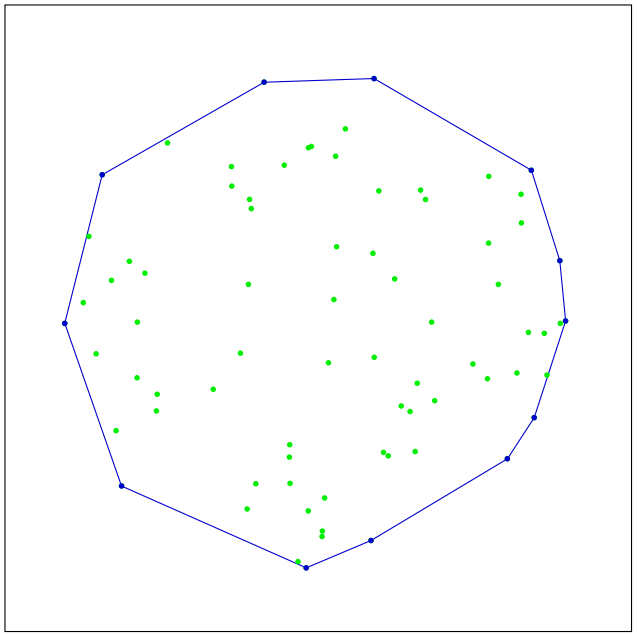
Representation

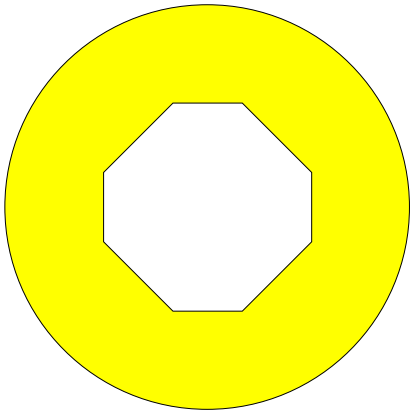
Arithmetic

Algorithms & Data Structures

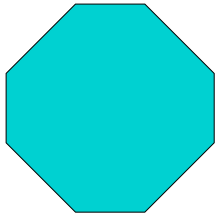
Generic Programming

Generic Algorithm Example





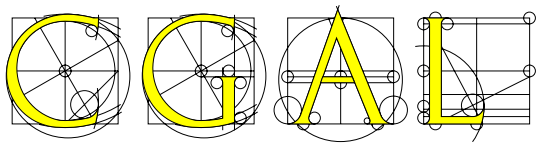
Convex Hull



Point

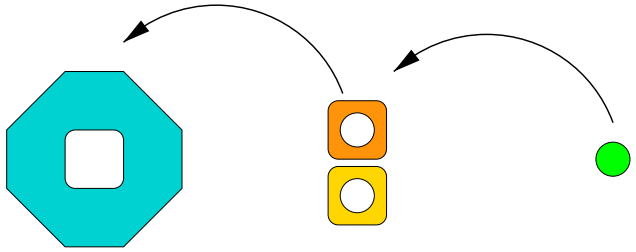
Less_xy

Left_turn



Geometry Kernels

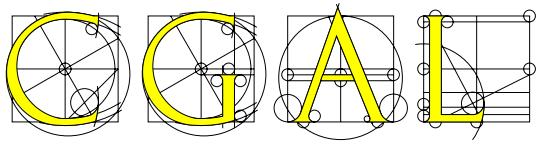
Parameterized by a number type



Cartesian	(x, y)
Homogeneous	(h_x, h_y, h_w)

Number type may be, for example:

`Filtered_exact`, `leda_rational`, `leda_real`, ...



Users

- Users from academia and industry
- Users from graphics, vision, GIS, robotics, CAD/CAM, Aerospace, ...
- Over 1500 downloads of latest release
- Over 700 members of `cgal-discuss-l` mailing list

Arithmetic and Exact Geometry Kernels

- Exact geometry kernels provide software objects that behave like their mathematic counterparts
- LEDA and CGAL provide exact and efficient kernels for "rational" geometry
- "rational geometry" = only $+$, $-$ $\times$, and $/$.
- rational geometry carries a long way:
  - points, lines, hyperpoints, hyperplanes, circles,
  - convex hulls, triangulations, Delaunay and Voronoi, geometric optimization, polygons, geometric search structures,
- non-rational geometry is evolving
- exact kernels are based on number types *integer*, *rational*, *bigfloat*, *filtered integer*, *real*,

A High-Level View of Geometric Computing

- takes numerical data, e.g., a set of n points given by their Cartesian coordinates.
- computes a combinatorial object, e.g., the Voronoi diagram or the convex hull, defined by the objects.
- branches based on geometric predicates, e.g., $orientation(p, q, r)$ or $in\_circle(p, q, r, s)$.
- tests are usually not independent
- may reach "impossible" states, if predicates are evaluated incorrectly, e.g.,
  - a point outside a polygon sees no edge of the polygon
  - a point outside a polygon sees all edges of the polygon
- algorithm will crash or compute nonsense.

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	ouput
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

Floating Point Arithmetic Destroys Geometry

The intersection point of two lines lies on both lines.

- create four points a, b, c, and d with random integer coordinates in $[0\,..\,10^4]$
- create the lines $l_1(a,b)$ and $l_2(c,d)$
- increment *count* if l_1 and l_2 are parallel or $p = l_1 \cap l_2$ lies on l_1
- repeat experiment 10^5 times

```
count = 0; max_coord = 10000;
for (i = 0; i < 100000; i++)
{ point a, b, c, d, p;
  random_point(a,max_coord);                  // same for b, c, and d

  line l1(a,b), l2(c,d);                      // create lines

  if ( !l1.intersection(l2,p) || l1.contains(p) )  count++;
     // if l1 and l2 are parallel or intersection point lies on l1
}
cout << count; // outputs 33178
```

and with LEDA's Exact Geometry Kernel

```
count = 0;  max_coord = 10000;
for (i = 0; i < 100000; i++)
{ rat_point a, b, c, d, p;              // !!!!
  random_point(a,max_coord);

  rat_line l1(a,b), l2(c,d);            // !!!!

  if ( !l1.intersection(l2,p) || l1.contains(p) ) count++;
}
cout << count; // outputs 100000
```

- CGAL's and LEDA's exact geometry kernel **guarantee** that software objects behave like their mathematical counterparts
- overhead in running time is a small constant factor

Exact Evaluation of Predicates

- amounts to computing the sign of expressions, e.g.,

$$orientation(p,q,r) = sign \begin{vmatrix} 1 & 1 & 1 \\ x_p & x_q & x_r \\ y_p & y_q & y_r \end{vmatrix}$$

- sign evaluation
  - for rational expressions
    - floating point filter
    - exact evaluation with rational arithmetic as fall back
  - for algebraic expressions:
    - separation bounds
    - sign test = numerical evaluation + separation bounds
- high level predicates:
  - which face contains the point p?

Floating Point Filter

To determine $sign(E)$

- evaluate with floating point arithmetic: $\tilde{E}$
- compute an error bound B
- if $(|\tilde{E}| > B)$ return the sign of $\tilde{E}$
- otherwise, evaluate E with exact arithmetic and return the sign obtained

Insight: sign determination is possible with approximate value computation

- all geometric predicates in LEDA and CGAL are filtered in this way.
- generation of filters is automated (expression compiler by Stefan Funke)
- experience: exact evaluation is rarely needed

Error Bound Computation

- we use a clever variant of interval arithmetic
- $B = 2^{-53} \cdot ind_E \cdot mes_E$
- ind_E is precomputed from the structure of E.
- mes_E is computed on-line

E	$\tilde{E}$	mes_E	ind_E
a, integer	$fl(a)$	$\lvert fl(a)\rvert$	1
a, float int	$fl(a)$	$\lvert fl(a)\rvert$	0
$A + B$	$\tilde{A} \oplus \tilde{B}$	$mes_A \oplus mes_B$	$1 + \max(ind_A, ind_B) \cdot \delta$
$A - B$	$\tilde{A} \ominus \tilde{B}$	$mes_A \oplus mes_B$	$1 + \max(ind_A, ind_B) \cdot \delta$
$A \cdot B$	$\tilde{A} \odot \tilde{B}$	$mes_A \odot mes_B$	$(*)$

$$(*) = 1 + (ind_A + ind_B + 2^{-53} \cdot ind_A \cdot ind_B) \cdot \delta \qquad \delta = 1 + 2^{-53}$$

- standard interval arithmetic is an alternative, slightly slower

Efficacy of Floating Point Filter

	Orientation			Side of circle		
d	number	exact	%	number	exact	%
8	130431	0	0.00	64176	0	0.00
10	147814	0	0.00	77409	136	0.18
12	149233	0	0.00	78693	105	0.13
22	149057	0	0.00	78695	113	0.14
32	149059	0	0.00	78695	115	0.15
42	149059	0	0.00	78695	115	0.15
∞	149059	0	0.00	78695	115	0.15

Delaunay diagram computation for 10000 random points in the unit square.

d = precision (number of binary places) used for the Cartesian coordinates

number of tests, number (percentage) of tests requiring exact arithmetic

Efficiency of Floating Point Filter

d	Float kernel	Rat kernel	RK without filter
8	2.58	3.59	16.33
10	2.8	3.98	18.36
12	2.83	4.04	18.63
22	2.82	4.02	20.51
32	2.86	3.96	20.77
42	2.83	4.01	26.02
∞	2.83	3.99	33.2

10000 random points in the unit square

d = the precision (number of binary places) used for the Cartesian coordinates

running time in seconds for floating point kernel, for rational kernel with floating point filter, for rational kernel without floating point kernel

Sign Tests of Algebraic Expressions

determine the sign of algebraic numbers given by algebraic expression dags

Examples:

- determine sign of $\sqrt{17}+\sqrt{21}-\sqrt{\sqrt{17}+\sqrt{21}+2\sqrt{361}}$
- compare $\frac{17+\sqrt{21}}{19}$ and $\frac{18+\sqrt{22}}{20}$

Motivation

- evaluation of geometric predicates (incircle, side-of) amounts to sign determination
- non-linear objects and distances lead to expressions involving roots
- want a general method for evaluating geometric predicates
- **Alternative:** methods for specific predicates
  BMS (ESA 94), Devillers/Fronville/Mourrain/Teillaud (SoCG 2000))

The Intersecting Circular Arcs Demo

Input: A set of circular arcs

- circular arcs are parts of circles
- endpoints = intersections between circles and lines

Output: The arrangement defined by the arcs

The Separation Bound Approach

- a *separation bound* for a class of expressions is an easily computable function *sep* mapping expressions into positive real numbers such that

  $$\text{for every expression } E: \qquad val(E) \neq 0 \quad \Longrightarrow \quad |val(E)| \geq sep(E)$$

- separation bound yields sign test based on numerical computation

  $\epsilon \leftarrow 1;$
  while (true)
  { compute an approximation $\tilde{E}$ with $|val(E) - \tilde{E}| < \epsilon$;
  if $(|\tilde{E}| \geq \epsilon)$ return $sign(\tilde{E})$;
  if $(\epsilon < sep(E)/2)$ return "sign is zero"; // since $|val(E)| \leq \epsilon + \epsilon < sep(E)$
  $\epsilon \leftarrow \epsilon/2;$
  }

- worst case complexity is determined by separation bound:

  maximal precision required is logarithm of separation bound (bit bound)

- easy cases are decided quickly (a **big** plus of the separation bound approach)

The Separation Bound Approach

- a *separation bound* for a class of expressions is an easily computable function *sep* mapping expressions into positive real numbers such that

$$\text{for every expression } E: \qquad val(E) \neq 0 \quad \Longrightarrow \quad |val(E)| \geq sep(E)$$

- Mignotte (82), Canny (87), Yap/Dube (95), BFMS (97), Scheinermann (00), Li/Yap (01), BFMS (01)
- separation bound approach to sign determination is used in
  - number type *Expr* in the CORE-package (Dube/Li/Yap)
  - number type *real* in LEDA

The BFMS-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

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

A Consequence

If $E \neq 0$ then one of the first $2^{k(E)} \log u(E)$ binary places of E is nonzero and hence determines its sign.

