Geometric Computing

The Science of Making Geometric Algorithms Work

Kurt Mehlhorn

Max-Planck-Institut für Informatik and Saarland University

Overview

- The Real-RAM Model and the Innocent Years (till 85)
- Sobering Experiences (85 – 90)
- Floating Point Arithmetic and Geometry
- Numerical Analysis versus Computational Geometry
- Approaches to Reliable Geometric Computing (90 – today)
    - Make Floating Point Arithmetic Work
    - Exact Computation Paradigm
- Exact Computation Paradigm: State of the Art
    - Examples of what can be done
    - A powerful theme: exact decisions based on approximate computation
- CGAL, the Computational Geometry Algorithms Library
- Outlook

The Real RAM

The computation model we shall refer to is that of a random access machine (RAM) in the sense of Aho, Hopcroft, and Ullman, with the only modification that real number arithmetic replaces integer arithmetic.

F. Preparata and S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, CACM, 20, 2, (Feb 77), p.88:

Remarks

It is the natural model for CG, as the parameters (point coordinates, line coefficients, radii of circles, . . .) are real values.

Numerical analysis used it successfully; so it seemed like an innocent assumption.

The Real-RAM is key to the success of the field

Remarks

One can buy RAMs, one cannot buy a Real-RAM.

Computers offer bounded length integer and floating point arithmetic in hardware and arbitrary precision arithmetic in software.

Remarks

The research community developed and analyzed algorithms, it was not interested in implementation.

It was common belief that floating point arithmetic would be a good enough substitute for real arithmetic, as it is in numerical analysis.

Sobering Experiences (85 – 90)

- In the mid 80s, the CG community and others began to implement geometric algorithms (line segment intersection, Delaunay and Voronoi diagrams, arrangements of lines and hyperplanes).

Real RAM = RAM + floating point arithmetic

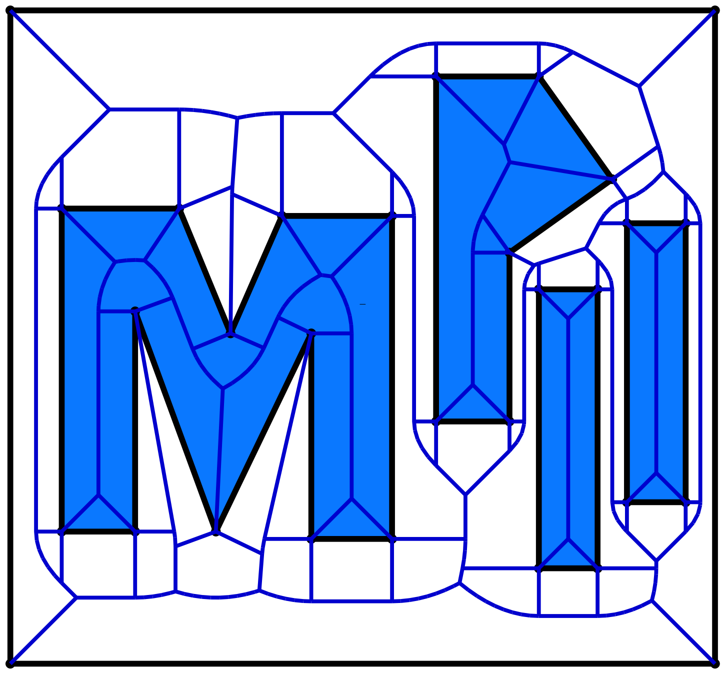
Sobering Experiences (85 – 90)

- In the mid 80s, the CG community and others began to implement geometric algorithms (line segment intersection, Delaunay and Voronoi diagrams, arrangements of lines and hyperplanes).

Real RAM = RAM + floating point arithmetic

- Floating point arithmetic is a poor substitute for real arithmetic. Most implementations did not work.
- Degenerate cases were cumbersome.
- Debugging was a nightmare.
- I had asked a student to implement Voronoi diagrams of line segments:

  we found a few examples on which the implementation worked.



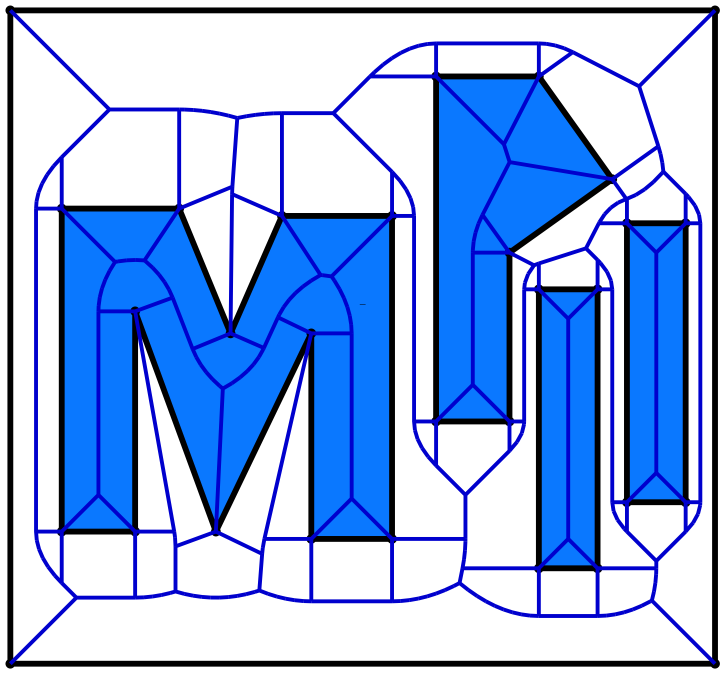
Sobering Experiences (85 – 90)

- In the mid 80s, the CG community and others began to implement geometric algorithms (line segment intersection, Delaunay and Voronoi diagrams, arrangements of lines and hyperplanes).

Real RAM = RAM + floating point arithmetic

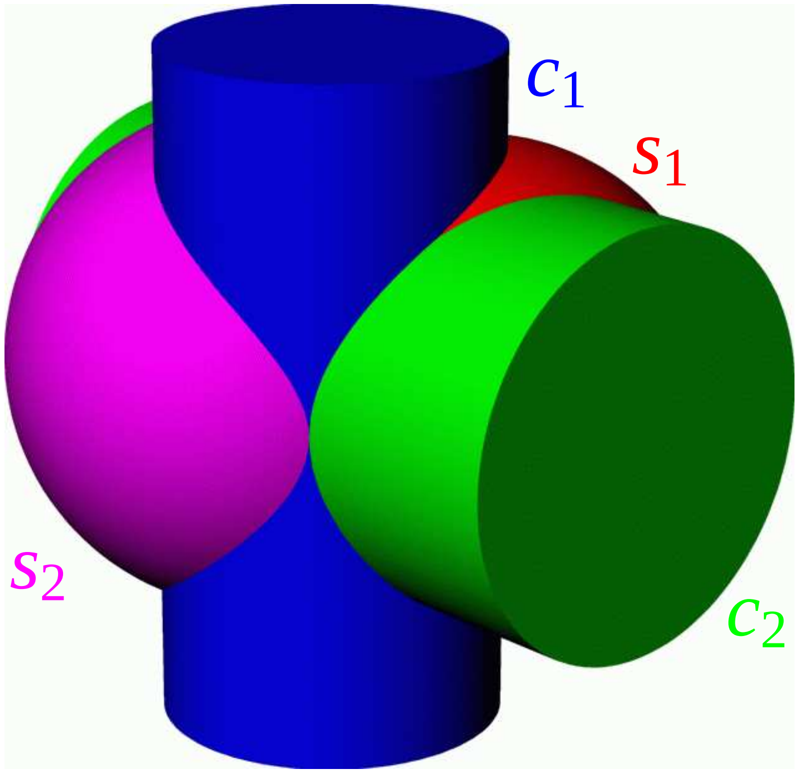
- Floating point arithmetic is a poor substitute for real arithmetic. Most implementations did not work.
- Degenerate cases were cumbersome.
- Debugging was a nightmare.
- I had asked a student to implement Voronoi diagrams of line segments:

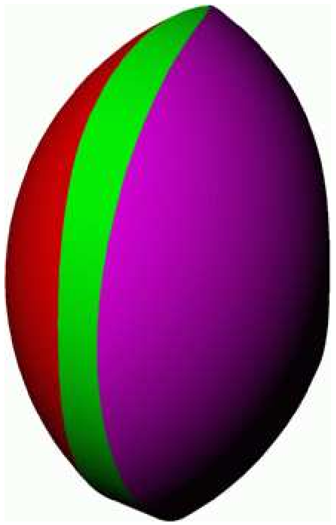
  we found a few examples on which the implementation worked.



- Starting in the late 80s, implementation of geometric algorithm became an area of research (Fortune, Guibas, Milenkovic, Stolfi, Sugihara, Yap)

The Intersection of Four Simple Solids





output is not a set, but a combinatorial object plus coordinates

Rhino3D: $(((s_1 \cap s_2) \cap c_2) \cap c_1) \quad \longrightarrow$ successful

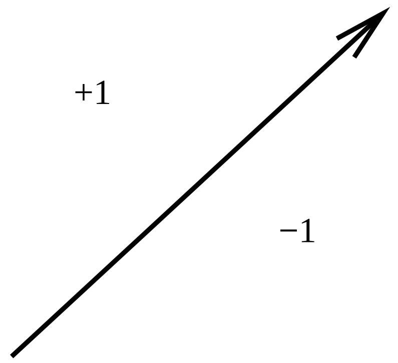
$(((c_1 \cap c_2) \cap s_1) \cap s_2) \quad \longrightarrow$ “**Boolean operation failed**”

geometric problems are non-continuous functions from input to output

Geometry and Floating Point Arithmetic

- Orientation Predicate: three points p, q, and r in the plane either lie on a common line or form a left or right turn

  $$orient(p,q,r) = 0,\ +1,\ -1$$



- analytically

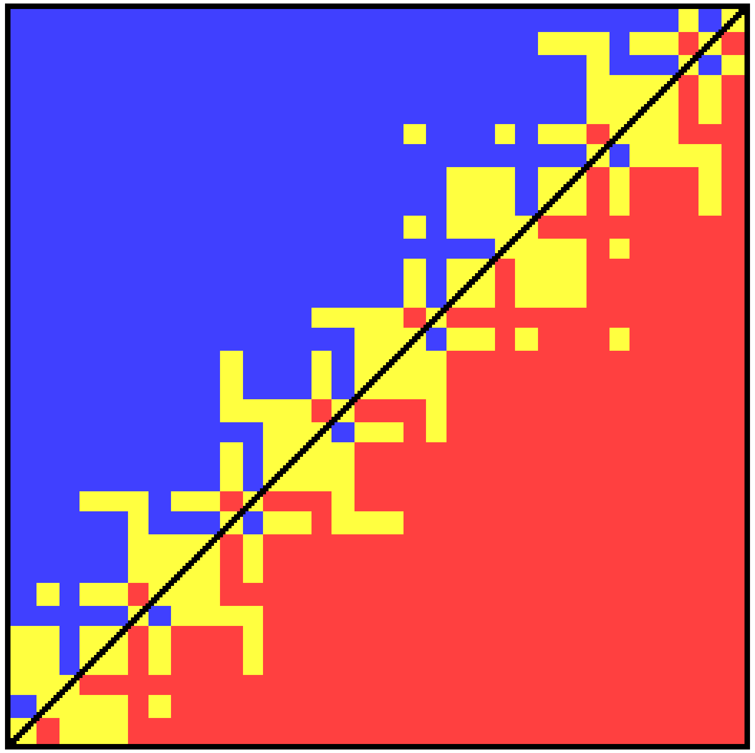
$$orient(p,q,r) = sign(\det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix})$$

- $float\_orient(p,q,r)$ is the result of evaluating $orient(p,q,r)$ in floating point arithmetic.

- Observe: we formulate our algorithms in terms of geometric predicates, we use arithmetic to implement these predicates.

Geometry of Float-Orient

$p = (0.5, 0.5)$, $q = (12, 12)$ and $r = (24, 24)$



0.5 $\qquad$ $0.5 + 255 \cdot 2^{-53}$

picture shows

$$float\_orient((p_x + xu, p_y + yu), q, r)$$

for $0 \le x, y \le 255$, where $u = 2^{-53}$.

the line $\ell(q, r)$ is shown in black

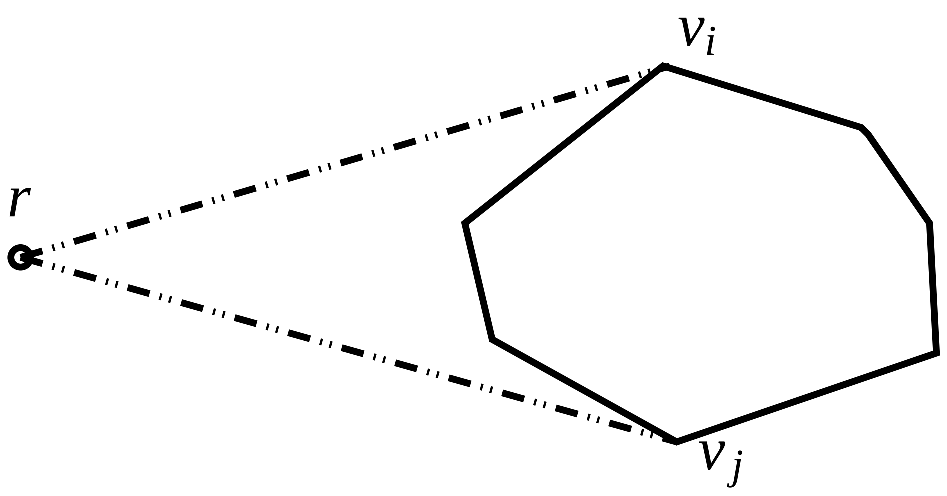
near the line many points are mis-classified

