Implementing Geometric Algorithms for Real-World Applications
With and Without EGC-Support

Stefan Huber\(^1\) Martin Held\(^2\)

\(^1\)Institute of Science and Technology Austria

\(^2\)FB Computerwissenschaften
Universität Salzburg, Austria

GCC 2013, Rio de Janeiro, Brazil
June 17–20
Outline

1. Three industrial codes and their design principles:
   - FIST
   - VRONI
   - STALGO

2. Adding CORE and MPFR backend.
FIST

- Triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation [Held, 2001a].

- Handles
  - degenerate input,
FIST

- Triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation [Held, 2001a].

- Handles
  - degenerate input,
  - self-overlapping input,
FIST

- Triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation [Held, 2001a].

- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input.
FIST

- Triangulates polygons with holes in 2D and 3D,
  - based on ear-clipping and
  - multi-level geometric hashing to speed up computation [Held, 2001a].

- Handles
  - degenerate input,
  - self-overlapping input,
  - self-intersecting input.

- No Delaunay triangulation, but heuristics to generate “decent” triangles.
- Typical applications in industry: triangulation of (very) large GIS datasets, triangulation of “planar” faces of 3D models.
Vroni/ArcVroni

- Computes Voronoi diagrams of
  - points, straight-line segments and circular arcs,
  - based on randomized incremental insertion and a topology-oriented approach [Held and Huber, 2009, Held, 2001b].

Also computes (weighted) medial axis, offset curves, and maximum-inscribed circle.

Typical applications in industry: generation of tool paths (e.g., for machining or sintering), generation of buffers in GIS applications.
Vroni/ArcVroni

- Computes Voronoi diagrams of
  - points, straight-line segments and circular arcs,
  - based on randomized incremental insertion and a topology-oriented approach [Held and Huber, 2009, Held, 2001b].

- Also computes
  - (weighted) medial axis,
Vroni/ArcVroni

- Computes Voronoi diagrams of
  - points, straight-line segments and circular arcs,
  - based on randomized incremental insertion and a topology-oriented approach [Held and Huber, 2009, Held, 2001b].

- Also computes
  - (weighted) medial axis,
  - offset curves, and
  - maximum-inscribed circle.
Vroni/ArcVroni

- Computes Voronoi diagrams of
  - points, straight-line segments and circular arcs,
  - based on randomized incremental insertion and a topology-oriented approach [Held and Huber, 2009, Held, 2001b].

- Also computes
  - (weighted) medial axis,
  - offset curves, and
  - maximum-inscribed circle.

- Typical applications in industry: generation of tool paths (e.g., for machining or sintering), generation of buffers in GIS applications.
Computing straight skeletons of planar straight-line graphs, based on a refined wavefront propagation using the motorcycle graph [Huber and Held, 2012, Huber, 2012].
Computing straight skeletons of

- planar straight-line graphs,
- based on a refined wavefront propagation using the motorcycle graph [Huber and Held, 2012, Huber, 2012].

Also computes

- Mitered offset curves, and
Computing straight skeletons of
planar straight-line graphs,
based on a refined wavefront propagation using the motorcycle graph [Huber and Held, 2012, Huber, 2012].

Also computes
- Mitered offset curves, and
- roof models resp. terrains.
Success stories

- More than 100 commercial licenses world-wide for FIST, Vroni/ArcVroni and STALGO.
  - A few hundred Euros (for ArcVroni) up to a few thousand Euros/Dollars (FIST, VRONI, STALGO).
Success stories

- More than 100 commercial licenses world-wide for FIST, Vroni/ArcVroni and STALGO.
  - A few hundred Euros (for ArcVroni) up to a few thousand Euros/Dollars (FIST, VRONI, STALGO).

- “Industrial-strength” implementations achieved:
  - Only a handful of bug reports in more than ten years
  - of heavy commercial and academic use, and lots of satisfied customers.
Datasets from industry

- Real-world data often means no quality at all:
  - brute-force simplifications / approximations of data,
  - data cleaned up manually and “visually”,
  - etc.

- As a consequence:
  - All sorts of degeneracies, self-intersections, tiny gaps, etc.

General position must not be assumed.