Example I

$$x = \text{some integer with } L \text{ binary places}$$

$$A = (\sqrt{x+1} + \sqrt{x}) \cdot (\sqrt{x+1} - \sqrt{x})$$

$$B = A - 1$$

Then

$$u(B) \approx u(A) \approx 4x \qquad \text{and thus} \qquad \log u(B) = \log u(A) \approx 2 + L$$

and hence to decide whether $B = 0$ it suffices to look at the first

$$4L$$

binary places of B.

The LEDA Program Corresponding to Example I

The theorem is packaged in the LEDA data type *real*. It provides exact arithmetic for arithmetic expressions involving square 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 have been used successfully in several geometric programs, e.g., Voronoi diagrams of line segments, arrangements of circular arcs, ...

Example 2

$$A = \sqrt{\ldots\sqrt{(\ldots(2^2)^{\cdots})^2 + 1}} - 2$$

To decide that $A \neq 0$ it suffices to look at the first $2^k \log 4$ binary places of A.

Remark: First nonzero bit of A is about 2^k places after binary point.

More Discussion

- $k(E) = 0$: $E = 0$ or $E \geq 1$.
- The bound is almost sharp

$$\sqrt{\ldots\sqrt{(\ldots(2^2)\ldots)^2+1}}-2$$

$$(4,...)^{1-2^k} \leq E \leq 2^{-2^k}$$

- Our bound is always at least as good as the bounds by Mignotte (75) and Canny (85) and sometimes much better.

  For example, for the expression of the previous item:

	our bound	Canny's bound
general k	$2^k \log 4$	$2k \cdot 4^k$
$k = 10$	2000	20 000 000

  However, Canny's bound also applies to more general situations.

Proof of BFMS-Bound for Division-Free Expressions

E = an expression, integral operands, operations $+, -, *, \sqrt[k]{\ }$ for integral $k \geq 2$.

$D(E)$ = product of the indices of the radical operations in E. (the index $\sqrt[k]{\ }$ is k.)

	$u(E)$
integer N	$\lvert N \rvert$
$E_1 \pm E_2$	$u(E_1) + u(E_2)$
$E_1 \cdot E_2$	$u(E_1) \cdot u(E_2)$
$\sqrt[k]{E_1}$	$\sqrt[k]{u(E_1)}$

Then $val(E) = 0$ or

$$\left(u(E)^{D(E)-1}\right)^{-1} \leq |val(E)| \leq u(E).$$

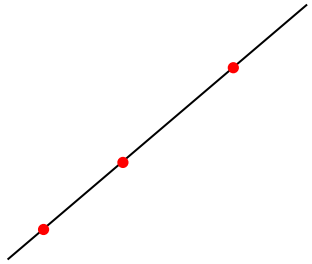
- $u(E)$ is an upper bound on the value of E and all its conjugates
- the product of $val(E)$ times all its conjugates is an integer
  (namely the constant coefficient of the minimal polynomial)
- thus

$$val(E) \cdot u(E)^{D(E)-1} \geq 1$$

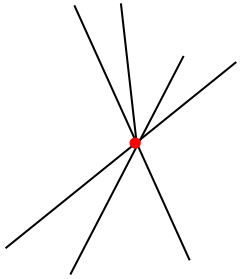
Algorithms vs. Implementations

Fact #1: Input to algorithms is quite often assumed to be void of degeneracies:

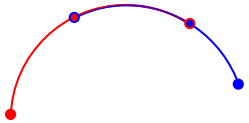
collinear points

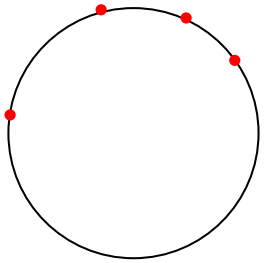


common intersection

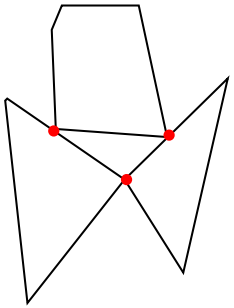


overlapping objects





cocircular points



vertex adjacencies

and many others . . .

Algorithms vs. Implementations

Fact #2: Input to programs that implement these algorithms is often NOT void of degeneracies.

Result: Programs may produce

- incorrect results,
- imprecise results,
- or NO results (*i.e.*, the program crashes),

which are generally unacceptable outcomes.

Robust Algorithmic Approaches

- Treat each degenerate case separately.
- Perturb input to achieve general position .
- Reformulate general-position algorithms so degeneracies are not special .
- Assure topological consistency of geometric structures .

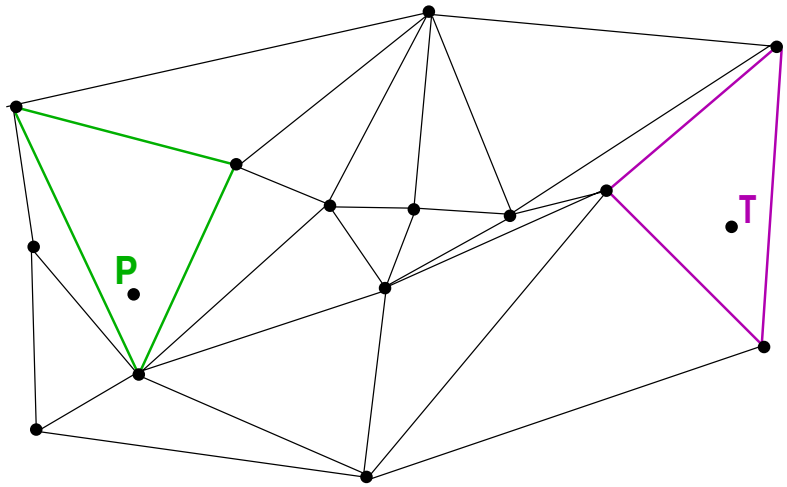
Example: Walking through a Triangulation

Given:

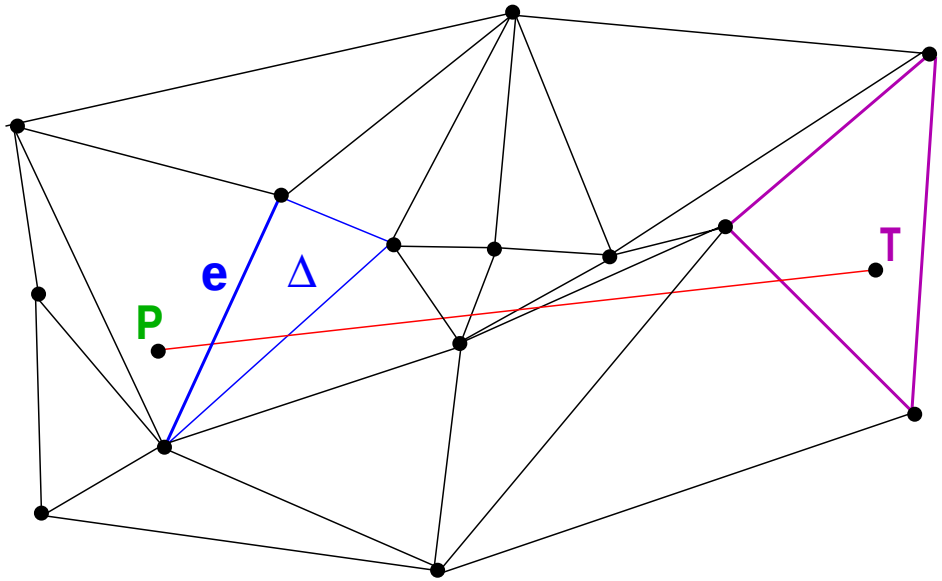
- a triangulation
- a point P contained in a triangle Δ_P
- a second point T

Find:

- the triangle Δ_T containing T.



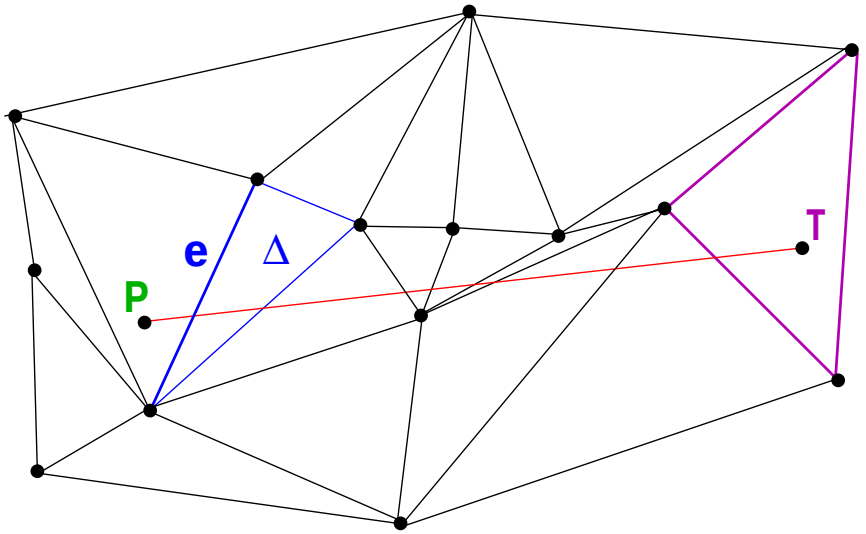
General-Position Approach



- Assume no three points are collinear.
- Walk along segment $s = PT$ maintaining the following invariants:
  - Edge e is an edge whose relative interior is intersected by s.
  - Triangle Δ is incident to e and lies in the same halfspace as T with respect to e.

