Summer School on Algorithms: Course on Exact Geometric Computation

Chee Yap
Courant Institute of Mathematical Sciences
New York University

Abstract

Most geometric software today will crash because of numerical errors. If you have used CAD software, you will know this experience first-hand. Can we ever produce numerical and geometric software that are free of numerical nonrobustness? Some have declared that this is impossible.
Motivated by such widespread problems of nonrobustness, researchers in the last decade have been developing a range of techniques to address the problem. An approach called Exact Geometric Computation has proven highly successful; this approach has been encoded into libraries such as LEDA, CGAL, and Core Library.
The basic capability of such libraries amounts to guaranteeing arbitrary (user-specified) precision in a numerical computation. To be useful, this capability has to be efficient enough to compete with current nonrobust software. Such techniques lie at the interface between algebraic and numerical computation.
In these lectures, we introduce the theory and techniques underlying this kind of computation. Students will use the Core Library for programming exercises.

For textbook, we use a book manuscript "Robust Geometric Computation" by K.Mehlhorn and C.Yap (download from http://cs.nyu.edu/yap/bks/egc/ for some chapters). For information about Robust Geometric Computation see http://cs.nyu.edu/exact/. Core Library is also distributed with Geometry Factory, the commercial version of CGAL (see

http://www.geometryfactory.com/).

COURSE MECHANICS

- There are four days of $3\frac{1}{2}$ hours sessions.
- Each session is divided into 2 or 3 lectures, followed by 20-30 minute problem sessions.
- Will assign exercises and programming assignments involving the Core Library. Reference for programming is [Core Tutorial].
- You must work in groups of two each. Two groups may form a supergroup of size 4.
- At the end of this course, I hope to pick one group for a prize. I don't have any fixed criteria, but perhaps that will evolve as we proceed.
- A general reference for these lectures will be my book with Mehlhorn, downloadable at my announced website. I will refer to this book as [Mehlhorn-Yap].

COURSE OVERVIEW

	DAY ONE
1	What is EGC?
2	Core Library and Precision-Driven Computation
	DAY TWO
3	Algebraic Computation
4	Numeric Computation
5	Filter Technology
	DAY THREE
6	Curves
7	Surfaces
8	Perturbation
	DAY FOUR
9	Theory of Real Computation
10	Transcendental Computation
11	Shortest Path Amidst Discs

Lecture 1. What is EGC?

- I. Numerical Nonrobustness Phenomenon
- II. This is EGC

Nonrobustness Examples

- Software Crashes
  - intermittent, numerical causes
- Mesh Generation
  - Point Classification Problem
- Surface-Surface Intersection (SSI) in CAD
  - Considered unsolved
- Front Tracking in Physics Simulation
  - Front surface may self-intersect

Responses to Nonrobustness

- "It is a rare event"
- "Use more stable algorithms"
- "Avoid ill-conditioned inputs"
- "Epsilon-tweaking"
- "There is no solution"
- "Our competitors cannot solve it either" (CAD companies)

What is Epsilon-Tweaking, really?

- "Never compare directly to 0"
  - IF $x = 0$ THEN ...
  - IF $|x| \leq \varepsilon$ THEN ...
- Replace lines by fat lines and fat points
- What is the intersection of two fat lines?
- How to consistently work out the geometry of fat objects?

Why we must solve this

- Scientific Productivity
  - Programmers/researchers' time in debugging
  - All CAD software can fail
- Economic Impact of Nonrobustness
  - Barrier to full automation in industry
  - Mesh Generation: 1 failure per 5 million cells [Aftosmis]
- Mission Critical Computation
  - Patriot missile in Gulf War (1990)
  - French Ariane Rocket failure (1995)
  - North Sea Oil Platform Collapse (1996)

CFD in Aircraft Industry

In Computational Fluid Dynamics (CFD) applications with 50 million elements:

- 10-20 minutes for surface mesh generation
- 3-4 hours for volume meshing
- 1 hour for actual flow analysis
- 2-4 weeks for geometry repair

– T. Peters and D. Ferguson (Boeing Company)

Results from CAD Software

- Example from Mehlhorn and Hart (2001)
- Boolean Operation Test:
  - (1) Construct regular n-gon P
  - (2) Rotate P by α degrees to form Q
  - (3) Form union $R = P \cup Q$. So R is now a $4n$-gon

(contd.) CAD Software

Results from several commercial systems:

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 min	too difficult!
Microstation95	100	1.0e-2	2 sec	correct
Microstation95	100	0.5e-2	3 sec	incorrect!
Rhino3D	200	1.0e-2	15 sec	correct
Rhino3D	400	1.0e-2	–	crash!
CGAL/LEDA	5000	6.175e-6	30 sec	correct
CGAL/LEDA	5000	1.581e-9	34 sec	correct
CGAL/LEDA	20000	9.88e-7	141 sec	correct

The last three rows, from the CGAL/LEDA software, are always correct.