Kettner/Mehlhorn/Pion/Schirra/Yap: ESA 2004, CGTA 2008

A Simple Convex Hull Algorithm

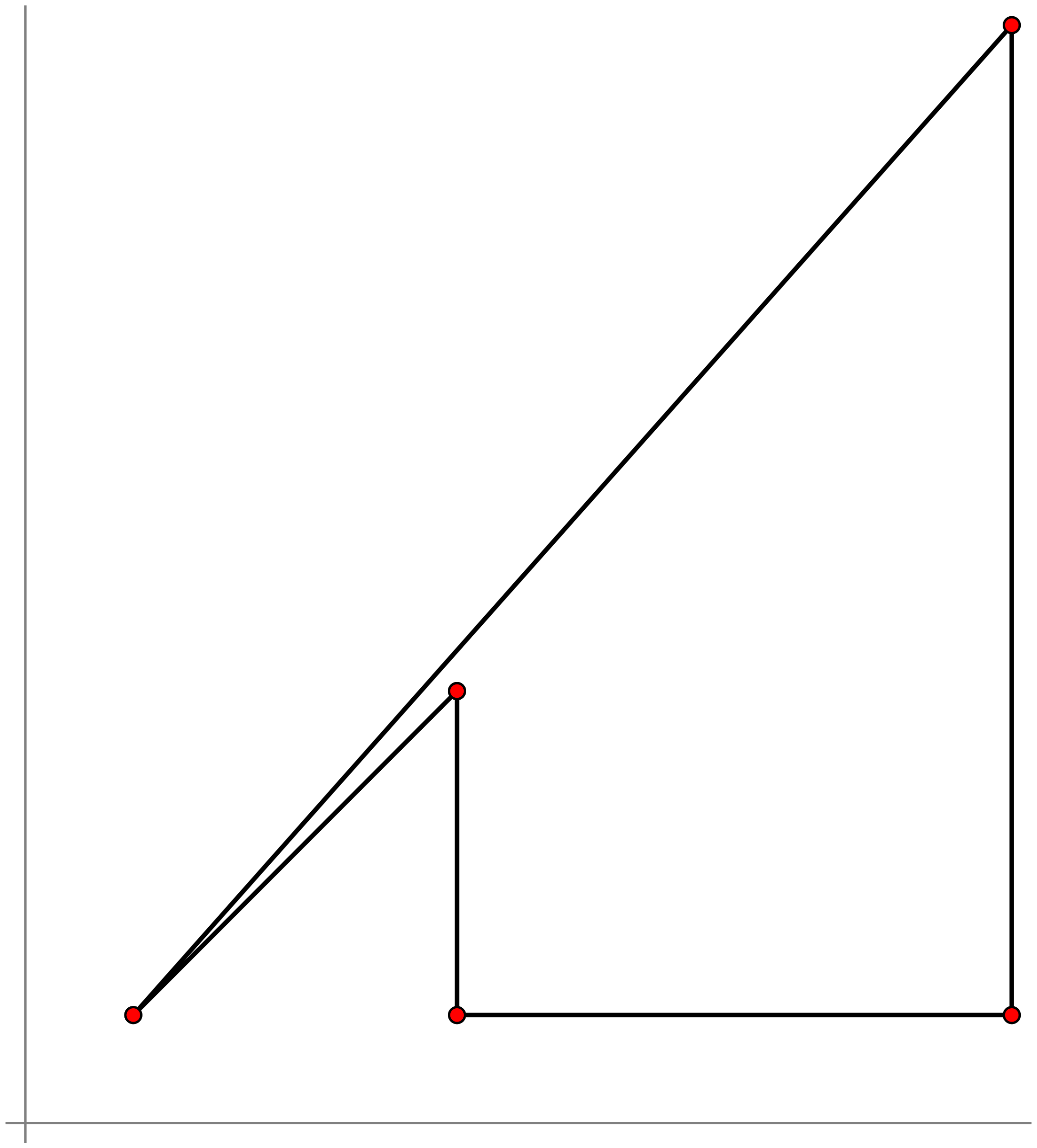
- alg considers the points one by one, maintains vertices of current hull in counter-clockwise order

```
Initialize hull with first three points.
for all $r \in S$ do
  if there is an edge $e$ visible from $r$ then
    compute the sequence $(v_i, \ldots, v_j)$ of edges visible from $r$
    and replace the subsequence $(v_{i+1}, \ldots, v_{j-1})$ by $r$.
```