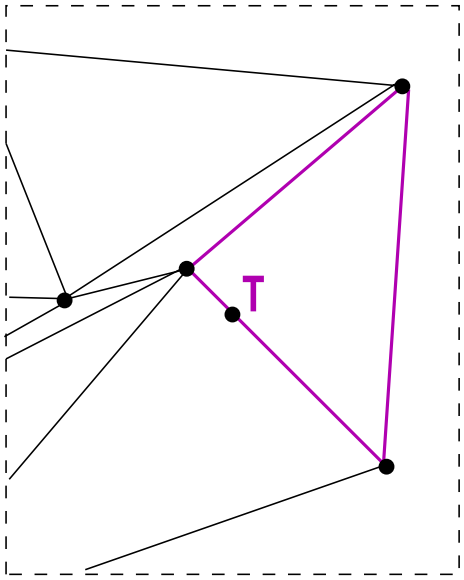
General-Position Algorithm

1. if $T \in \Delta_P$, done;
2. e = edge of Δ_P intersected by s;
3. Δ = other triangle incident to e;
4. while ($T \notin \Delta$) {
   e = other edge of Δ intersected by s;
   Δ = other triangle incident to e;
   }



Possible Problems

T is on an edge of the triangulation.



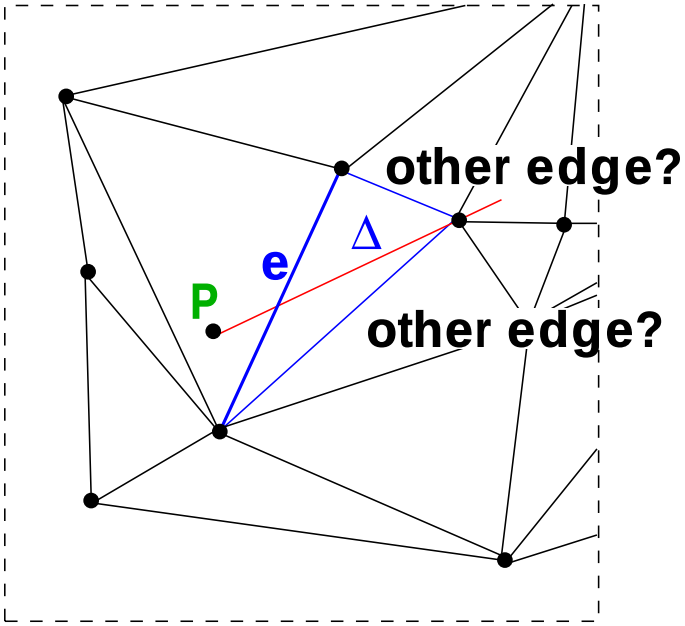
No problem. Change the test

$$T \notin \Delta$$

to include the boundary of Δ:

$$T \notin \overline{\Delta}$$

s passes through a vertex



Problem.

e = other edge of Δ ...

Which is the "other edge"?

The Special Cases Approach

Change the algorithm to include the following:

If s passes through a vertex then ...

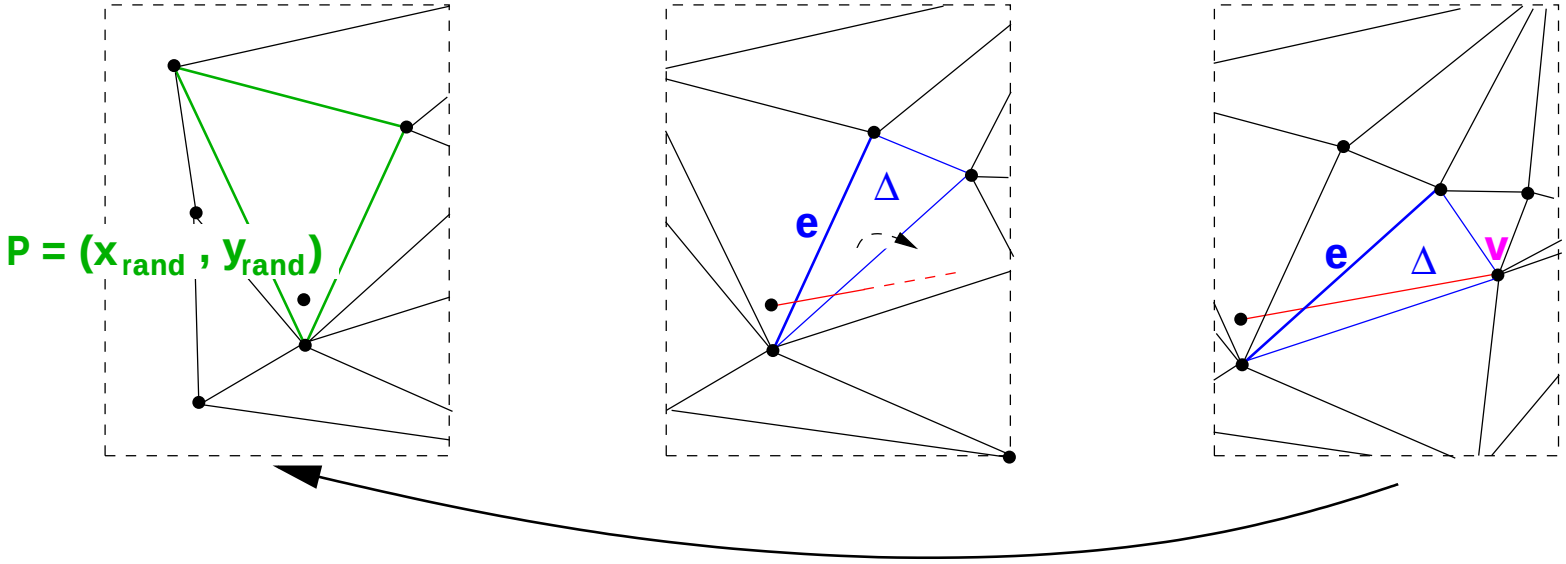
If s contains an edge of the triangulation then ...

Notice:

- Detailing all the special cases is generally difficult and leads to complicated code.
- Difficulty of maintaining invariants of a general-position algorithm in the face of degeneracies can lead to errors.

Conclusion: The special cases approach is generally not a good idea.

A Randomized Approach



1. Choose starting point P at random from Δ_P.
2. Run the general-position algorithm.
3. If a vertex v is crossed, restart at step 1.

A Randomized Approach

Claim: With high probability a random choice of P will work.

Why?

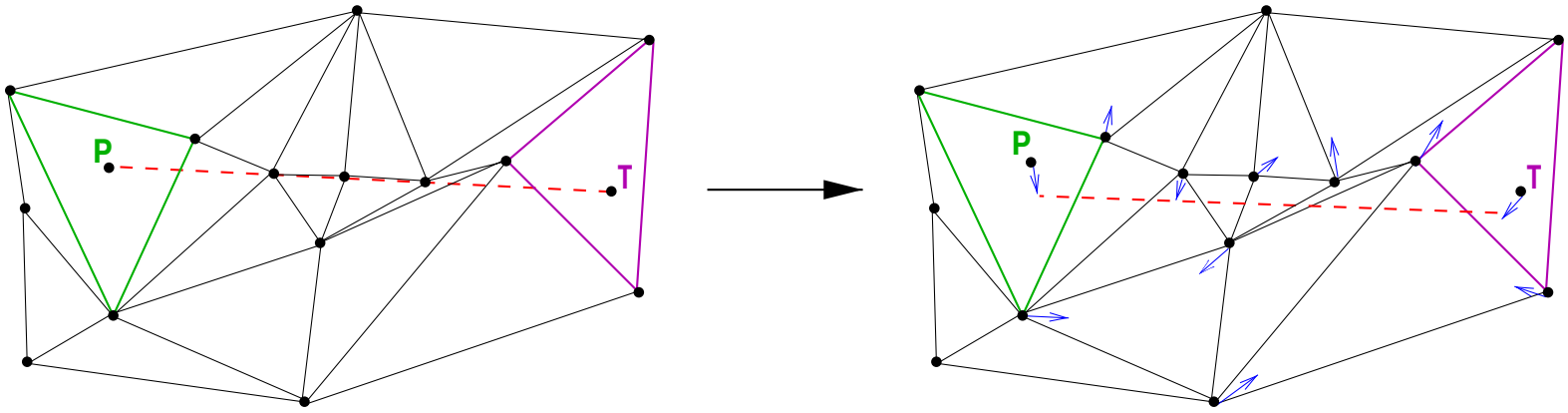
- Each vertex eliminates a segment in Δ_P
- Finite number of vertices
- Δ_P has positive area and the segments have 0 area.

$\Rightarrow$ Almost all points in Δ_P will work

Conclusion: Theoretical claim is true, but in practice . . . ???

A Perturbation Approach

1. Move the input by an infinitesimal amount so the degeneracies disappear.

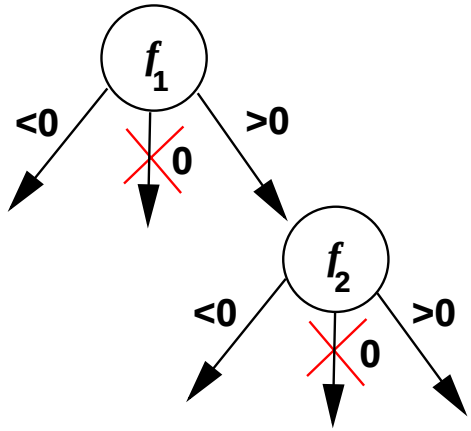


2. Run the general-position algorithm on the perturbed input.

3. Extract the results for the original input.

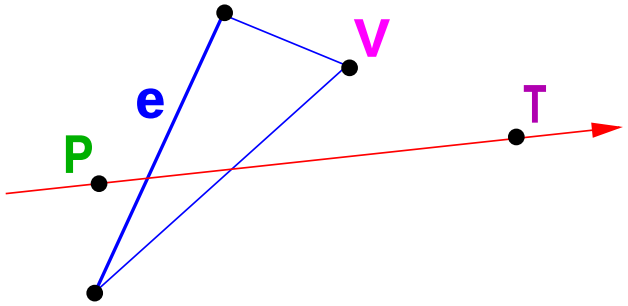
Valid Perturbations

Valid perturbations remove the 0 branches from the computation tree.



- Each predicate in the computation tree is implemented as a function that determines the sign of a polynomial.
- Instead of moving the input points, one actually need only "perturb" the predicates that operate on the input.

Perturbing a Predicate



Example: The orientation of $V = (V_x, V_y)$ with respect to the directed line from $P = (P_x, P_y)$ to $T = (T_x, T_y)$ is determined by evaluating $sign(D)$:

$$D = det \begin{pmatrix} 1 & 1 & 1 \\ P_x & T_x & V_x \\ P_y & T_y & V_y \end{pmatrix}$$

Perturbing a Predicate

For a small positive ϵ, create new points:

$$P' = (P_x + \pi_P(\epsilon), P_y + \pi_P(\epsilon))$$
$$T' = (T_x + \pi_T(\epsilon), T_y + \pi_T(\epsilon))$$
$$V' = (V_x + \pi_V(\epsilon), V_y + \pi_V(\epsilon))$$

Then the orientation predicate should determine $sign(D')$:

$$D' = det \begin{pmatrix} 1 & 1 & 1 \\ P'_x & T'_x & V'_x \\ P'_y & T'_y & V'_y \end{pmatrix} = D + g(\epsilon)$$

where $g(\epsilon)$ is a polynomial in ϵ.

