Reliable and Efficient Geometric Computation

Kurt Mehlhorn
Max-Planck-Institut für Informatik

slides and papers are available at my home page
My Waterloo Co-Authors

Cheriyan: Maximum Flow (SICOMP 96), Algs for Dense Graphs (Algorithmica 96), Highest-Level Selection (IPL 99)

Koenemann: Exact Geometric Computation in LEDA (CompGeo 96)

Munro: Partial Match Retrieval (IPL 84), Random Variates (ICALP 93), Multiple Selection (ICALP 05)
Geometric Computing
The Goal: Reliable and Efficient Geometric Computing

in particular, a reliable and efficient CAD kernel

reliable = produce a sensible output for all inputs

sensible output =
  • the mathematically correct output or
  • something provably close to the correct output

efficient = at most ten times slower than existing unreliable implementations

Why am I interested?
  • mathematically challenging
  • industrially relevant
  • I blundered once: the first release of geometry in LEDA was unreliable
State of the Art

Most existing implementations (commercial or academic) are unreliable

- may crash or produce non-sensical answers

Where do we stand?

- we = reliable geometric algorithms project at MPI + EU-projects CGAL, GALIA, ECG and ACS
- linear (lines, planes, points) geometry in 2d and 3d: nice academic work + first industrial impact
- curved geometry in 2d: nice academic work + first industrial impact
- curved geometry in 3d: nice academic work
- implementations available in LEDA, CGAL, and EXACUS (ESA 2005)

How do we work?

- develop the required theory and system architecture and build prototypical systems to validate the theory and to have impact beyond our own community
Examples I: Intersection of 3d-Solids

Rhino3d crashes on this input  the correct output
Examples II: Intersection of Planar 3d-Solids

Task: construct a regular cylinder $P$ (base = regular $n$-gon) obtain $Q$ from $P$ by a rotation by $\alpha$ degrees about its center, and compute the union of $P$ and $Q$

<table>
<thead>
<tr>
<th>System</th>
<th>$n$</th>
<th>$\alpha$</th>
<th>time</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACIS</td>
<td>1000</td>
<td>1.0e-4</td>
<td>30 sec</td>
<td>correct</td>
</tr>
<tr>
<td>ACIS</td>
<td>1000</td>
<td>1.0e-6</td>
<td>30 sec</td>
<td>incorrect</td>
</tr>
<tr>
<td>CGAL/LEDA</td>
<td>1000</td>
<td>1.0e-6</td>
<td>44 sec</td>
<td>correct</td>
</tr>
<tr>
<td>CGAL/LEDA</td>
<td>2000</td>
<td>1.0e-7</td>
<td>900sec</td>
<td>correct</td>
</tr>
</tbody>
</table>

Granados/Hachenberger/Hert/Kettner/Mehlhorn/Seel: ESA 2003

Hachenberger/Kettner: ESA 2005
Example III: Curved Polygons

- the green polygon is the union of the red and the blue polygon
- edges are half-circles (more generally, conic arcs)
- computation takes about 30 seconds for polygons with 1000 edges
- requires extension of sweep line algorithm and exact computation with algebraic numbers of degree at most four

Berberich/Eigenwillig/Hemmer/Hert/Mehlhorn/Schömer: ESA 2002
Example IV: Degeneracies

A highly degenerate example:

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

algorithm computes a planar map and not only a picture

Berberich/Eigenwillig/Hemmer/Schömer/Wolpert
CompGeo 2004
What is difficult?

- algs are designed for the real-RAM and non-degenerate inputs
  - real-RAM = machine computes with real numbers in the sense of mathematics: exact roots of polynomials, sine, cosine, . . .
  - non-degenerate inputs: no three points on a line, no three curves through a point, . . .

- but real inputs are frequently degenerate and

- real computers are not real-RAMs (32 bit integer and double precision floating point arithmetic)

- the next three slides illustrate the pitfalls of floating point computation
The Orientation Predicate

three points $p$, $q$, and $r$ in the plane either lie
• on a common line or form a left or right turn
$\text{orient}(p,q,r) = 0,$ $+1,$ $-1$

• analytically

\[
\text{orient}(p,q,r) = \text{sign} \left( \det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix} \right) \\
= \text{sign} \left( (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x) \right).
\]

• $\det$ is twice the signed area of the triangle $(p,q,r)$

• $\text{float}_\text{orient}(p,q,r)$ is result of evaluating $\text{orient}(p,q,r)$ in floating point arithmetic
Geometry of Float-Orient

\[ p = (0.5, 0.5), \quad q = (12, 12) \text{ and } r = (24, 24) \]

picture shows

\[
\text{float}_\text{orient}((p_x + xu, p_y + yu), q, r)
\]

for \(0 \leq x, y \leq 255\), 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
A Simple Convex Hull Algorithm

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

- Initialize $L$ to the counter-clockwise triangle $(a, b, c)$.