Data sizes:
- From a few thousand segments/arcs in a machining application
- to a few million segments in a GIS application.
Efficiency requirements

- From real-time map generation on a smart phone
- to minutes of CPU time allowed on some high-end machine.
- In general, linear space complexity and a close-to-linear time complexity is expected.
Engineering principles: Use alternative computations

- Algebraically equivalent terms need not be equally reliable on fp arithmetic.
  - Check whether a computation becomes unstable, and use an alternative approach.
- Sample application: Compute the bisector $b$ between $f$ and $g$. 
Engineering principles: Use alternative computations

- Algebraically equivalent terms need not be equally reliable on fp arithmetic.
  - Check whether a computation becomes unstable, and use an alternative approach.
- Sample application: Compute the bisector $b$ between $f$ and $g$. 

![Diagram showing the bisector between two lines](image)
Engineering principles: Topology-oriented approach

- First used by Sugihara et alii [1992, 2000].
- Define topological criteria that the output has to meet.
  - Use fp-computations to choose among different topological set-ups if two or more set-ups fulfill all criteria.

Sample application:
- Incremental insertion of a point into a Voronoi diagram.
- The portion of the Voronoi diagram to be deleted forms a tree.
Engineering principles: Topology-oriented approach

- First used by Sugihara et alii [1992, 2000].
- Define topological criteria that the output has to meet.
  - Use fp-computations to choose among different topological set-ups if two or more set-ups fulfill all criteria.

- Sample application:
  - Incremental insertion of a point into a Voronoi diagram.
  - The portion of the Voronoi diagram to be deleted forms a tree.
Engineering principles: Topology-oriented approach

- First used by Sugihara et alii [1992, 2000].
- Define topological criteria that the output has to meet.
  - Use fp-computations to choose among different topological set-ups if two or more set-ups fulfill all criteria.

- Sample application:
  - Incremental insertion of a point into a Voronoi diagram.
  - The portion of the Voronoi diagram to be deleted forms a tree.
Engineering principles: Topology-oriented approach

- First used by Sugihara et alii [1992, 2000].
- Define topological criteria that the output has to meet.
  - Use fp-computations to choose among different topological set-ups if two or more set-ups fulfill all criteria.

Sample application:
- Incremental insertion of a point into a Voronoi diagram.
- The portion of the Voronoi diagram to be deleted forms a tree.
Engineering principles: Topology-oriented approach

- First used by Sugihara et alii [1992, 2000].
- Define topological criteria that the output has to meet.
  - Use fp-computations to choose among different topological set-ups if two or more set-ups fulfill all criteria.

- Sample application:
  - Incremental insertion of a point into a Voronoi diagram.
  - The portion of the Voronoi diagram to be deleted forms a tree.
Engineering principles: Epsilon relaxation

1. TypicalComputationalUnit()
2. begin
3. \( \epsilon \leftarrow \epsilon_{\text{min}} \) // Set \( \epsilon \) to maximum precision
4. while \( \epsilon \leq \epsilon_{\text{max}} \) do
5. result \( \leftarrow \) ComputeUnit(\( \epsilon \)) // Compute some data
6. if CheckResult(result, \( \epsilon \)) then
7. return result // Topological/numerical checks
8. else
9. ComputeUnitReset()
10. \( \epsilon \leftarrow 10 \cdot \epsilon \) // Relaxation of epsilon
11. end
12. end
13. if not CheckInputLocally() then // Is input sound?
14. CleanInputLocally() // Fix problems in the input
15. RestartComputationGlobally() // Restart from scratch
16. else
17. return ComputeUnitDesperateMode() // Time to hope for the best
18. end
19. end
Engineering principles: Avoiding geometric decisions

- Simulation of wavefront propagation, DCEL

extended wavefront
Engineering principles: Avoiding geometric decisions

- Simulation of wavefront propagation, DCEL

[Diagram showing wavefront propagation and DCEL]
Engineering principles: Avoiding geometric decisions

- Simulation of wavefront propagation, DCEL
- Straight-forward: remove $e_1, e_2, v_1, v_2$
Engineering principles: Avoiding geometric decisions

- Simulation of wavefront propagation, DCEL
- Straight-forward: remove $e_1, e_2, v_1, v_2$
  - Add $v$ and relink it with $v'_1, v'_2$.
  - Involves geometric decisions! And multiple events can occur simultaneously.
Engineering principles: Avoiding geometric decisions