This last column shows 3 forms of failure: *crashing, looping or silent error*.

Accuracy as a Contractual Issue

- Geometric Tolerancing in Manufacturing Science
  - Manufacturing is inherently imprecise
- Moore's Law in modern Precision Engineering
  - "On a Log scale, precision increases linearly with time in each industry." (Voelcker)
- What is $3 \pm 0.1cm \times 5 \pm 0.2cm$?
  - Naive interpretation vs. zone interpretation
- Industry Standards (ASME Y14.5M-1982), (ISO)
- How do we measure conformance?

  - Assessment methodology via state-of-art CMMs

- CMM (Coordinate Measuring Machines) are hardwares coupled with software.

- Richard Walker (1988): Government Alert
  - Software errors are widely divergent
  - Crisis in procurement procedures

- Economic cost of over- or under-tolerancing
  - The $10,000 mistake/paper weight

- Anecdoc: Aluminum soda can tops: 0.35 ± 0.05mm

  - If improved to 0.33 ± 0.03mm, save millions/year

- Another standard problem: measuring roundness
  - [Mehlhorn-Shermer-Yap, SCG'97] Complete Procedure

PART II:
THIS IS EGC

Geometric Predicates

- Signed Area

$$\Delta(A,B,C) := \det \begin{bmatrix} a & a' & 1 \\ b & b' & 1 \\ c & c' & 1 \end{bmatrix}$$

$$= \det \begin{bmatrix} a & a' & 1 \\ b-a & b'-a & 0 \\ c-a & c'-a & 0 \end{bmatrix} = \det \begin{bmatrix} b-a & b'-a' \\ c-a & c'-a' \end{bmatrix}$$

where $A = (a, a')$, $B = (b, b')$, etc.

- Basic Properties
  - Continuity
  - Predicate $LeftTurn(A,B,C) \equiv$ "$\Delta(A,B,C) > 0$"
- Generalization to d-Dimensions

* Signed volume, $\Delta(A_0, \ldots, A_d) = \det \begin{bmatrix} A_0 & 1 \\ A_1 & 1 \\ \vdots & \\ A_d & 1 \end{bmatrix}$

Elementary Problems of CG

- Convex Hull
- Arrangement of Hyperplanes
- Voronoi Diagrams
- Shortest Paths amidst Polygonal Obstacles

Approaches to Nonrobustness

- Illustration: what is an "approximate line"?
  - Pixels on a grid [Bresenham line]
  - Fat lines or zones [Sequin-Segal]
  - Polygonal lines [Greene-Yao, Milenkovic]
  - Finite precision parameter lines [Sugihara]
- General Approaches:
  - Finite Precision Geometry
  - Interval Geometry
  - Consistency Approach
  - Topological Approach
- Why they don't work

What is Geometry?

- Euclid, Descartes, Hilbert, Dieudonne,...
- Geometry is about discrete relations
  - Is a point on a line?
  - Does a plane cut a sphere?
- "Geometry = Combinatorics + Numerics"
- Consistency Requirement must hold !
  - If not, we get a qualitative error
- E.g., Convex Hull, Voronoi diagrams

On the Consistency Approach

- Example of Non-Geometric Computation:
  - Shortest Path Problem for graphs with edge weights
- Fortune's Concept of Parsimonious Algorithm
  - Never make a redundant comparison!
- Why this approach is not practical:
  - [Hoffmann, Hopcroft, Karasick]: consistent intersection of 3 convex polygons
- Topological Approach of Sugihara
  - A form of consistency approach

* Focus of selected properties to be preserved – e.g., planarity

How to Compute Exactly, in the Geometric Sense

- Algorithm = Sequence of Steps
- Steps = Computation or Tests
- Tests determine the branching path
- Combinatorial relations are determined by path
- THEREFORE, if we take the correct path, the combinatorics will be correct (i.e., consistent)
- – THIS SLIDE IS THE "TAKE-HOME MESSAGE"!