Perturbing a Predicate

Orientation of perturbed points determined by:

$$sign(D') = sign(D + g(\epsilon))$$

For sufficiently small ϵ, we have:

- for non-degenerate cases ($D \neq 0$)

$$sign(D') = sign(D)$$

- for degenerate cases ($D = 0$)

$$sign(D') = sign(\text{first non-zero coefficent of } g(\epsilon))$$

Notice: ϵ does not actually appear in the computation.

Perturbations – When Do They Work?

- The pertubation functions $\pi(\epsilon)$ must guarantee that

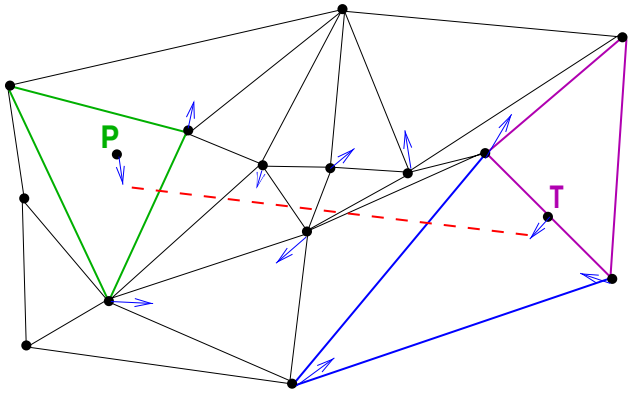
$$D' \neq 0$$

  - linear functions work with high probability
  - choice of functions $\pi(\epsilon)$ is main difference among different perturbation techniques

- When original program computes a continuous mapping from input to output, pertubed computation computes correct result.

Perturbations – The Costs

- When original mapping is not continuous, output may not be correct:



  but one hopes it is close to correct.

- Sometimes significant postprocessing required to recover correct result for original input .
- Evaluation of perturbed predicates involves more comparisons.
- Sometimes perturbing input makes problem more complicated .

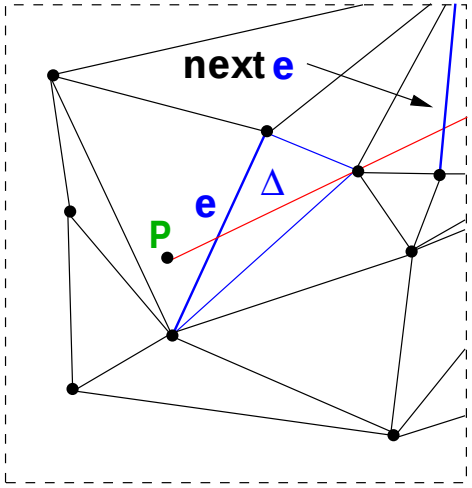
Perturbations – The Benefits

- General-position algorithm can be applied without modification.
- Rapid prototyping possibilities.
- For certain problems, a correct result is guaranteed even in the face of degeneracies.

Conclusion: Perturbation is a powerful technique in certain circumstances.

Refined Invariants Approach

Idea: Define invariants that can easily be maintained in the face of degeneracies.

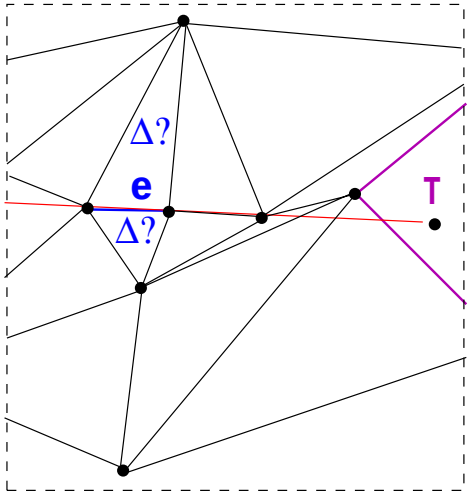


Original Invariant: Edge e is an edge whose relative interior is intersected by s.

Problems:

- next e satisyfing this may not be and edge of Δ
- including endpoints of e in invariant means next e is not well defined

Refined Invariants Approach

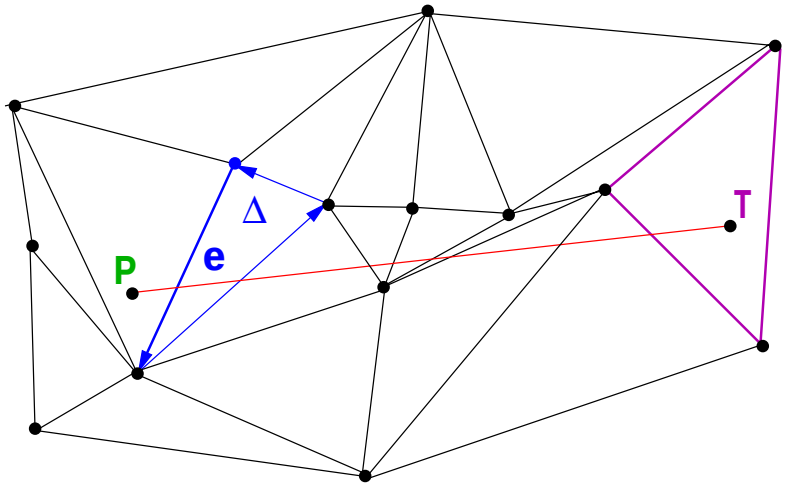


Original Invariant: Triangle Δ is incident to e and lies in the same halfspace as T with respect to e.

Problem:

- when e and s overlap Δ not well defined

Refined Invariants

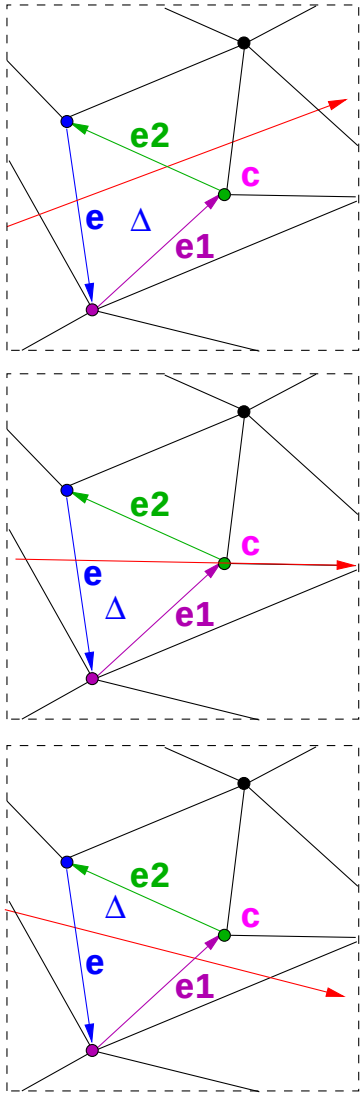


Maintain an oriented edge e and a triangle Δ such that:

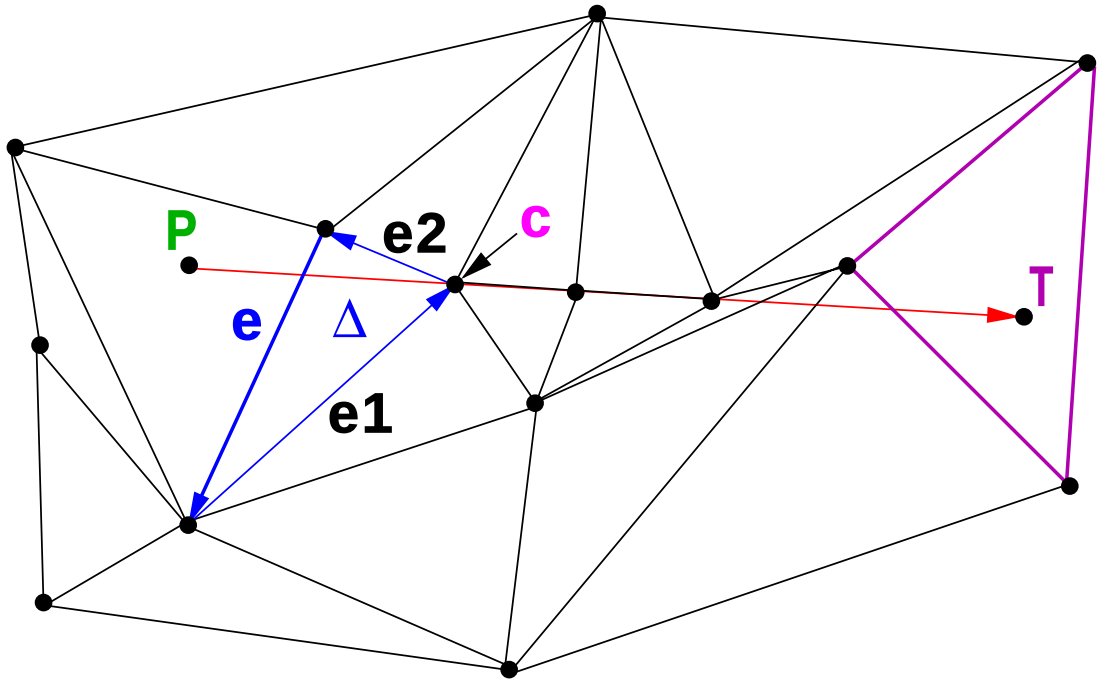
- Δ is the triangle to the left of e
- T lies to the left of e and P lies to the right of e
- s intersects the half-closure of e (*i.e.*, relative interior or the source)

The Refined Algorithm

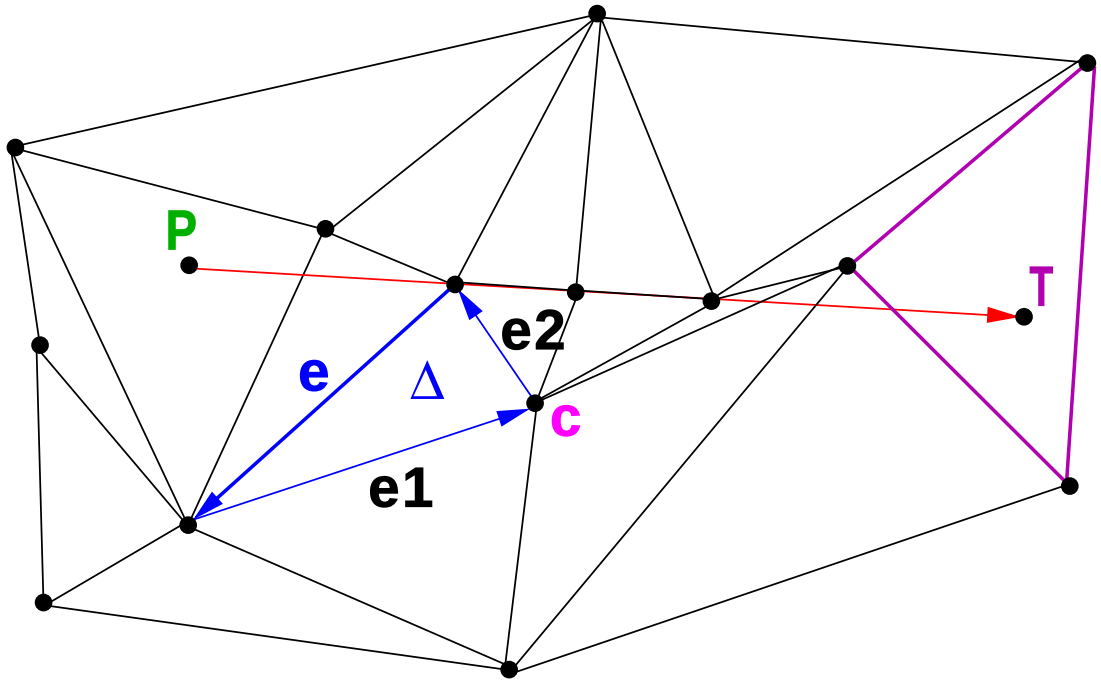
1. if $T \in \overline{\Delta_P}$, done;
2. e = half-edge of Δ_P intersected by s;
3. Δ = triangle incident to $reverse(e)$;
4. while ($T \notin \overline{\Delta}$) {
   c = vertex opposite e
   if (c right of s) then
   $e = reverse(e_2)$
   Δ = triangle right of e_2
   else
   $e = reverse(e_1)$
   Δ = triangle right of e_1
   }