  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 $r$.
      replace the subsequence $(v_{i+1}, \ldots, v_{j-1})$ by $r$.
    end if
  end for

-
The Effect on a Simple Convex Hull Algorithm

- 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 \((p_1, p_4)\). The magnified schematic view below shows that we have a 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 either the hull shown in the figure on the right or...
Solutions

- Solutions for single algorithms.

- The Exact Geometric Computation Paradigm (ECG)
  - implement a Real-RAM to the extent needed in computational geometry
  - the challenge is efficiency
  - redesign the algorithms so that they can handle all inputs and have small arithmetic demand
  - Exact Computation Paradigm applies to all geometric algorithms
  - basis for LEDA, CGAL, and EXACUS

- Approximation via Controlled Perturbation
  - compute the correct result for a slightly perturbed input
  - initiated by Danny Halperin and co-workers and refined and generalized by us
  - Controlled perturbation applies to a large class of geometric algorithms
  - successfully used for Delaunay, Voronoi, arrangements of circles and spheres
Controlled Perturbation
Geometry of Float-Orient

- picture shows

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

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

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

- near the line many points are mis-classified

- outside a narrow strip around the curve of degeneracy, points are classified correctly !!!

- how narrow is narrow?
- true for all geometric predicates?
- if true, can we exploit to design reliable algorithms
Basics

- our program operates on points $q_1$ to $q_n$

  to perturb a point $q_i$:
  - move it to random point $p_i$ in the disk $B_\delta(q_i)$ of radius $\delta$ centered at $q_i$

- programs branch on the sign (+1, 0, -1) of expressions

- we use floating point arithmetic with mantissa length $L$

- the maximum error in evaluating an expression $E$ is $M_E$

  $M_E = \text{something} \cdot 2^{-L}$

- if $|E| > M_E$, it is safe to evaluate $E$ with floating point arithmetic and to branch on the sign of the result

- we have a geometric program that works for all non-degenerate inputs (if executed with exact real arithmetic)
Converting a Program to Controlled Perturbation

• guard every predicate evaluation, i.e.,
  
  replace \text{branch on sign of } E \text{ by}
  
  if (|E| \leq \text{max error in evaluation of } E) \text{ stop with exception;}
  \text{branch on sign of } E

• and then run the following master program
  • initialize $\delta$ and $L$ to convenient values
  • loop
    • perturb input
    • run the guarded algorithm with floating point precision $L$
    • if the program fails, double $L$ and rerun

• observe that program needs to be changed only slightly
  • guards for predicates and master loop

• guards can be avoided by use of interval arithmetic
Converting a Program to Controlled Perturbation

- guard every predicate evaluation, i.e.,
  
  replace \( \text{branch on sign of } E \) by
  
  if \( |E| \leq \text{max error in evaluation of } E \) stop with exception;
  branch on sign of \( E \)

- and then run the following master program
  
  - initialize \( \delta \) and \( L \) to convenient values
  
  - loop
    
    - perturb input
    
    - run the guarded algorithm with floating point precision \( L \)
    
    - if the program fails, double \( L \) and rerun

Theorem: For a large class of geometric programs: modified program terminates and returns the exact result for the perturbed input. Moreover (!!!), can quantify relation between \( \delta \) and \( L \).
How Narrow is Narrow?

- \( \text{orient}(p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)) = \text{sign}(E) \)
- \( E = 2 \cdot \text{signed area } \Delta \text{ of the triangle } (p, q, r) \)
- if coordinates are bounded by \( M \), maximal error in evaluating \( E \) with floating point arithmetic with mantissa length \( p \) is \( 28 \cdot M^2 \cdot 2^{-L} \)
- if \( 2|\Delta| > 28 \cdot M^2 \cdot 2^{-L} \), \( \text{float}_\text{orient} \) gives the correct result

\[ |\Delta| = \left(\frac{1}{2}\right)\text{dist}(q, r) \cdot \text{dist}(\ell(q, r), p) \]

- if \( \text{dist}(q, r) \cdot \text{dist}(\ell(q, r), p) > 28 \cdot M^2 \cdot 2^{-L} \), \( \text{float}_\text{orient} \) gives the correct result
- if \( \text{dist}(\ell((q, r), p)) \geq 28 \cdot M^2 \cdot 2^{-L}/\text{dist}(q, r) \), \( \text{float}_\text{orient}(p, q, r) \) gives the correct result.
How Narrow is Narrow?