Features of EGC

- Naive EGC – compute all numbers without error
  - Infeasible for linear geometry [Yu, Purdue Thesis 1991]
- Exactness is in the geometry, NOT the arithmetic
- Approximate Numbers can be (MUST BE) exploited
- Standard geometry
  - No need for "finite precision geometry"
- All standard geometric algorithms can be robust!
  - No need for new "robust algorithms" for each problem

- Our solution is general (algebraic case)
- General EGC Number Library
  - Compare other approaches
  - Core Library [Yap et al (1999)]
  - Leda Real [Mehlhorn et al (1999)]
  - Any programmer can write robust programs!

What if I really want Inexactness?

- First response: Assume Nominal Exactness
  - E.g., convex hull for inexact points
- What if Nominal Exactness isn't Possible?
  - Reformulate the problem as another (harder) exact problem
  - E.g., triangulation for inexact points from a surface
- Exactness is the best way of achieving consistency
  - "It is NOT about exactness, but about consistency"

Robustness as a Resource

- Robustness need not be a All-Or-Nothing Proposition
- The Robustness-Speed Trade-off Curve
- How to exploit Moore's Law
- Outline of a Compiler-based approach

Beyond EGC?

- Guaranteed Accuracy Computation
  - A priori precision guarantees
- Interval Arithmetic or Validated Computation
  - A posteriori precision guarantees
- p-bits of Absolute Error:
  - $|\widetilde{x} - x| \leq 2^{-p}$.
- p-bits of Relative Error:
  - $|\widetilde{x} - x| \leq 2^{-p}|x|$.
- OBSERVATION: guaranteeing one relative bit implies sign guarantee

Conclusions

- Nonrobustness problems is important to solve for economic and scientific reasons
- Through EGC has shown that nonrobustness, for a large class of problems, can be systematically solved
- EGC and Guaranteed Accuracy Computation as a new mode of numerical computation, with many applications
- An area at the interface of numerical, algebraic and geometric computation

Exercises

IMPLEMENT THIS ALGORITHM IN THE MOST STRAIGHTFORWARD WAY YOU CAN. USE ARRAYS INSTEAD OF LINKED LISTS, ETC.
GENERAL REMARK ABOUT IMPLEMENTATION: We are less interested in Object-Oriented Programming (OOP) techniques than in algorithms. So, I would be happy if you make all variables public, etc.

An Incremental Convex Hull Algorithm

The incremental algorithm maintains the current convex hull CH of the points seen so far. Initially, CH is formed by choosing three non-collinear points in S. It then considers the remaining points one by one. When considering a point r, it first determines whether r is outside the current convex hull polygon. If not, r is discarded. Otherwise, the hull is updated by forming the tangents from r to CH and updating CH appropriately.
The algorithm maintains the current hull as a circular list $L = (v_0, v_1, \ldots, v_{k-1})$ of its extreme points in counter-clockwise order. The line segments (v_i, v_{i+1}), $0 \leq i \leq k-1$ (indices are modulo k) are the *edges* of the current hull. If orient$(v_i, v_{i+1}, r) < 0$, we say r *sees* the edge (v_i, v_{i+1}), and say the edge (v_i, v_{i+1}) is *visible* from r. If orient$(v_i, v_{i+1}, r) \leq 0$, we say that the edge (v_i, v_{i+1}) is *weakly visible* from r. After initialization, $k \geq 3$. The following properties are key to the operation of the algorithm.
Property A: A point r is outside CH iff r can see an edge of CH.
Property B: If r is outside CH, the edges weakly visible from r form a non-empty consecutive subchain; so do the edges that are not weakly visible from r.
If $(v_i, v_{i+1}), \ldots, (v_{j-1}, v_j)$ is the subsequence of weakly visible edges, the updated hull

is obtained by replacing the subsequence $(v_{i+1}, \ldots, v_{j-1})$ by r. The subsequence $(v_i, \ldots, v_j)$ is taken in the circular sense. E.g., if $i > j$ then the subsequence is $(v_i, \ldots, v_{k-1}, v_0, \ldots, v_j)$. From these properties, we derive the following algorithm:

1. Initialize L to the counter-clockwise triangle (a, b, c).
   Remove a, b, c from S.
2. FOR ALL $r \in S$
   IF (there is an edge e visible from r)
   Compute the sequence $(v_i, \ldots, v_j)$ of edges that are weakly visible from r;
   Replace the subsequence $(v_{i+1}, \ldots, v_{j-1})$ by r;

To turn the sketch into an algorithm, we provide more information about the substeps.

