The Arithmetic Toolbox in CGAL

Computational Geometry Algorithms Library

Sylvain Pion

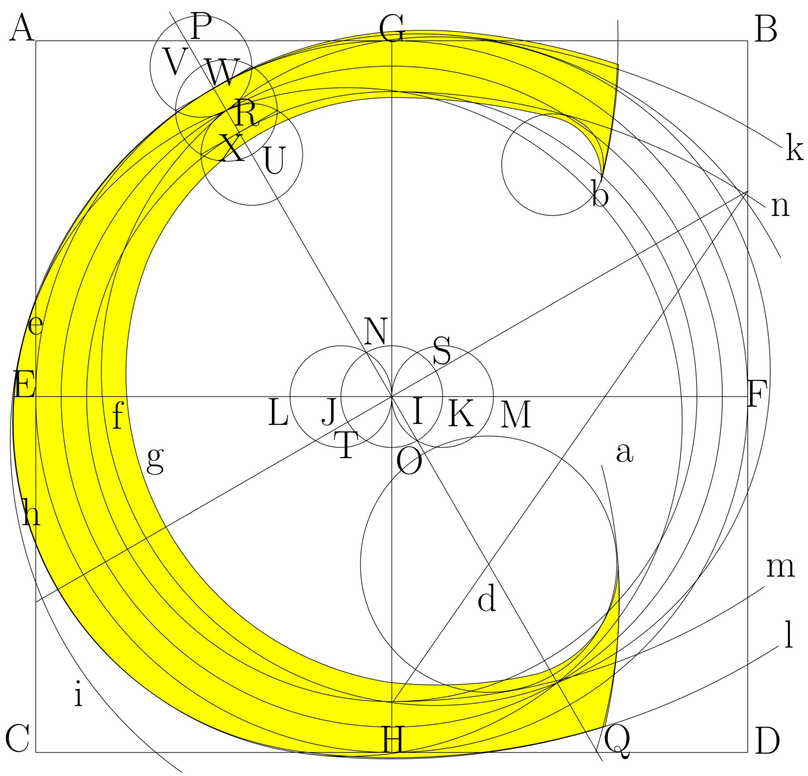
Mini-workshop on iRRAM/MPC/MPFR

Schloß Dagstuhl

April 18-20, 2018

Talk outline

- Brief overview of CGAL
- Robustness issues
- Arithmetic toolbox



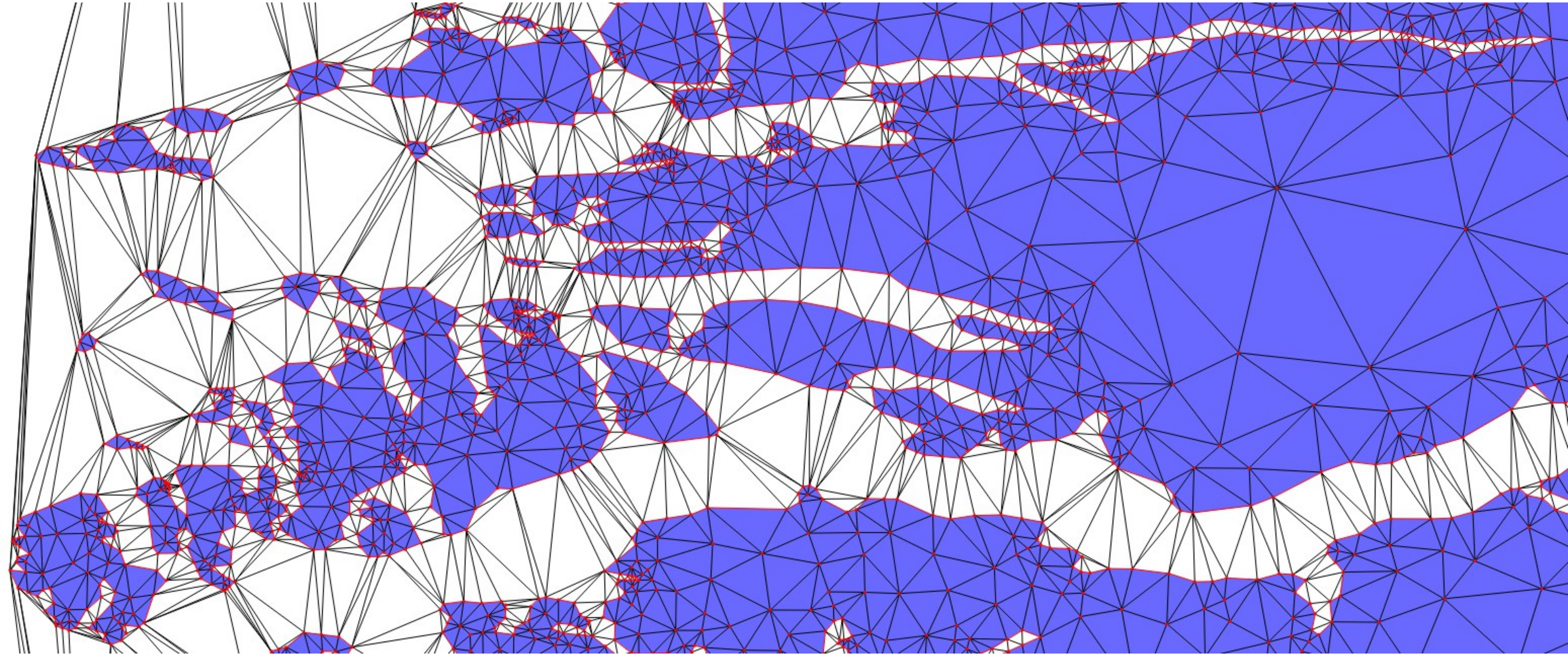
Brief overview of CGAL

CGAL project

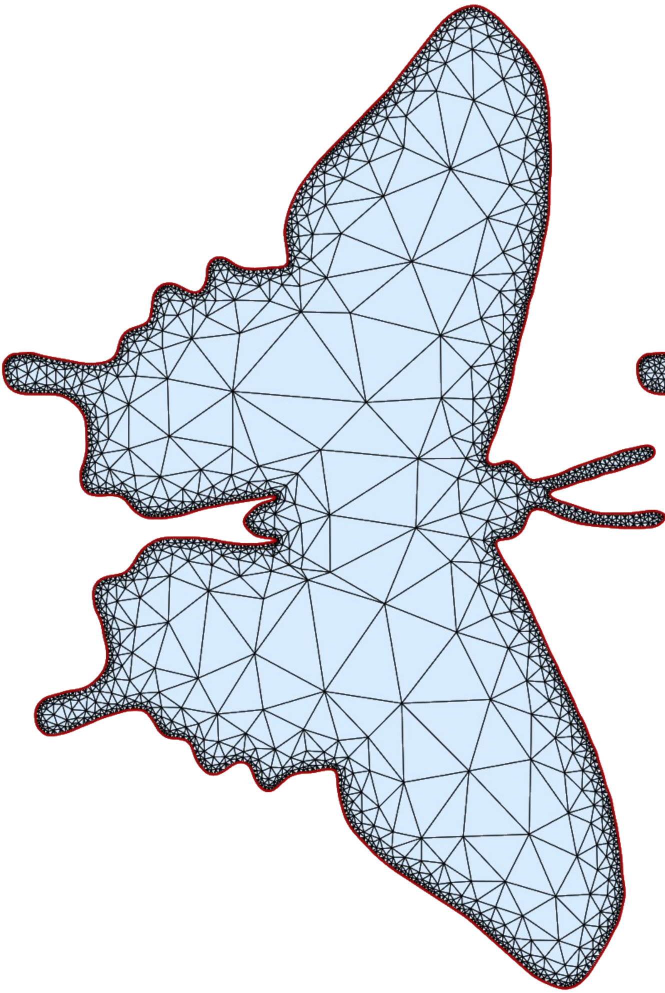
- 20 years old project started in European academia
- 1M C++ template lines of geometric algorithms
- Open Source (GPL) and Commercial licenses
- >100 of commercial customers from :
  - CAD/CAM, VLSI, Digital maps, GIS, Medical, Telecom, Geophysics (Oil&Gas)...
- Key design goals :
  - Genericity (app. domain independent algorithms)
  - Robustness
  - Efficiency