- Simulation of wavefront propagation, DCEL
- Straight-forward: remove $e_1, e_2, v_1, v_2$
  - Add $v$ and relink it with $v'_1, v'_2$.
  - Involves geometric decisions! And multiple events can occur simultaneously.
- Better: remove $v_1, v_2$ but *repost* $e_1, e_2$ to $v$.
  - No geometric decisions involved.
Adding CORE backend

Canonical adaptions:

▶ Set $\text{EPS}$ to 0.
▶ Migrate $\text{fabs}(\text{expr}) < \text{EPS}$ to $\text{fabs}(\text{expr}) \leq \text{EPS}$. 
Adding CORE backend

Canonical adaptations:

- Set EPS to 0.
- Migrate fabs(expr) < EPS to fabs(expr) <= EPS.

Migrating C to C++:

- printf("%f", val); scanf("%f", &val);
- malloc, free → new, delete

More subtle problems encountered:

- Expr::intValue() rounds "inexact":
 -rounds up or down, depending on expression tree.
- Decision based on finitely many bits.
- Work-around: migrate intValue() to floor().
Adding CORE backend

Canonical adaptations:

▶ Set \( \text{EPS} \) to 0.
▶ Migrate \( \text{fabs}(\text{expr}) < \text{EPS} \) to \( \text{fabs}(\text{expr}) \leq \text{EPS} \).

Migrating C to C++:

▶ \text{printf}("%f", \text{val}); \text{scanf}("%f", &\text{val});
▶ \text{malloc}, \text{free} \rightarrow \text{new}, \text{delete}

More subtle problems encountered:

▶ \text{Expr::intValue()} rounds “inexact”:
  ▶ Rounds up or down, depending on expression tree.
  ▶ Decision based on finitely many bits.
  ▶ Work-around: migrate \text{intValue()} to \text{floor}().
Summary:

- FIST works with CORE.
- Vroni and Stalgo could not be executed.
  - Willi Mann’s bug fixes and performance patches in CORE-2.1.
  - Still, several CPU-minutes did not suffice to determine sign of a single expression stemming from simple inputs.
Adding MPFR backend

Canonical adaptations:

- \( \text{EPS} \) needs to depend on precision.
  - We used a heuristic formula:
    \[
    \text{EPS} := \epsilon_{fp} \cdot 2^{-100 \cdot \left( \sqrt{\text{prec}/53} - 1 \right)},
    \]
  - where \( \epsilon_{fp} \) is the former machine-precision \( \text{EPS} \).
Adding MPFR backend

Canonical adaptions:

- EPS needs to depend on precision.
  - We used a heuristic formula:

\[
\text{EPS} := \epsilon_{fp} \cdot 2^{-100 \cdot (\sqrt{\text{prec}/53} - 1)},
\]

where \(\epsilon_{fp}\) is the former machine-precision EPS.

Practical work required:

- MPFR is not shipped with a C++ wrapper.
  - Code that generates wrapper classes with the required operators overloaded.
- Partial C to C++ migration, as for CORE.
Experimental results: FIST

- 21175 polygons (w/ and w/o holes).
- Six arithmetic configurations:
  - fistFp, fistShew, fistCore, fistMp{53, 212, 1000}
Experimental results: FIST

- 21175 polygons (w/ and w/o holes).
- Six arithmetic configurations:
  - fistFp, fistShew, fistCore, fistMp\{53, 212, 1000\}
- Conclusion:
  - Shewchuck’s predicates have negligible impact on speed.
  - fistMP* about 24× slower than fistFp.
  - fistCore about 60× slower than fistFp.

**Figure:** Runtime per seconds divided by $n \log n$. fistFp, fistMp212, fistCore.
Experimental results: FIST

Correctness of inexact configurations?

- Verification code:
  - Bentley-Ottmann, implemented with exact \( \text{mpq\_t} \) from GMP.
  - Take 0.1 as closest fp-number using \texttt{atof()}.
- No errors found!