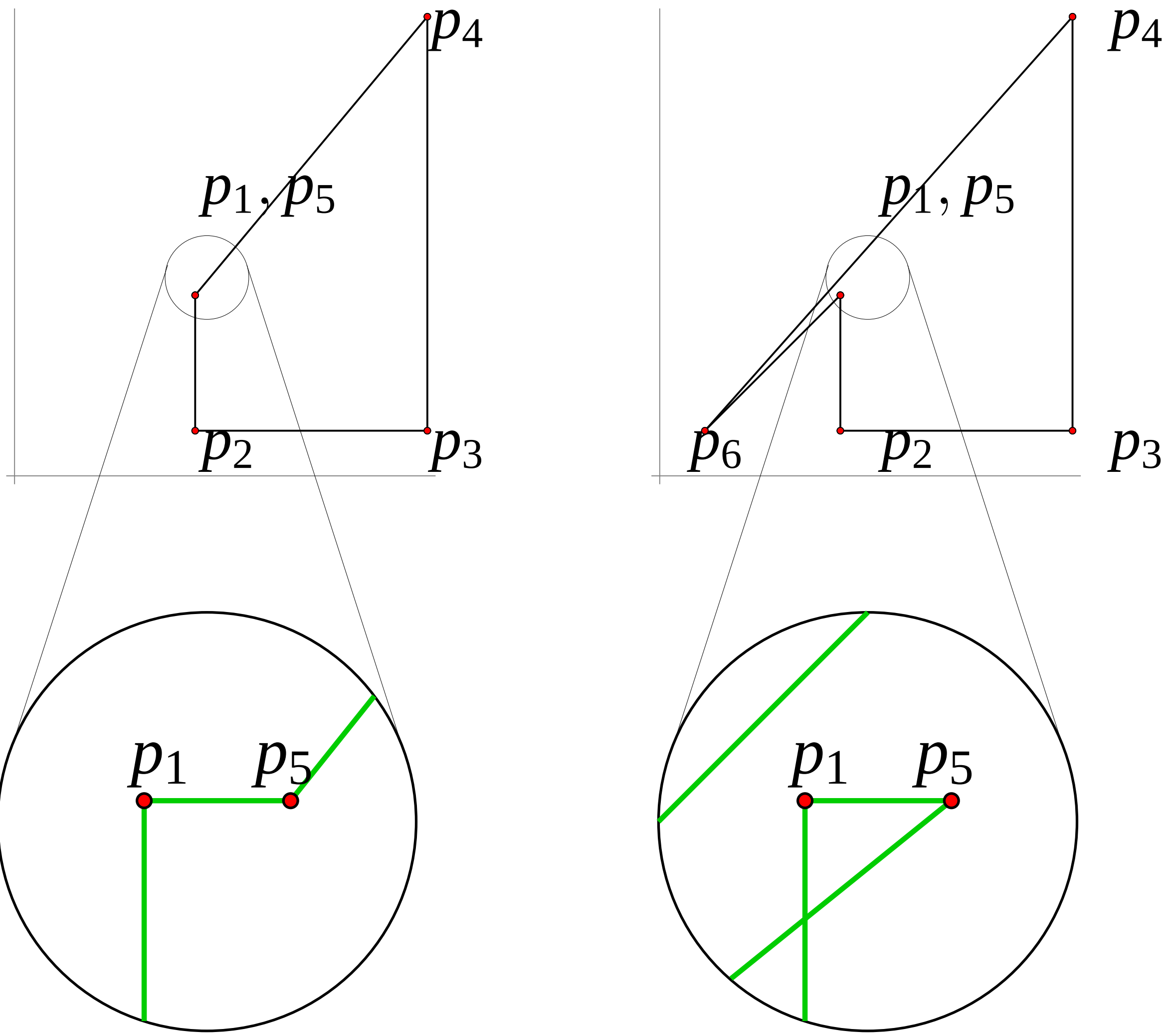


of course: the visible edges form a contiguous subsequence

An Output of this Algorithm



Small Errors, Large Effects



- the hull of p_1 to p_4 is correctly computed
- p_5 lies close to p_1, lies inside the hull of the first four points, but float-sees the edge (p_1, p_4).
  Concave corner at p_5.
- point p_6 sees the edges (p_1, p_2) and (p_4, p_5), but does **not** see the edge (p_5, p_1).
- we obtain ...

Numerical Analysis versus Computational Geometry

- Numerical analysis mainly computes continuous functions, e.g., the eigenvalues of a matrix.
- Algorithms of numerical analysis are self-correcting.
- We are to a large extent interested in non-continuous functions, e.g., the convex hull of a set of points.

  The output is a combinatorial object (sequence of vertices) not a set.
- Numerical analysis calls such problems ill-posed or at least ill-conditioned.
- We use arithmetic to make yes/no decisions, e.g.,

  does p lie on ℓ or not?
- Our algorithms are not self-correcting.

Approaches to Reliable Geometric Computing

- Reliable = program does what it claims to do.
- Make Floating Point Arithmetic Work
- The Exact Geometric Computation Paradigm (ECG) Yap, 93

Approaches to Reliable Geometric Computing

- Reliable = program does what it claims to do.
- Make Floating Point Arithmetic Work
  - redefine the problem: approximation instead of true output
  - modify algorithms to cope with imprecise arithmetic
  - restrict inputs, e.g., integer coordinates in $[-2^{20}..2^{20}]$.
  - solutions are algorithm specific, not generic
  - Milenkovic, Sugihara, Shewchuck, Halperin, Mehlhorn, Osbild, Sagraloff
- The Exact Geometric Computation Paradigm (ECG) Yap, 93

Approaches to Reliable Geometric Computing

MAX-PLANCK-GESELLSCHAFT

- Reliable = program does what it claims to do.
- Make Floating Point Arithmetic Work
  - redefine the problem: approximation instead of true output
  - modify algorithms to cope with imprecise arithmetic
  - restrict inputs, e.g., integer coordinates in $[-2^{20}..2^{20}]$.
  - solutions are algorithm specific, not generic
  - Milenkovic, Sugihara, Shewchuck, Halperin, Mehlhorn, Osbild, Sagraloff
- The Exact Geometric Computation Paradigm (ECG) Yap, 93
  - implement a Real-RAM to the extent needed
    the challenge is efficiency and keeping extent low
  - redesign the algorithms so that they can handle all inputs and have small arithmetic demand
  - ECG applies to all geometric algorithms
  - basis for LEDA and CGAL

The ECG Paradigm: State of the Art

- Implement a Real-RAM to the extend needed.
- For linear geometric objects, efficiency and reliability can be achieved simultaneously (1992 – 2002)
  - This required to develop some theory and
  - serious systems building (LEDA, CGAL)
- Working Hypothesis: to a large extent, this also holds true for nonlinear geometric objects (2002 –
  - This requires lots of new theory
  - and serious systems building (CGAL)

The ECG Paradigm: State of the Art

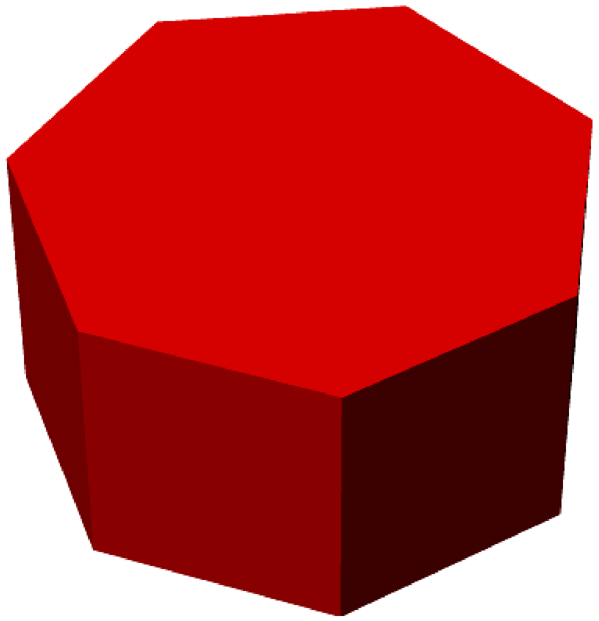
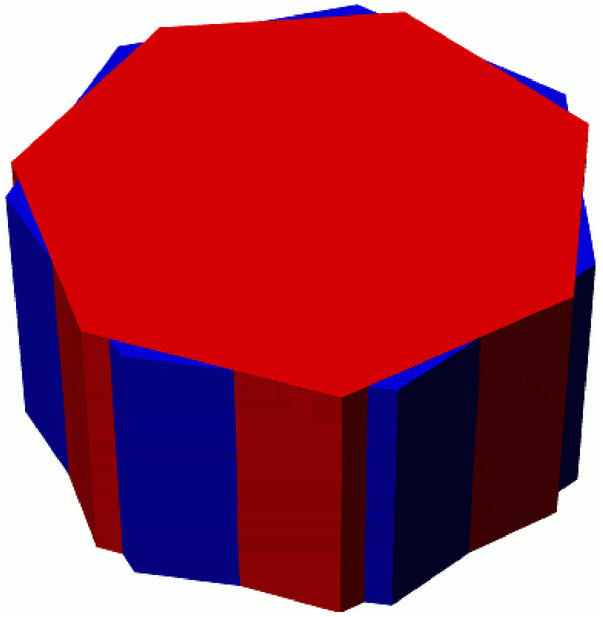
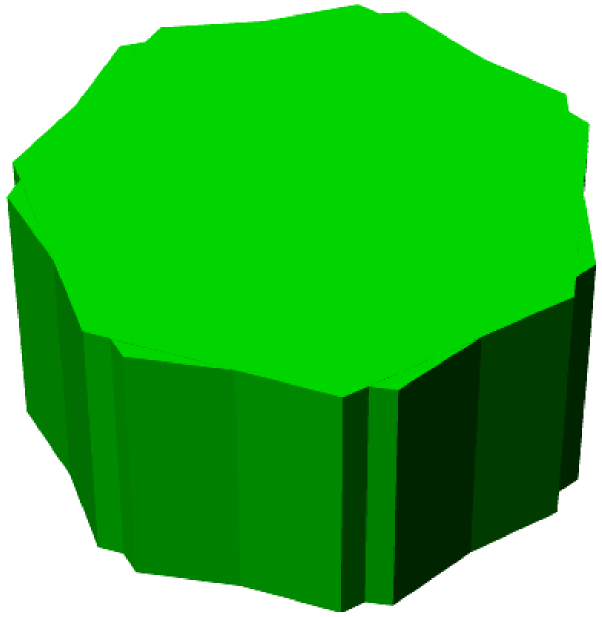
- Implement a Real-RAM to the extend needed.