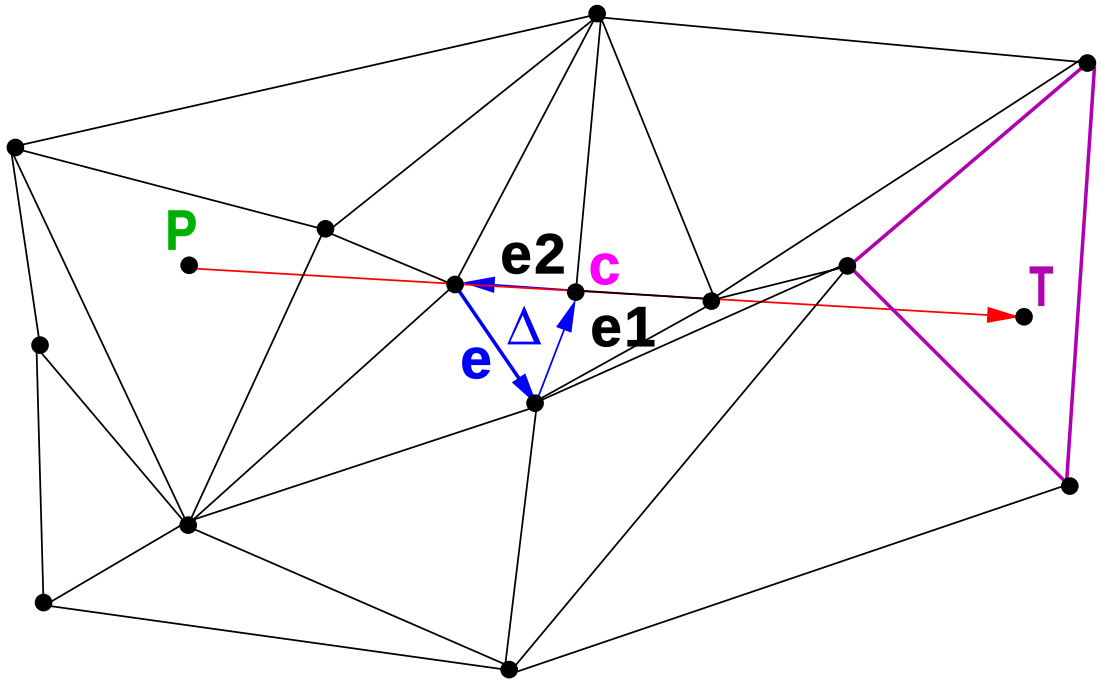
The Refined Algorithm



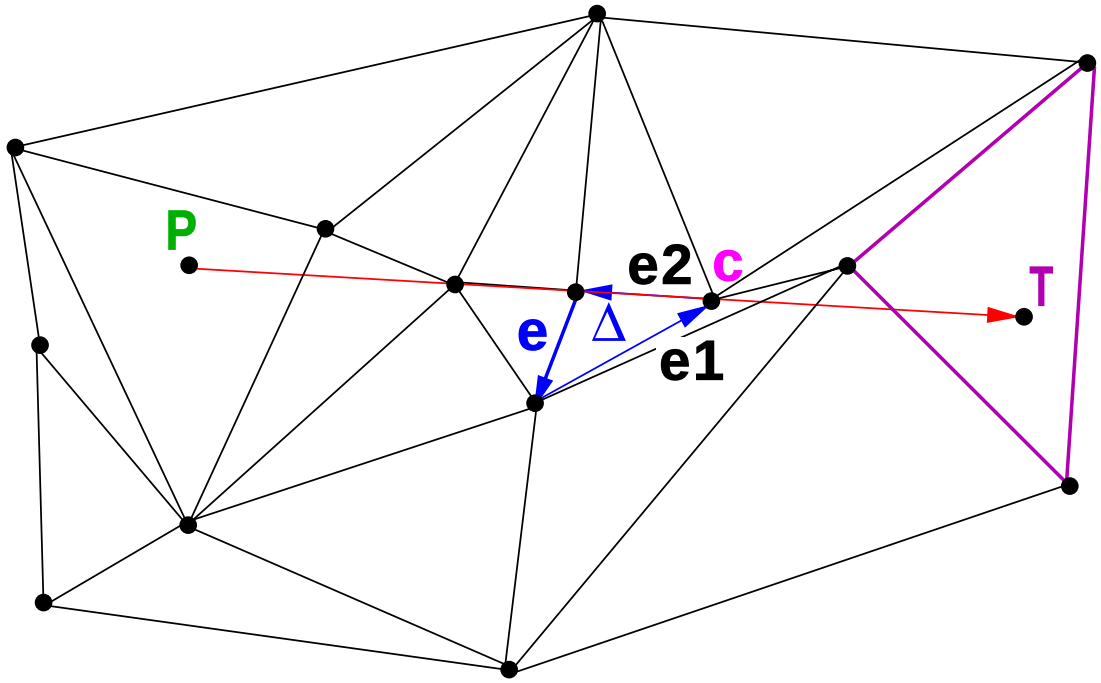
The Refined Algorithm



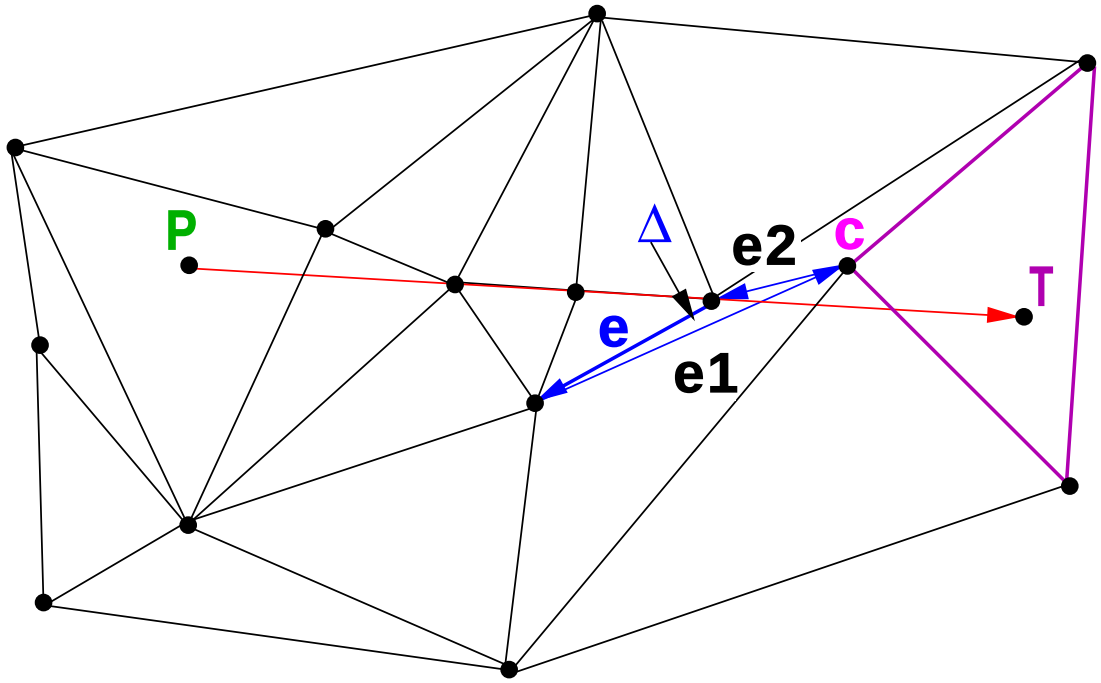
The Refined Algorithm



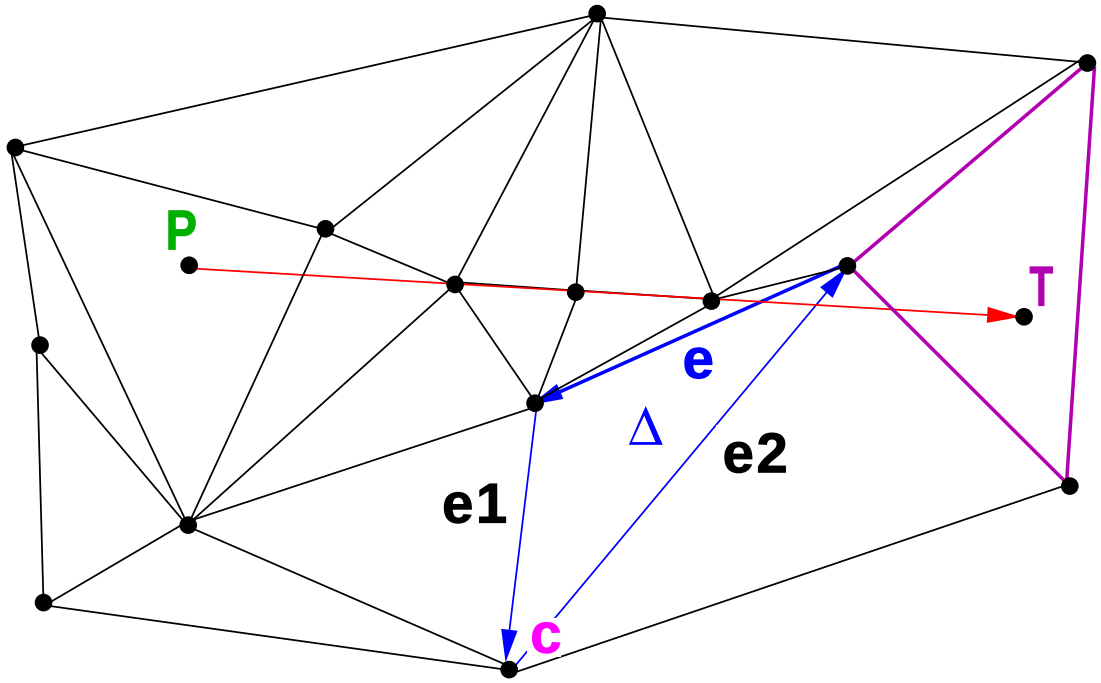
The Refined Algorithm



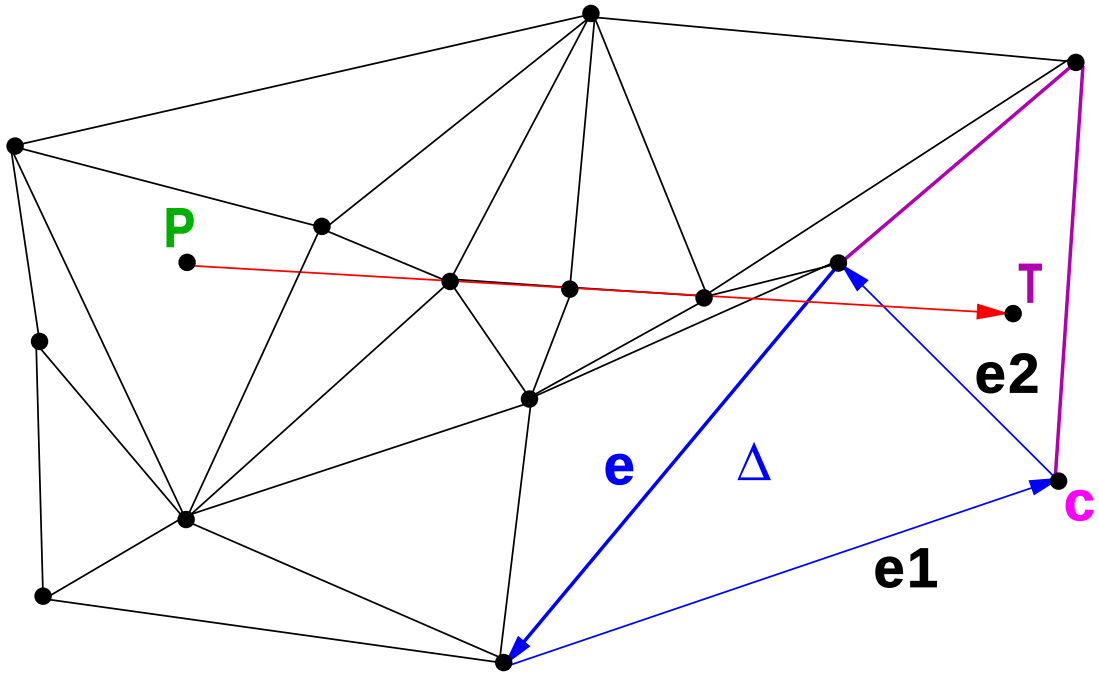
The Refined Algorithm



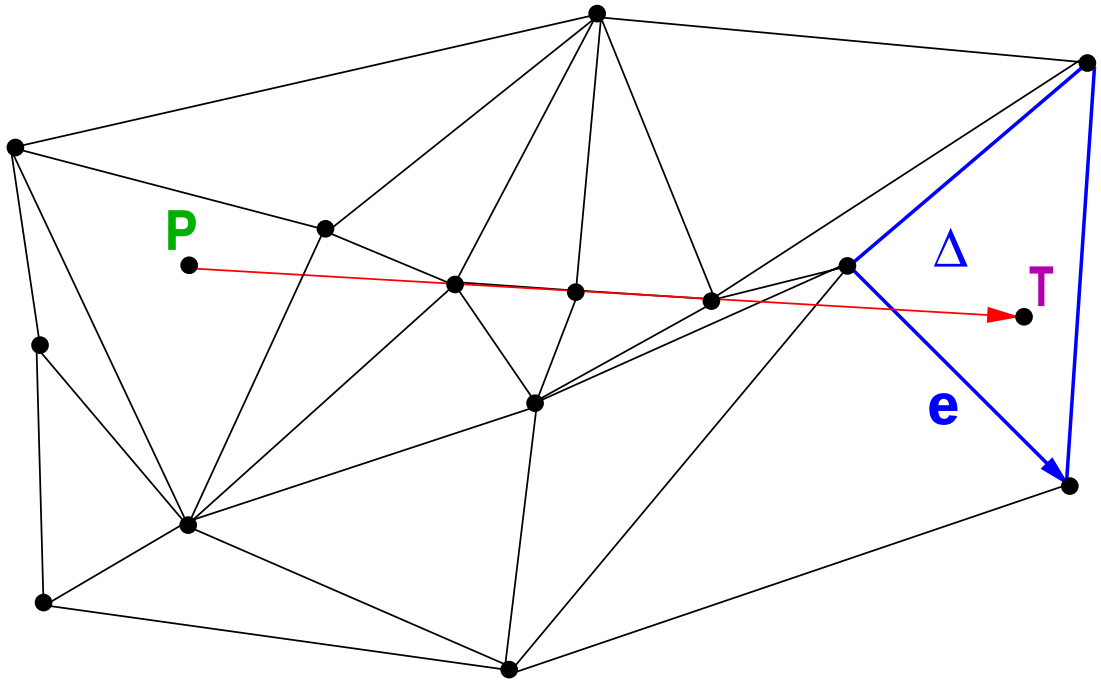
The Refined Algorithm



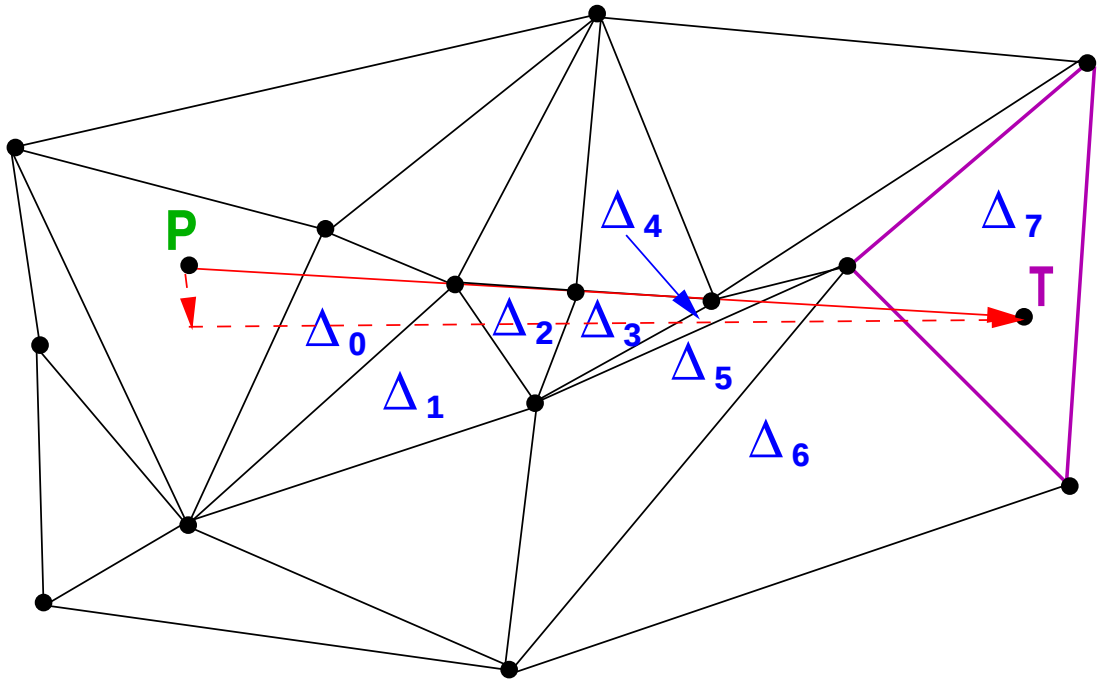
The Refined Algorithm



The Refined Algorithm



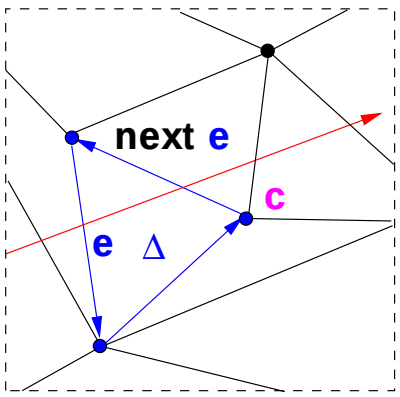
The Refined Algorithm

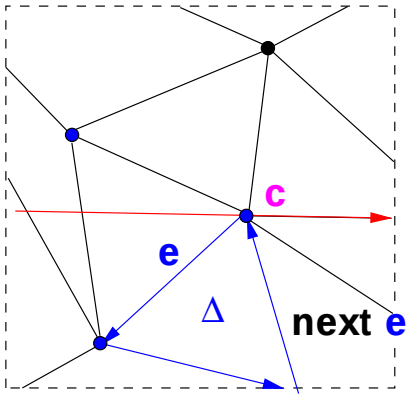


Observe:

- Walk through triangles as if the input line were perturbed downward a bit.
- Edge e is never collinear with s.

The Refined Algorithm – Why Does It Work?





- Always move closer to T because the next e
  - intersects s closer to T
  - OR makes a smaller angle with s.
- Degeneracy (3 collinear points) included in the invariant:
  - s intersects the half-closure of e

Refined Invariants Approach