- \( \text{orient}(p, q, r) = \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)) = \text{sign}(E) \)
- \( E = 2 \cdot \) signed area \( \Delta \) of the triangle \((p, q, r)\)
- if coordinates are bounded by \( M \), maximal error in evaluating \( E \) with floating point arithmetic with mantissa length \( p \) is \( 28 \cdot M^2 \cdot 2^{-L} \)
- Punch Line: if \( \text{dist}(\ell((q, r), p)) \geq 28 \cdot M^2 \cdot 2^{-L} / \text{dist}(q, r) \), \( \text{float}_\text{orient}(p, q, r) \) gives the correct result.

on the right, \( q \) and \( r \) have one third the distance than in figure on the left
Controlled Perturbation I

- consider algorithms using only the orientation predicate
- input points $q_1, \ldots, q_n$: perturb into $p_1, \ldots, p_n$ such that all evaluations for the perturbed points are f-safe.
Controlled Perturbation I

- consider algorithms using only the orientation predicate
- input points $q_1, \ldots, q_n$: perturb into $p_1, \ldots, p_n$ such that all evaluations for the perturbed points are $f$-safe.
- assume $p_1$ to $p_{n-1}$ are already determined:
  - choose $p_n$ in a circle of radius $\delta$ about $q_n$ such that whp
  - $p_n$ lies outside all strips of half-width $28 \cdot M^2 \cdot 2^{-L}/\text{dist}(p_i, p_j)$ about $\ell(p_i, p_j)$ for $1 \leq i < j \leq n - 1$
Controlled Perturbation I

- consider algorithms using only the orientation predicate
- input points \( q_1, \ldots, q_n \): perturb into \( p_1, \ldots, p_n \) such that all evaluations for the perturbed points are \( f \)-safe.
- assume \( p_1 \) to \( p_{n-1} \) are already determined:
  - choose \( p_n \) in a circle of radius \( \delta \) about \( q_n \) such that whp
  - \( p_n \) lies outside all strips of half-width \( 28 \cdot M^2 \cdot 2^{-L} / \text{dist}(p_i, p_j) \) about
  \( \ell(p_i, p_j) \) for \( 1 \leq i < j \leq n - 1 \)

- \( \text{whp} = (\text{choice of } p_n \text{ fails with prob } \leq 1/(2n)) \)
- prob, some choice fails is \( \leq 1/2 \)
- with prob \( 1/2 \), perturbed points are \( f \)-safe
- need that strips cover at most fraction \( 1/(2n) \) of ball \( B_\delta(q_n) \)
Controlled Perturbation II

- assume $p_1$ to $p_{n-1}$ are already determined:
  - choose $p_n$ in a circle of radius $\delta$ about $q_n$ such that whp
  - $p_n$ lies outside all strips of half-width $28 \cdot M^2 \cdot 2^{-L}/\text{dist}(p_i, p_j)$ about $\ell(p_i, p_j)$ for $1 \leq i < j \leq n - 1$
  - need that strips cover at most fraction $1/(2n)$ of ball $B_\delta(q_n)$
Controlled Perturbation II

1. Assume $p_1$ to $p_{n-1}$ are already determined:
   - Choose $p_n$ in a circle of radius $\delta$ about $q_n$ such that whp
   - $p_n$ lies outside all strips of half-width $28 \cdot M^2 \cdot 2^{-L} / \text{dist}(p_i, p_j)$ about $\ell(p_i, p_j)$ for $1 \leq i < j \leq n - 1$
   - Need that strips cover at most fraction $1 / (2n)$ of ball $B_\delta(q_n)$

2. A small problem: strips can be arbitrarily wide
3. IDEA: also guarantee $\text{dist}(p_i, p_j) > \gamma$ for some $\gamma$
4. Then size of forbidden region $\leq n \cdot \pi \cdot \gamma^2 + n^2 \cdot (28 \cdot M^2 \cdot 2^{-L} / \gamma) \cdot 2 \cdot \delta$
Controlled Perturbation II

• assume $p_1$ to $p_{n-1}$ are already determined:
  • choose $p_n$ in a circle of radius $\delta$ about $q_n$ such that whp
  • $p_n$ lies outside all strips of half-width $28 \cdot M^2 \cdot 2^{-L}/\text{dist}(p_i, p_j)$ about
    $\ell(p_i, p_j)$ for $1 \leq i < j \leq n-1$
  • need that strips cover at most fraction $1/(2n)$ of ball $B_\delta(q_n)$
• IDEA: also guarantee $\text{dist}(p_i, p_j) > \gamma$ for some $\gamma$
• then size of forbidden region $\leq n \cdot \pi \cdot \gamma^2 + n^2 \cdot (28 \cdot M^2 \cdot 2^{-L}/\gamma) \cdot 2 \cdot \delta$
• want: size of FR $\leq \pi \cdot \delta^2/(2n)$
Controlled Perturbation II