- For linear geometric objects, efficiency and reliability can be achieved simultaneously (1992 – 2002)
  - This required to develop some theory and
  - serious systems building (LEDA, CGAL)

- Working Hypothesis: to a large extent, this also holds true for nonlinear geometric objects (2002 –
  - This requires lots of new theory
    - new geometric algorithms with smaller arithmetic demand,
    - advances in computer algebra, in particular polynomial system solving, and
    - better interplay between symbolic, numeric and geometric computing
  - and serious systems building (CGAL)

Intersection of Two Polygonal Cylinders

- construct cylinder P, base is regular n-gon,
- obtain Q from P by rotation by α degrees,
- and compute the **union** of P and Q

Intersection of Two Polygonal Cylinders

- construct cylinder P, base is regular n-gon,
- obtain Q from P by rotation by α degrees,
- and compute the **union** of P and Q

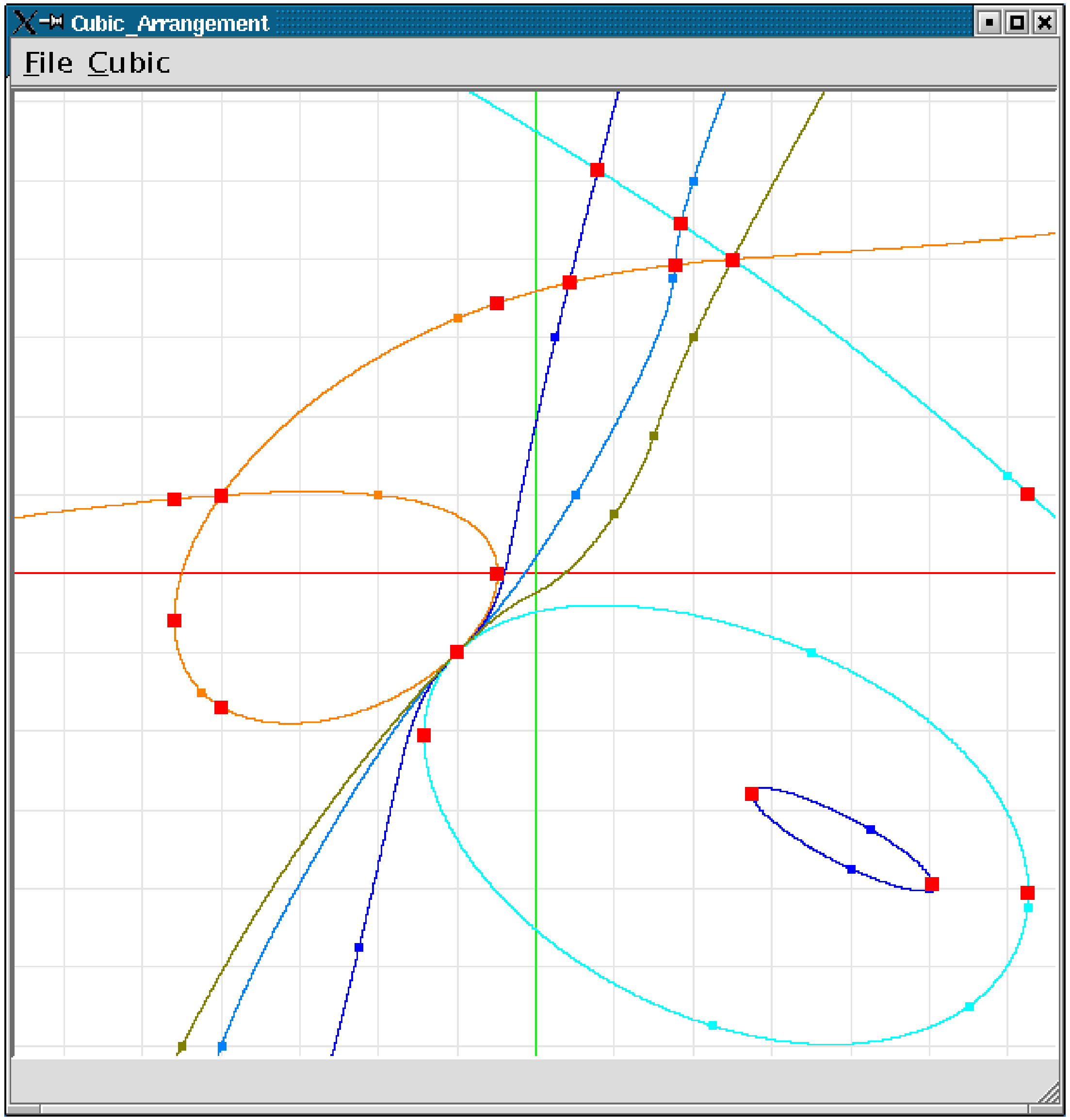
system	n	α	time	output
ACIS	1000	1.0e-4	10 sec	correct
ACIS	1000	1.0e-6	2sec	incorrect answer
Rhino3D	200	1.0e-2	15sec	correct
Rhino3D	400	1.0e-2	–	CRASH
CGAL/LEDA	5000	6.175e-06	30 sec	correct
CGAL/LEDA	5000	1.581e-09	34 sec	correct
CGAL/LEDA	$\longrightarrow \infty$	$\longrightarrow 0$	$\longrightarrow \infty$	correct

Hachenberger/M/Kettner

Some Results for Nonlinear Geometry

Arno Eigenwillig	Erik Berberich	Efi Fogel (Tel Aviv)
Dany Halperin (Tel Aviv)	Michael Hemmer	Michael Kerber
Lutz Kettner (Mental Images)	Kurt Mehlhorn	Michael Sagraloff
Elmar Schömer (Mainz)	Raimund Seidel	Vikram Sharma
Stefan Schirra (Magdeburg)	Ron Wein (Tel Aviv)	Nicola Wolpert (Stuttgart)
Rudolf Fleischer (Shanghai)		