- Careful consideration of degeneracies incorporated into the invariants.
- The refined algorithm:
  - can be equally as simple as the general-position algorithm
  - works for all input without fail.
- Must be applied to each problem separately.

Conclusion: With better invariants, one can achieve absolutely robust algorithms that are easy to implement.

Topological Consistency Approach

Idea:

- Consider structure (topology) before numerics
- Use numerics only to choose between multiple consistent structures.

Results:

- Algorithms that always produce some output no matter how imprecise the numerics.
- Output converges to correct result as precision increases.

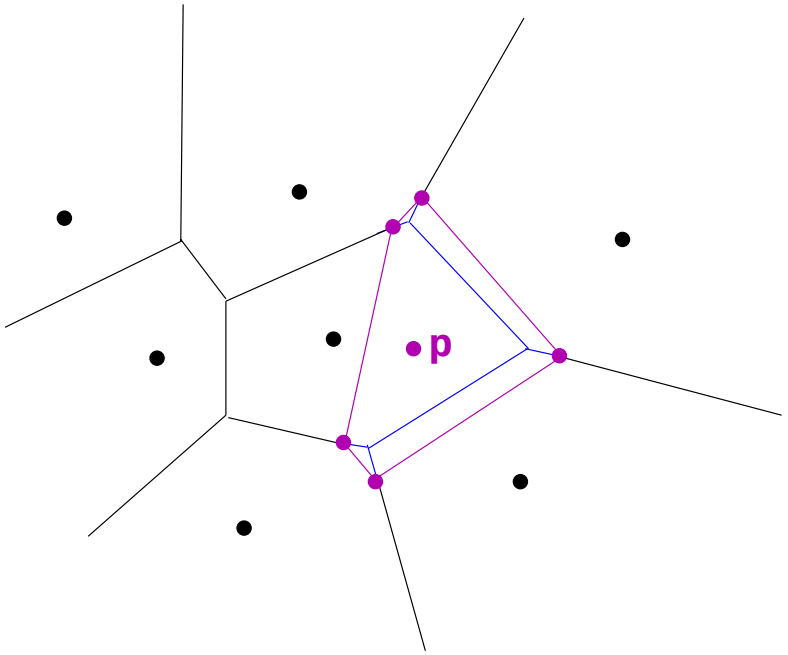
Incremental Construction of Voronoi Diagram

For each point p:

- Construct region R_p for new point p from bisectors with existing points.
- Remove the edges contained in R_p

Facts:

- The edges removed form a tree.
- The edges removed from each region are connected.