- assume $p_1$ to $p_{n-1}$ are already determined:
  - choose $p_n$ in a circle of radius $\delta$ about $q_n$ such that whp
  - $p_n$ lies outside all strips of half-width $28 \cdot M^2 \cdot 2^{-L} / \text{dist}(p_i, p_j)$ about $\ell(p_i, p_j)$ for $1 \leq i < j \leq n-1$
  - need that strips cover at most fraction $1/(2n)$ of ball $B_\delta(q_n)$

- IDEA: also guarantee $\text{dist}(p_i, p_j) > \gamma$ for some $\gamma$

- then size of forbidden region $\leq n \cdot \pi \cdot \gamma^2 + n^2 \cdot (28 \cdot M^2 \cdot 2^{-L} / \gamma) \cdot 2 \cdot \delta$

- want: size of FR $\leq \pi \cdot \delta^2 / (2n)$

- fix $\gamma$ so as to minimize FR and obtain

$$\text{any } L \geq 2 \log(M/\delta) + 4 \log n + 9 \text{ works}$$

- $M = 1000, \delta = 0.001, n = 1000, L \geq 2 \cdot 20 + 4 \cdot 10 + 9 = 89$
Generalization to All (??) Geometric Predicates

<table>
<thead>
<tr>
<th>General</th>
<th>Orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>predicate $P(x_1, \ldots, x_k) = \text{sign} f(x_1, \ldots, x_k)$</td>
<td>orientation $(p, q, r)$</td>
</tr>
<tr>
<td>$x_1$ to $x_k$ points (in the plane)</td>
<td></td>
</tr>
<tr>
<td>$x = (x_1, \ldots, x_{k-1})$ fixed, $x = x_k$ variable</td>
<td>$q, r$ fixed, $p$ variable</td>
</tr>
<tr>
<td>$C_x = {x : f(x, x) = 0}$, curve of degeneracy</td>
<td>$C = {p : \text{orient}(p, q, r) = 0}$</td>
</tr>
<tr>
<td>$C_x$ is either the entire plane or a curve</td>
<td>plane or $\ell(q, r)$</td>
</tr>
</tbody>
</table>

Relate $f(x, x)$ to the distance of $x$ from $C_x$.

$$f(x, x) \geq g(x) \cdot \text{dist}(C_x, x)$$

Forbidden region becomes tubular neighborhood of $C_x$ of width $M_f / g(x)$

analyse $g(x)$ recursively
Generalization II

- in ICALP 06 paper, we show how to analyse a large class of predicates in the same way
  - predicates with a fixed number of arguments

- controlled perturbation applies to any algorithm
  - using only predicates as above and
  - whose running time is bounded as a function of number of input points

- most algorithms in CGAL are covered
The Exact Computation Paradigm
Improved Algs: Arrangements of Algebraic Curves

- algebraic curve = zero set of a polynomial in variables $x$ and $y$
- assume rational coefficients
- $x^2 + y^2 = 9$ defines circle of radius 3
- compute $x$-coordinates of event points (vertical tangents, singularities, intersections)
- event point are algebraic numbers

- substitute $x$-values into algebraic curves and determine the roots of the resulting equations in $y$
- this requires to determine roots of polynomials with algebraic coefficients

- Seidel/Wolpert: CompGeo 2005: can do with roots of polynomials with rational coefficients
Efficient Computation with Algebraic Numbers

- $p(x) = \sum_{0 \leq i \leq n} p_i x^i$, a polynomial of degree $n$
- $p_n \geq 1, p_i \leq 2^\tau$ for all $i$  \hspace{1cm} \tau$ bits before binary point
- $sep(p) = \text{minimum distance between any two roots of } p$, the root separation of $p$.
- **Theorem:** Isolating intervals for real roots can be computed in time polynomial in $n$ and $\tau + \log 1/sep(p)$.

- more precisely, $O(n^4(\tau + \log(1/sep(p)))^2)$ bit operations
  requires $O(n(\tau + \log(1/sep(p))))$ bits of each coefficient
- for integer coefficients, our algorithm has the same complexity as previous algs
- experiments: $p(x)$ a polynomial with integer coefficients
  running times on $p(x)$, $\pi \cdot p(x)$, and $\sqrt{2} \cdot p(x)$ are essentially the same

Eigenwillig/Kettner/Krandick/Mehlhorn/Schmitt/Wolpert: CASC 2005
Summary

Most existing implementations (commercial or academic) are unreliable
  • may crash or produce non-sensical answers

Where do we stand?
  • we = reliable geometric algorithms project at MPI + EU-projects CGAL, GALIA, ECG and ACS
  • linear (lines, planes, points) geometry in 2d and 3d: nice academic work + industrial impact
  • curved geometry in 2d: nice academic work + industrial impact
  • curved geometry in 3d: nice academic work
  • implementations available in LEDA, CGAL, and EXACUS (ESA 2005)

How do we work?
  • develop the required theory and build prototypical systems to validate the theory and to have impact beyond our own community