A Basic Problem: Arrangement Computation



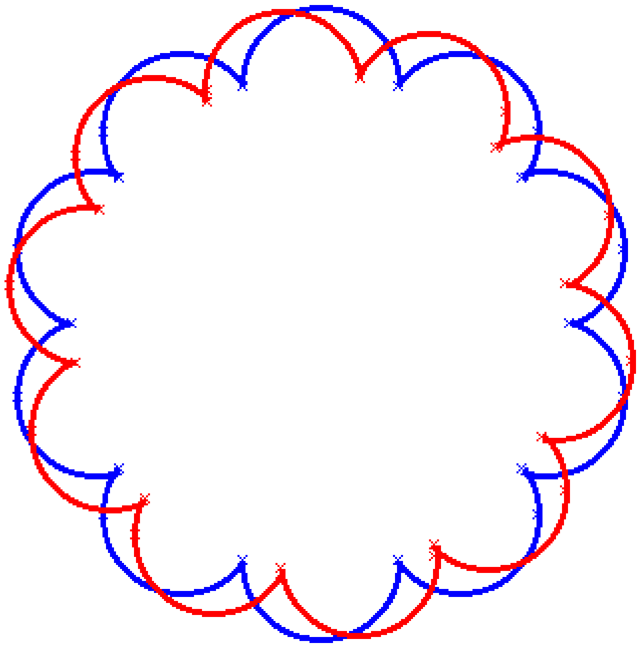
you see a planar embedded graph; the challenge is to compute the graph

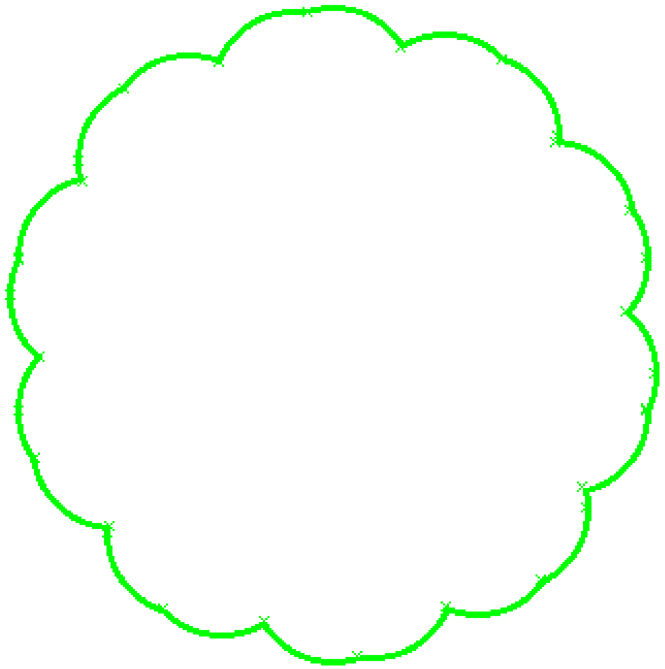
algorithms handle arbitrary degrees and degeneracies

- many curves have a common point
- different slopes
- same slope, different curvature,
- same slope and curvature, diff . . .

contributors: all

Boolean Operation on Curved Polygons





green polygon is union of red and blue

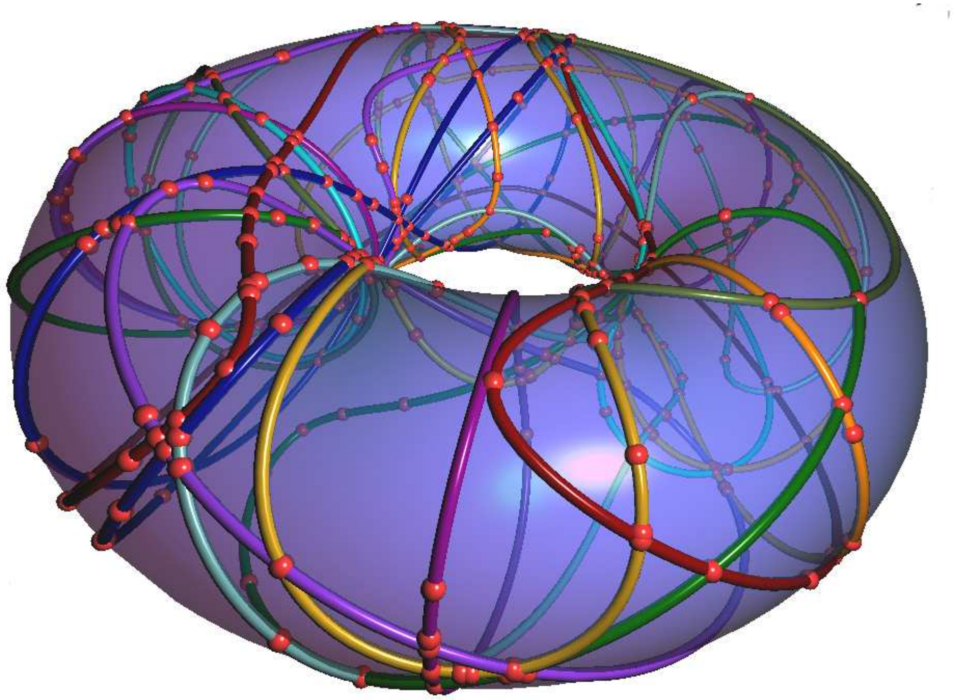
computation takes about 10 secs for polygons with 1000 vertices