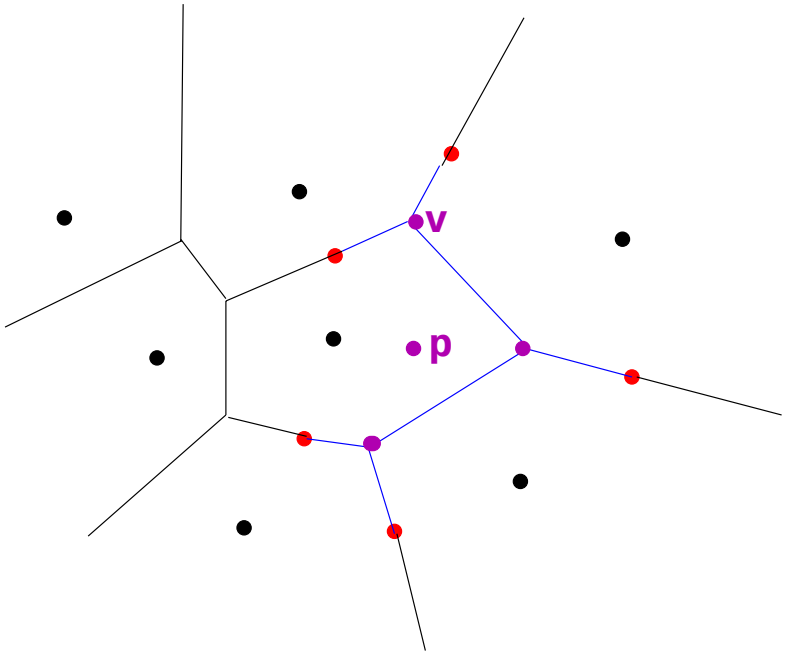


Topologically Consistent Construction

View Voronoi diagram as a graph.

- find a "seed vertex" v
- from v find a set of connected vertices V_0
- generate new nodes in graph on the cut edges for the subtree induced by V_0

Use numerics to decide which vertices to remove.



Topologically Consistent Algorithms

- Enforce structural invariants in their construction
- Shift emphasis from (often faulty) numerics to combinatorics
- Degeneracies are not special since these are defined by numerics
- Equally as correct as non-topologically oriented algorithms
- Results degrade nicely in the presence of imprecision.

Conclusion: Imposing structural invariants can make algorithms even more robust.

Handling Degeneracies in CGAL and LEDA

- Algorithms are implemented to handle all degeneracies.
- Geometry kernels provide exact predicates upon which to build robust algorithms.
- Flexibility of CGAL allows user
  - to provide a kernel of perturbed predicates
  - to employ a number type to implement perturbations .
  - to observe behavior of topologically consistent algorithms with various levels of numerical precision.

Some Famous Geometric Algorithms are Quite Complicated

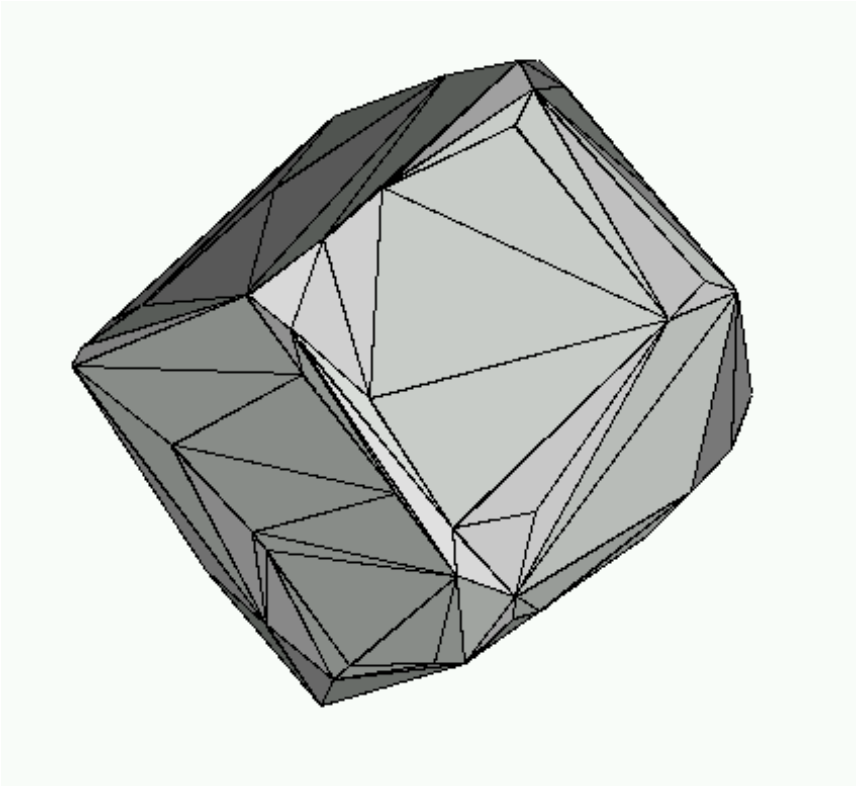
- Linear time triangulation of simple polygons, deterministic $O(n \log n)$ sweep
- simpler deterministic algs have inferior running time
- we may want to live with the inferior running time
- **randomization frequently leads to simple and efficient algorithms**

Convex Hull Problem

Input: a set of points S

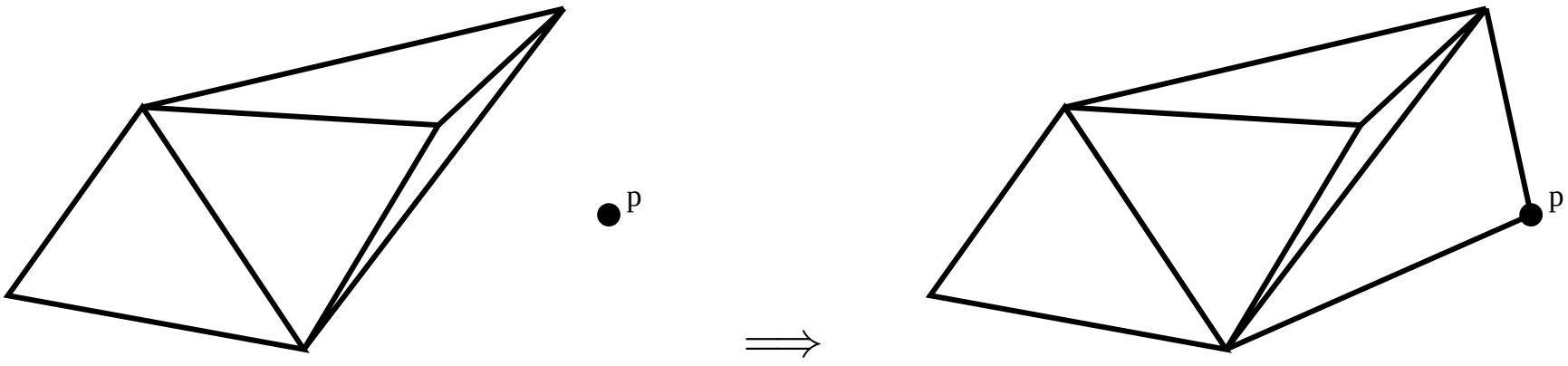
Output: a boundary description of the convex hull $CH(S)$ of S

In 3d, determine vertices, edges, and facets and their incidences



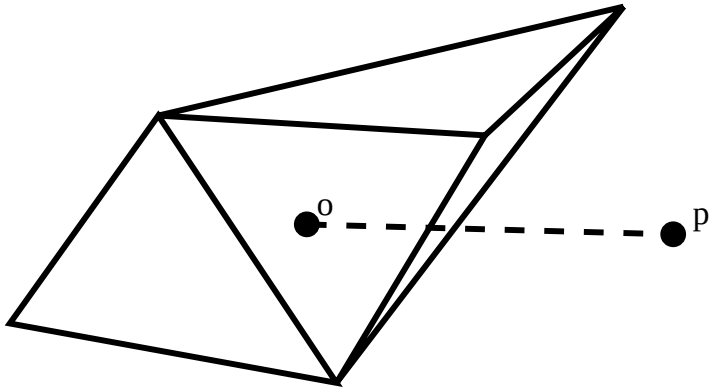
Convex Hulls by Incremental Construction (CMS, CGTA 93)

- consider the points in arbitrary order and maintain a triangulation of the current hull
- construct hull of the first point (= a point)
- in order to add a point p
  - if p is a dimension jump, make p an additional vertex of every simplex
  - otherwise, determine whether p is inside the current hull (see next slide)
  - if not, determine all facets visible from p
  - add a simplex with tip p for every visible facet



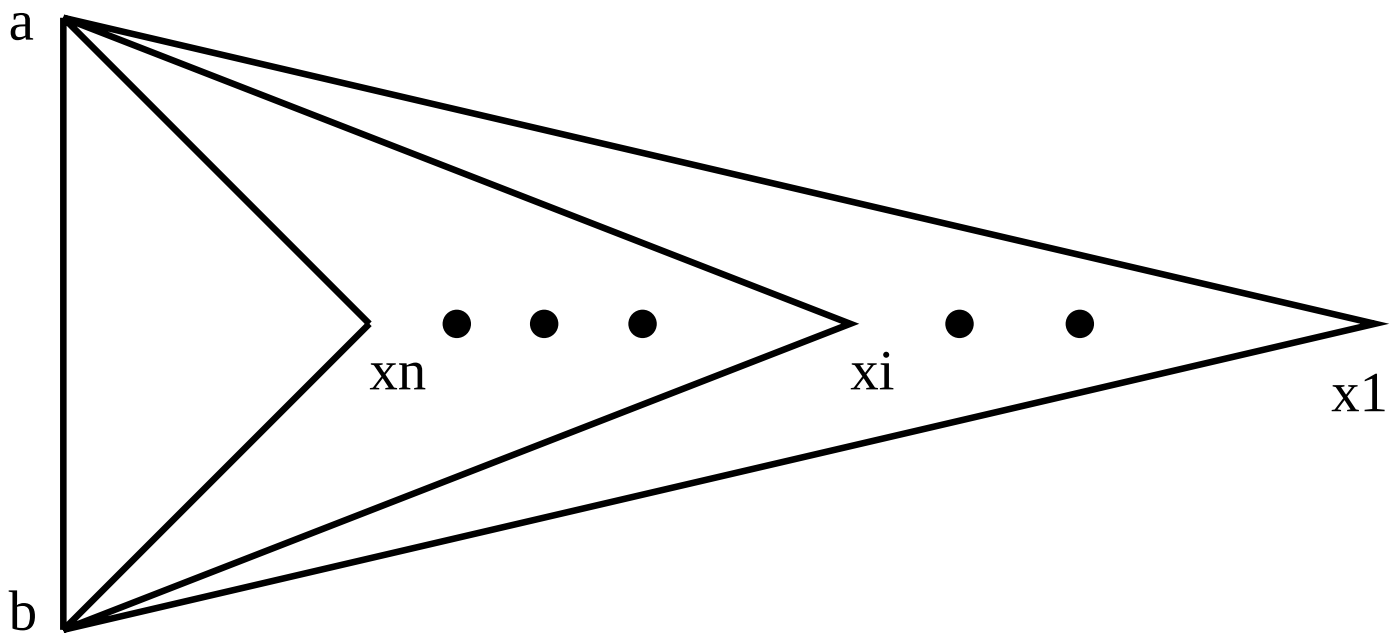
Details of the Walk

- o is a point in the interior of the first simplex
- walk through the triangulation along the segment op
- if walk hits a hull facet, p is outside the current hull
- find all visible hull facets by walking on surface of hull
- section on degeneracy discussed how to make the walk robust