Conclusion: Non-exactness no practical issue in pure fp applications.
Experimental results: Voronoi diagrams

- Vroni versus CGAL.
- 18787 polygons ($< 100000$ vertices)
- Six configurations:
  - vroniFp, vroniMp{53, 212, 1000}, cgvdFp
  - cgvdEx: CORE-based predicate kernel
Experimental results: Voronoi diagrams

- **Conclusion:**
  - vroniMp* about 50–70× slower than vroniFp.
  - cgvd* about 50–80× slower than vroniFp.
  - cgvdFp only 1.5× faster than cgvdEx.
    - Crashed on 937 datasets due to fp-exception.
  - On average, cgvdEx slightly faster than vroniMp*.
    - cgvdEx timings vary by a factor of 20.
    - A few cgvdEx results were numerically clearly wrong.

**Figure:** Runtime per seconds divided by $n \log n$. vroniFp, vroniMp212, cgvdEx.
Experimental results: Voronoi diagrams

Numerical precision of Voronoi nodes:

- **Deviation**: difference in the distances of a node to its defining sites.
- **Violation**: another site is closer to a node than defining sites.

![Graph showing deviation and violation with different precision levels](image)

**Figure**: Left: Deviation. Right: violation
EGC: A simple case study

A function \texttt{test}(N):

- Generate a shuffled array \texttt{A} with elements $\pm k_1, \ldots, \pm k_N$, with $k_i$ being random integers.
- We build the sum \texttt{S} over \texttt{A}.
- How long does $\texttt{S} == \texttt{Expr(0)}$ take?
A function \texttt{test}(N):

- Generate a shuffled array \( A \) with elements \( \pm k_1, \ldots, \pm k_N \), with \( k_i \) being random integers.
- We build the sum \( S \) over \( A \).
- How long does \( S == \text{Expr}(0) \) take?

Results depend on the set-up:

- Are filters working?
- How is the sum built?
  - Naive for-loop, or
  - in a balanced fashion.

The “default case”: with filters, naive for-loop.
CORE, naive sum:
- \(O(n^2)\) time
- w/ filter: \(O(n^2)\) mem

LEDAs: virtually zero runtime
EGC: a simple case study

What if we put stress on the filters?

- Add to the array $A$ five times $\sqrt{2}$ and $-\sqrt{2}$.
- How long will $S == \text{Expr}(0)$ take now?
naive sum:
- $O(n^2)$ time
- $O(n^2)$ mem

balanced sum:
- $O(1)$ or $O(n)$ time
- $O(n)$ mem
- filters have more impact

Disclaimer: Of course, these expressions will unlikely occur in real-world software.
EGC/MPFR: Conclusion

▶ EGC software can be fast, see Shewchuck’s Triangle.

Height-balancing expression trees might reduce the costs for time and space significantly.

We might observe different complexities in terms of big-Oh.

On- and offline structural optimization strategies for expression trees are worth to be investigated.

Adding EGC support to non-trivial software a-posteriori can be extremely challenging.

Different programming styles due to focus on either numerical accuracy or awareness of expression trees.

EGC-aware programming right from the start is necessary.

Adding MPFR support is straightforward.

MPFR boosts numerical accuracy.

MPFR helps to distinguish numerical errors from logical bugs.

Precision-elevation instead of epsilon-relaxation?
EGC/MPFR: Conclusion

- EGC software can be fast, see Shewchuck’s Triangle.
- Height-balancing expression trees might reduce the costs for time and space significantly.
  - We might observe different complexities in terms of big-Oh.
  - On- and offline structural optimization strategies for expression trees are worth to be investigated.
EGC/MPFR: Conclusion

- EGC software can be fast, see Shewchuck’s Triangle.
- Height-balancing expression trees might reduce the costs for time and space significantly.
  - We might observe different complexities in terms of big-Oh.
  - On- and offline structural optimization strategies for expression trees are worth to be investigated.
- Adding EGC support to non-trivial software a-posteriori can be extremely challenging.
  - Different programming styles due to focus on either numerical accuracy or awareness of expression trees.
  - EGC-aware programming right from the start is necessary.
EGC/MPFR: Conclusion

- EGC software can be fast, see Shewchuck’s Triangle.
- Height-balancing expression trees might reduce the costs for time and space significantly.
  - We might observe different complexities in terms of big-Oh.
  - On- and offline structural optimization strategies for expression trees are worth to be investigated.
- Adding EGC support to non-trivial software a-posteriori can be extremely challenging.
  - Different programming styles due to focus on either numerical accuracy or awareness of expression trees.
  - EGC-aware programming right from the start is necessary.
- Adding MPFR support is straight-forward
  - MPFR boosts numerical accuracy.
  - MPFR helps to distinguish numerical errors from logical bugs.
  - Precision-elevation instead of epsilon-relaxation?
Discontinuous problems and EGC

Straight skeletons can change discontinuously with the input:

The polygon is stored with finite precision to a file.