1. How does one determine whether there is an edge visible from r? We iterate over the edges in L, checking each edge using the orientation predicate. If no visible edge is found, we discard r. Otherwise, we take any one of the visible edges as the starting edge for the next item.
2. How does one identify the subsequence $(v_i, \ldots, v_j)$? Starting from a visible edge e, we move counter-clockwise along the boundary until a non-weakly-visible edge is encountered. Similarly, move clockwise from e until a non-weakly-visible edge is encountered.
3. How to update the list L? We can delete the vertices in $(v_{i+1}, \ldots, v_{j-1})$ after all

visible edges are found, as suggested in the above sketch ("the off-line strategy") or we can delete them concurrently with the search for weakly visible edges ("the on-line strategy").

There are four logical ways to negate Properties A and B:

- **Failure (A_1):** A point outside the current hull sees no edge of the current hull.
- **Failure (A_2):** A point inside the current hull sees an edge of the current hull.
- **Failure (B_1):** A point outside the current hull sees all edges of the convex hull.
- **Failure (B_2):** A point outside the current hull sees a non-contiguous set of edges.

Failures (A_1) and (A_2) are equivalent to the negation of Property A. Similarly, Failures (B_1) and (B_2) are complete for Property B if we take (A_1) into account. All these failures can be realized in machine precision arithmetic.
EXPERIMENTS: Run your program on the following input, in Levels 1 and 3. Explain what you see.
(1)

$$\begin{aligned} p_1 &= (7.3000000000000194, 7.3000000000000167) \\ p_2 &= (24.000000000000068, 24.000000000000071) \\ p_3 &= (24.000000000000005, 24.000000000000053) \\ p_4 &= (0.50000000000001621, 0.50000000000001243) \\ p_5 &= (8,4) \quad p_6 = (4,9) \quad p_7 = (15,27) \\ p_8 &= (26,25) \quad p_9 = (19,11) \end{aligned}$$

(2)

$$
\begin{aligned}
p_1 &= (27.64356435643, -21.881188118811881)\\
p_2 &= (83.36633663663366, 15.544554455445542)\\
p_3 &= (4, 4)\\
p_4 &= (73.41584158415 8, 8.861386138595)
\end{aligned}
$$

(3)

$$
\begin{aligned}
p_1 &= (200, 49.200000000000003)\\
p_2 &= (100, 49.600000000000001)\\
p_3 &= (-233.33333333333334, 50.933333333333333)\\
p_4 &= (166.66666666666669, 49.333333333333336)
\end{aligned}
$$

(4)

$$
\begin{aligned}
p_1 &= (0.50000000000001243, 0.50000000000000189)\\
p_2 &= (0.50000000000001243, 0.50000000000000333)\\
p_3 &= (24.000000000000005, 24.000000000000053)\\
p_4 &= (24.000000000000068, 24.000000000000071)\\
p_5 &= (17.300000000000001, 17.300000000000001)
\end{aligned}
$$

(5)

$$
\begin{aligned}
q_1 &= (0.10000000000000001, 0.10000000000000001)\\
q_2 &= (0.20000000000000001, 0.20000000000000004)\\
q_3 &= (0.79999999999999993, 0.80000000000000004)\\
q_4 &= (1.267650600228229 \cdot 10^{30}, 1.267650600228291 \cdot 10^{30})
\end{aligned}
$$

(6)

$p_1 = (24.000000000000005, 24.000000000000053)$ $p_2 = (24.0, 6.0)$

$p_3 = (54.85, 6.0)$ $p_4 = (54.850000000000357, 61.000000000000121)$

$p_5 = (24.000000000000068, 24.000000000000071)$ $p_6 = (6, 6).$

REFERENCES

- Paper "Robust Geometric Computation" in Handbook of Computational Geometry (eds.Goodman and O'Rourke), CRC Press, 2nd Edition, 2004. http://cs.nyu.edu/yap/papers/
- Chapter 1 of [Mehlhorn-Yap]

"A rapacious monster lurks within every computer, and it dines exclusively on accurate digits."
– B.D. McCullough (2000)

THE END