this is competitive with unreliable approaches

arrangement computation is the workhorse

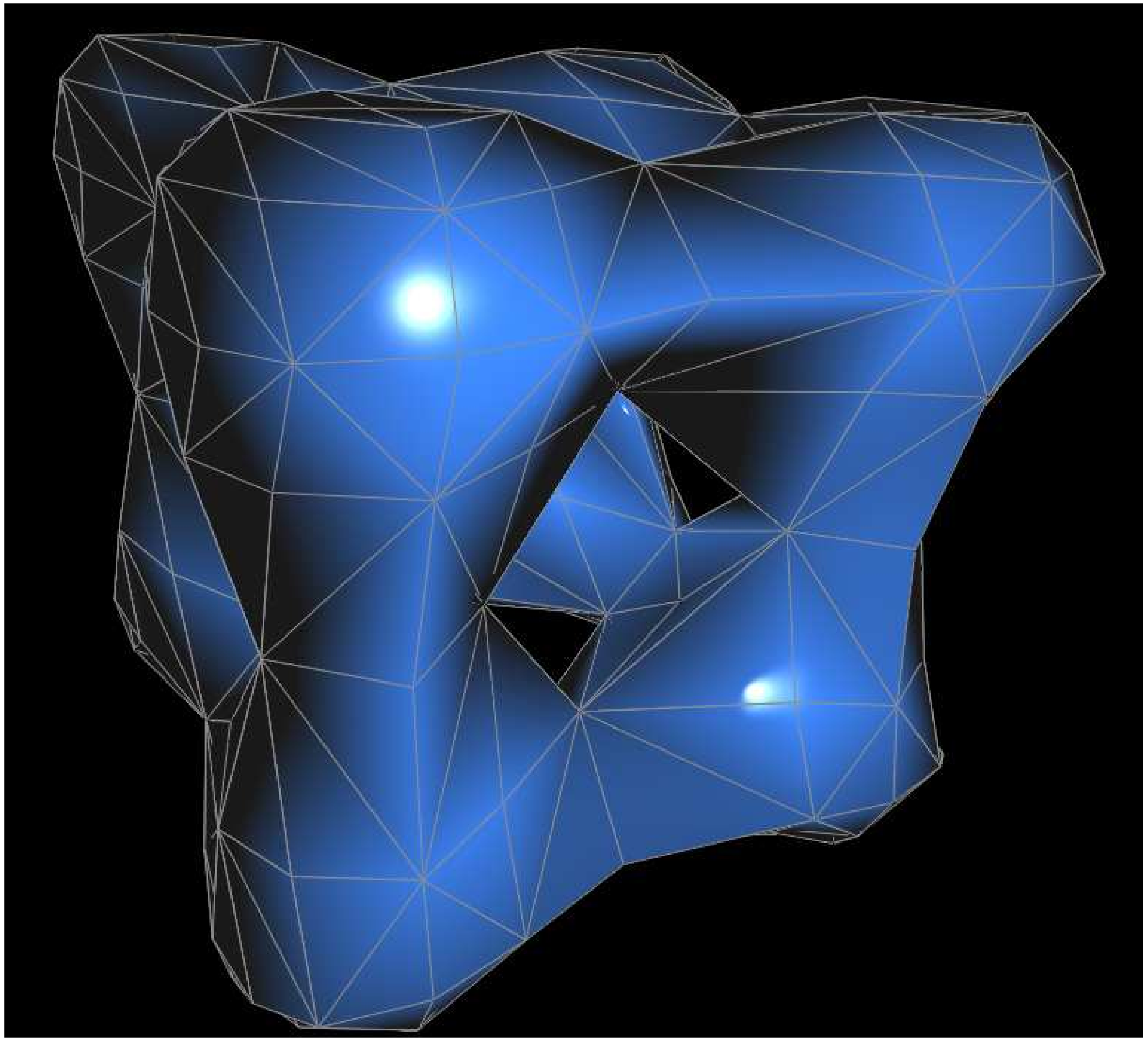
contributors: all

Arrangements on Surfaces



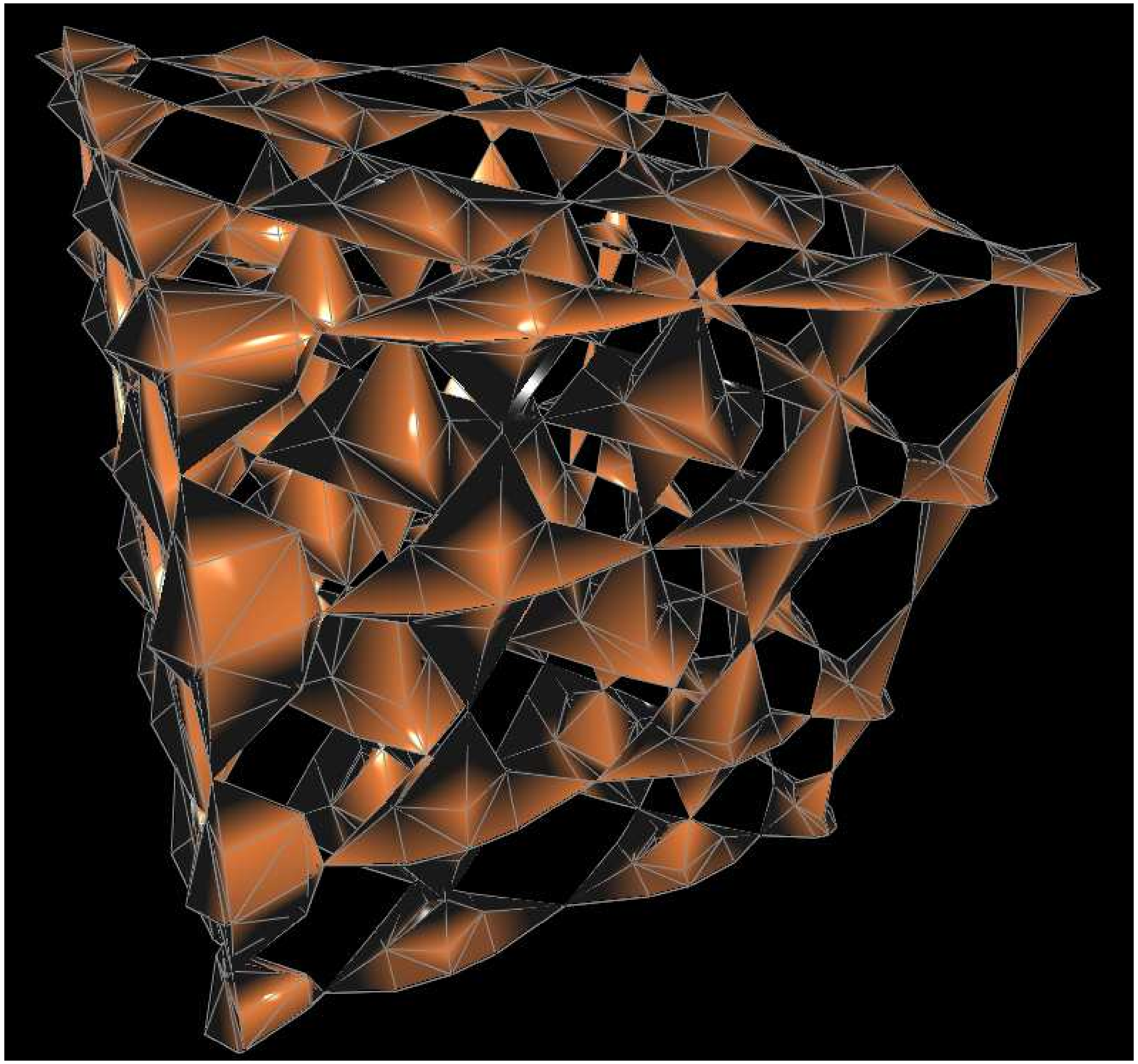
- a step towards 3d
- arrangement on a torus-like surface defined by quadric surfaces
- method (and software) works for surfaces such as spheres, cylinders, tori, paraboloids, . . . ,
- software is generic
- applications: point location, overlay, Minkowski sums, 3d boolean ops (???, Dupont, Hemmer, Petitjean, Schömer)
- Berberich, Fogel, Halperin, Hemmer, Mehlhorn, Schömer, Wein, Wolpert

Analysis of a Single Algebraic Surface



- a second step towards 3d
- topology of an algebraic surface
  - nodes = singularities
  - edges = self-intersections
  - faces = surface-patches
- triangulation of algebraic surfaces
- symbolic $\Rightarrow$ verified approximate

Analysis of a Single Algebraic Surface



- a second step towards 3d
- topology of an algebraic surface
- triangulation of algebraic surfaces

- Berberich, Kerber, Sagraloff

A General Theme

MAX-PLANCK-GESELLSCHAFT

compute numerically

and make sure that the results can be interpreted symbolically

Approach has increased the efficiency of our algorithms by three orders of magnitude

Exact Sign Computation of Radical 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

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 the 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 numerical approach)
- strategy above is basis for sign test in LEDA and CORE *reals*.