Running Time in Two Dimensions

- the running time of incremental construction is $O(n^2)$, since we walk through at most i triangles when we insert the i-th point and $\sum_{1 \le i \le n} i \le n^2$
- in the worst case the running time of incremental construction is quadratic



- if the points are inserted in random order, the running time is $O(n \log n)$.

Expected Running Time of IC

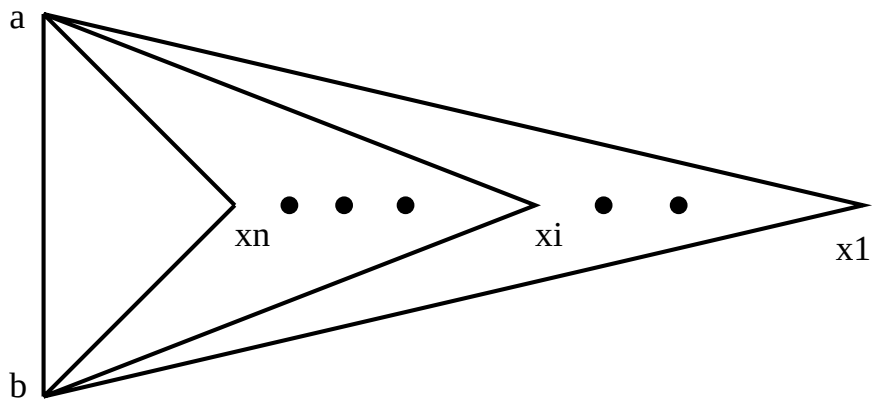
Theorem 1 *The expected running time of the incremental algorithm for convex hulls is $O(n \log n)$ if all insertion orders are equally likely.*

Assumption of the theorem is satisfied,

- when the points in S are generated according to a probability distribution for points in the plane.
- when the points in S are randomly permuted before the incremental construction process is started: *Randomized incremental construction (RIC).*

- RIC was invented by Ken Clarkson and Peter Shor in the late 80s
- very powerful paradigm
- the books by Boissonnat-Yvinec and Mulmuley center around the method
- **Remarks:** running time of RIC is $O(n)$ in the best case; this is the case when number of extreme points of S and every random subset of S is small

Analysis of Walk, A Special Case



assume: a and b are inserted first, x_1 is inserted last, and $x_2, \ldots, x_n$ are inserted in random order

- let o be a point on the segment $\overline{ab}$
- how many *constructed edges* will the segment $\overline{ox_1}$ intersect?
- edges (a, x_i) and (x_i, b) are constructed **iff** x_i is inserted before $x_1, \ldots\ x_{i-1}$
- probability for this to happen is $1/i$.
- expected time of walk for x_0 is

$$\sum_{1 \le i \le n} \frac{1}{i} = O(\log n)$$

IC versus RIC

			IC		RIC	
K	n	# extreme pts	Random	Sorted	Random	Sorted
S	4000	18	0.09	0.27	0.11	0.13
S	8000	23	0.16	0.76	0.22	0.21
S	16000	29	0.33	2.53	0.42	0.41
D	4000	59	0.1	0.45	0.11	0.1
D	8000	66	0.17	1.26	0.23	0.2
D	16000	87	0.43	3.48	0.5	0.41
C	4000	4000	0.32	15.57	0.34	0.37
C	8000	7995	0.7	65.93	0.75	0.71
C	16000	1.599e+04	1.47	253.4	1.53	1.57

S = n points in the square with side length $2R$, $R = 16000$

D = n points in the disk with radius R centered at the origin

C = n points (approximately) on the circle with radius R centered at the origin

random = points in random order

sorted = lexicographically sorted input

RIC randomly permutes input and then calls IC

Sweep versus RIC

			Sweep		RIC	
K	n	# extreme pts	Random	Sorted	Random	Sorted
S	20000	25	1.68	1.54	0.55	0.55
S	40000	29	3.6	3.26	1.26	1.43
S	80000	31	7.72	6.98	2.06	2.07
D	20000	106	1.75	1.59	0.55	0.56
D	40000	109	3.76	3.33	1.17	1.25
D	80000	152	8	7.02	2.42	2.58
C	20000	1.999e+04	1.82	1.67	2.13	2.12
C	40000	3.994e+04	3.96	3.57	4.46	4.41
C	80000	7.979e+04	8.6	7.78	10.31	10.04

columns and rows have the same meaning as on preceding slide

observe how well RIC does on inputs where the number of extreme points is small.

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
- many programs provide proof that their output is correct

  and

  checkers check proof.

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

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

Results

- theory: we developed checkers for
  - convex hulls in d-space, Voronoi diagrams, line segment intersection, ...
  - data structures, e.g., dictionaries and priority queues, ...
  - graph algorithms, e.g., planarity, flow, components, ...
- practice:
  - many of them have been implemented.
  - are part of LEDA and CGAL
  - have greatly increased reliability of the systems

Planarity Test

Input: A graph G

Question: Is G planar?

Answer: yes/no

Do you find this convincing? Run planarity demo

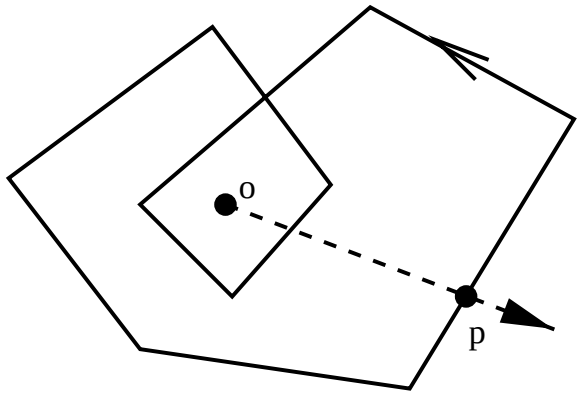
Proof: a planar drawing or a Kuratowski subgraph

Remark: Proof is readily checked.

Further Example: Run matching demo

Checking Convex Hulls

Given a simplicial, piecewise linear closed hypersurface 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 $0\vec{p}$ intersects closure of facet different from f

Discussion

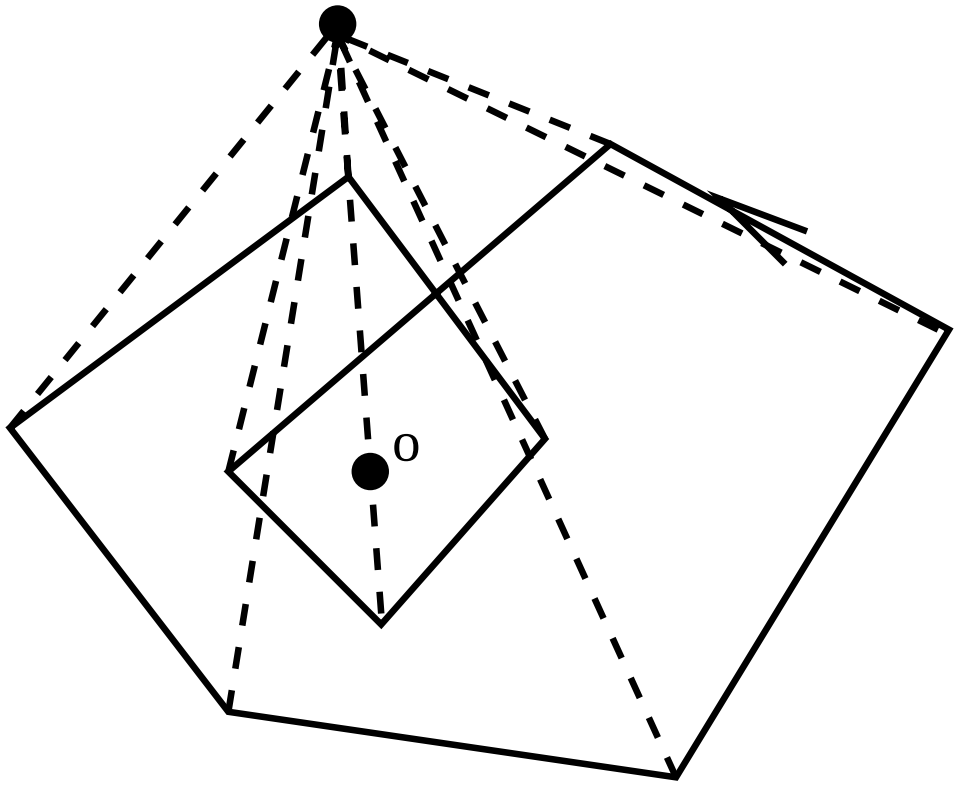
- one ray in third test
- test is fast (linear in size of F) and simple
  1. = scalar product
  2. = sign of linear function
  3. = linear system solving
     - Let f' be any facet and let h' be a supporting hyperplane.
     - $q = r \cap h'$ is a point.
     - q is a convex combination of vertices of f' and this combination is unique (since f' is a simplex). Thus, solve

$$q = \sum_{j=1}^{d} \lambda_j \cdot v_j$$

and check whether $0 \leq \lambda_j \leq 1$ for all j.

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.



Experiments

- the structure of the convex hull program
  1. incremental hull construction
  2. geometric primitives in d-space
  3. exact linear algebra over the integers
  4. arbitrary precision integers
- added checking to all but the lowest layer
- running time increases by $\leq 10\%$

Conclusions

- Checking allows one to test on **all** inputs and not only on inputs for which the output is known.
- checked programs are reliable
  - either give the correct answer
  - or catch the error themselves
- surprisingly many programs can be checked
- designing checkers is non-trivial
  - convex hull
  - priority queue
- KM usually designs the checker first