$\text{fp-codes are likely to produce the left skeleton/roof, which is intended.}$

$\text{EGC-codes produce the right skeleton/roof, which is undesired.}$

What is the lesser evil?

Either waive EGC,

Or forsake the desired output of the algorithm.
Discontinuous problems and EGC

Straight skeletons can change discontinuously with the input:

The polygon is stored with finite precision to a file.

fp-codes are likely to produce the left skeleton/roof, which is intended.

EGC-codes produce the right skeleton/roof, which is undesired.

What is the lesser evil?

Either waive EGC,

or forsake the desired output of the algorithm.
Discontinuous problems and EGC

Straight skeletons can change discontinuously with the input:

- The polygon is stored with finite precision to a file.
- fp-codes are likely to produce the left skeleton/roof, which is intended.
- EGC-codes produce the right skeleton/roof, which is undesired.

What is the lesser evil?

- Either waive EGC,
- Or forsake the desired output of the algorithm.

Stefan Huber, Martin Held: Geometric Algorithms for Real-World Applications

Open problems 26 of 29
Discontinuous problems and EGC

Straight skeletons can change discontinuously with the input:

- The polygon is stored with finite precision to a file.
  - fp-codes are likely to produce the left skeleton/roof, which is intended.
  - EGC-codes produce the right skeleton/roof, which is undesired.

What is the lesser evil?

- Either waive EGC,
- Or forsake the desired output of the algorithm.
Discontinuous problems and EGC

Straight skeletons can change discontinuously with the input:

- The polygon is stored with finite precision to a file.
  - fp-codes are likely to produce the left skeleton/roof, which is intended.
  - EGC-codes produce the right skeleton/roof, which is undesired.

What is the lesser evil?
- Either waive EGC,
- Or forsake the desired output of the algorithm.
Discontinuous problems and EGC

- $f$ is discontinuous on a sub-space $S$ (red) of the input space.
- “Reversed simulation of simplicity”?
A common yardstick

“Our algorithm runs in $O(n \log n)$ time in practice.”

“Our implementation behaved reliable in our tests.”
A common yardstick

“Our algorithm runs in $O(n \log n)$ time in practice.”

“Our implementation behaved reliable in our tests.”

However:

- Experiments often comprise only a few datasets.
- Datasets have no diversity.
- Different papers compare against different data, if at all.
A common yardstick

A standard computational geometry dataset library (SCGDL) would have many benefits:
A common yardstick

A **standard computational geometry dataset library (SCGDL)** would have many benefits:

- Experiments become more meaningful and comparable:
  - Precise timings and memory consumption.
  - How often did an implementation crash?
  - How many results were wrong?

- Enables a culture of extensive experimental evaluation.
- Brings CG and industry closer together.
- Implementing reliable geometric codes requires testing.
- An incentive to provide “gapless” and practical descriptions of algorithms.
A common yardstick

A standard computational geometry dataset library (SCGDL) would have many benefits:

- Experiments become more meaningful and comparable:
  - Precise timings and memory consumption.
  - How often did an implementation crash?
  - How many results were wrong?
- Enables a culture of extensive experimental evaluation.
  - Brings CG and industry closer together.
A common yardstick

A standard computational geometry dataset library (SCGDL) would have many benefits:

- Experiments become more meaningful and comparable:
  - Precise timings and memory consumption.
  - How often did an implementation crash?
  - How many results were wrong?
- Enables a culture of extensive experimental evaluation.
  - Brings CG and industry closer together.
- Implementing reliable geometric codes requires testing.
  - An incentive to provide “gapless” and practical descriptions of algorithms.
Figure: Taken from http://joyreactor.com/post/818128
FIST: Fast Industrial-Strength Triangulation of Polygons.

VRONI: An Engineering Approach to the Reliable and Efficient Computation of Voronoi Diagrams of Points and Line Segments.

Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight-Line Segments.

*Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*.
Shaker Verlag.

A Fast Straight-Skeleton Algorithm Based on Generalized Motorcycle Graphs.