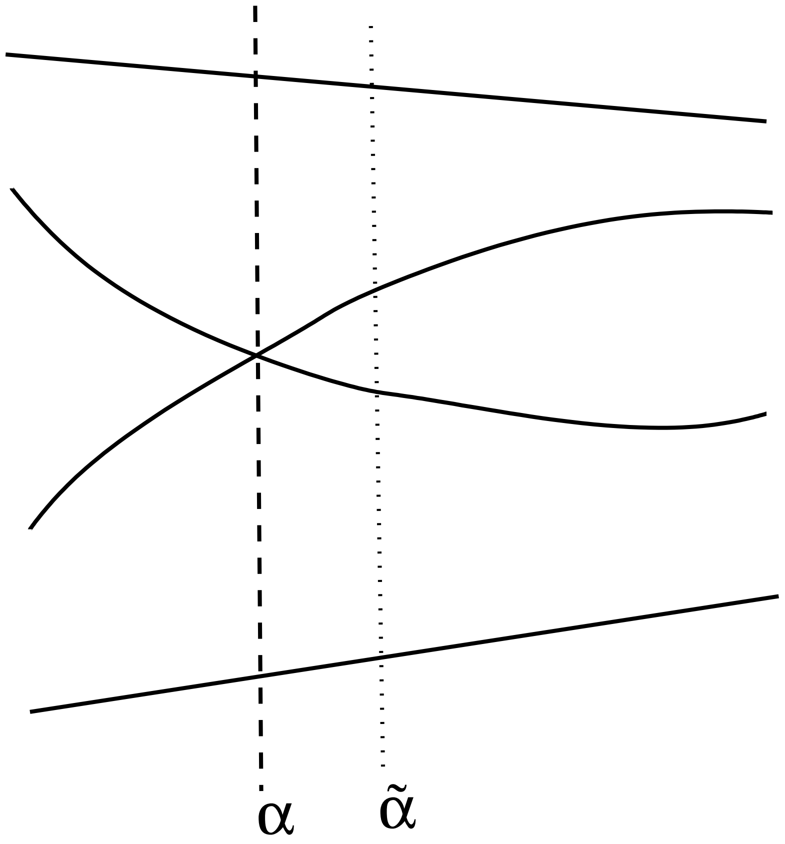
Root Isolation for Bitstream Polynomials

MAX-PLANCK-GESELLSCHAFT

- polynomial $p(x) = \sum_{0 \le i \le n} a_i x^i$.
- the a_i are arbitrary reals, can compute approximations, e.g., $\pi = 3.14\ldots$
- to isolate a real root z of p means to compute an interval $[a,b]$ with rational endpoints containing z and no other root of p
- same running time on $x^3 - 3$, on $\sqrt{2}x^3 - 3\sqrt{2}$, and on $\pi x^3 - 3\pi$
- all previous algorithms exhibited large differences
- isolate roots of $\tilde{p}(x) = \sum_{0 \le i \le n} \tilde{a}_i x^i$, where $\tilde{a}_i \approx a_i$

  issues: same number of real roots, intervals for $\tilde{p}$ apply to p, running time determined by geometry of zeroset of p
- new root isolator is key for our implementations
- symbolic $\Rightarrow$ verified approximate

Interplay: Symbolic – Verified Approximate



- given an algebraic curve $p(x,y)$, can determine a polynomial $R(x)$ whose roots are the x-coordinates of the critical points, here α
- we isolate α and obtain an approximation $\tilde{\alpha}$
- want zeros of $p(\alpha,y)$
- know from a symbolic computation that $p(\alpha,y)$ has three distinct real roots one of which is a double root
- determine roots of $p(\tilde{\alpha},y)$
- can infer situation at α from situation at $\tilde{\alpha}$
- how good does $\tilde{\alpha}$ have to be?
- Eigenwillig, Hemmer, Kerber, Seidel, Wolpert

CGAL

- CGAL = computational geometry algorithms library
- The goal of the CGAL Open Source Project is to provide easy access to efficient and reliable geometric algorithms in the form of a C++ library
- Why a library?
  - Build on the shoulders of others
  - Preserve good solutions
- Why CGAL?
  - CGAL offers many algorithms.
  - CGAL is generic, i.e., modules are easily reused, e.g.,

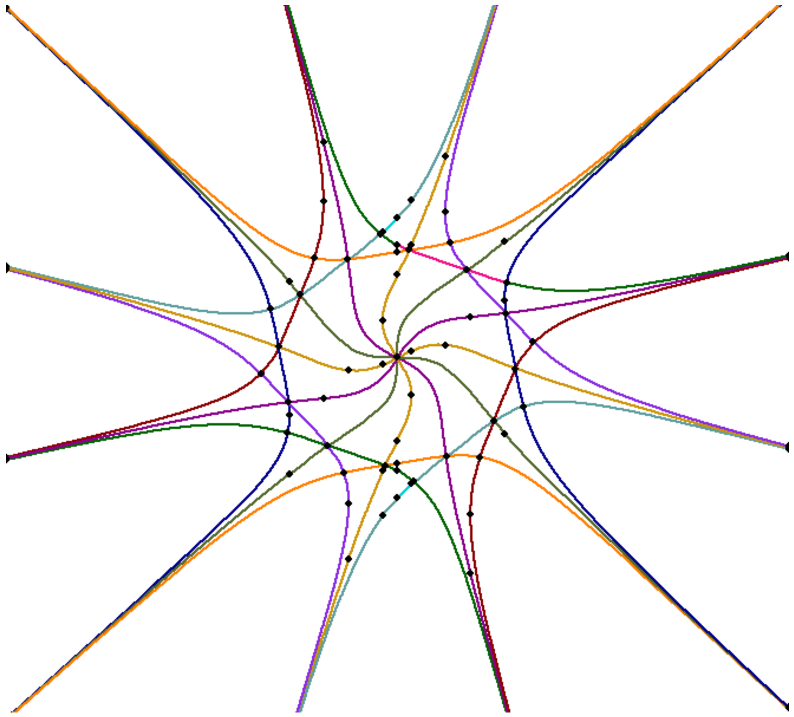
    gcd for polynomials over any ring and not just for integer polynomials

Genericity and Functionality

- arrangements, initially, for curves with integral coefficients

$$7x^3 + 4xy^2 + \dots$$

- rotation by α degrees:
  $x' = x\cos\alpha + y\sin\alpha,\ y' = \dots$
- $\alpha = 30$ degree, $\cos\alpha = \sqrt{3}/2$, $\sin\alpha = 1/2$
- new number type: $\mathbb{Z}[\sqrt{k}]$ for integer k
- now algorithms can also handle rotated curves (as long as sine and cosine can be expressed as roots)
- Hemmer, Kerber



Outlook

- symbolic $\Rightarrow$ verified approximate
  - improved running time without sacrificing correctness
  - but symbolic computations are still the bottleneck
- Bezier curves and surfaces
- 3D arrangements of surfaces,
- Boolean operations, offsets, . . . of 2d polygons whose edges are algebraic curve or Bezier curve segments
- Boolean operations, offsets, . . . of 3d polyhedra whose faces are algebraic surface or Bezier surface patches
- Rounding and Cascaded Constructions
- LEDA/CGAL have a fairly large academic and commercial user base
- achieve the same for nonlinear CGAL