Triangulations of Polygons

Triangulations of Polygons

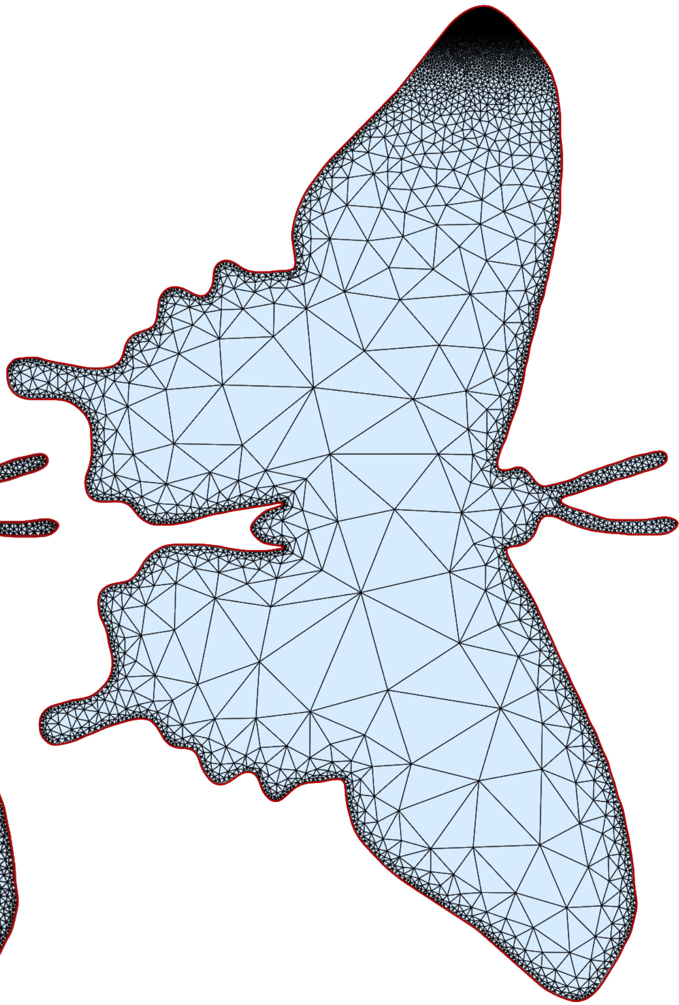


Parametrizable Meshes



Without constraint

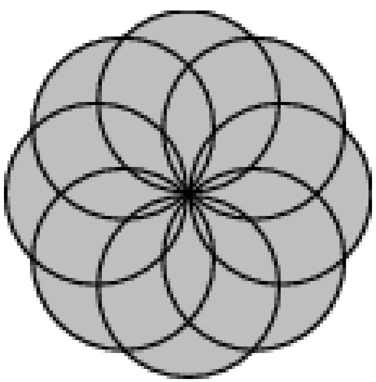
Uniform



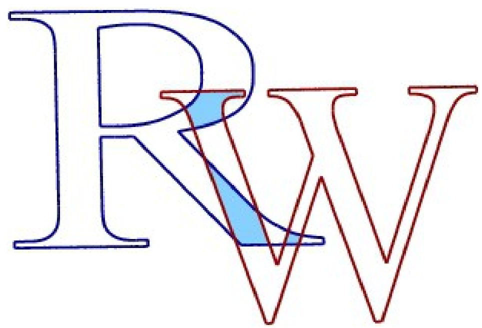
Size function

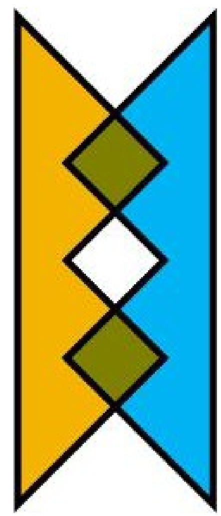
Boolean Operations

Union

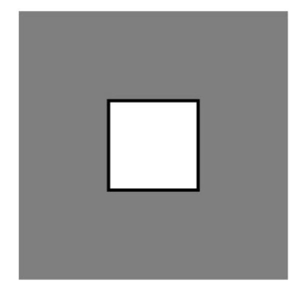


Intersection





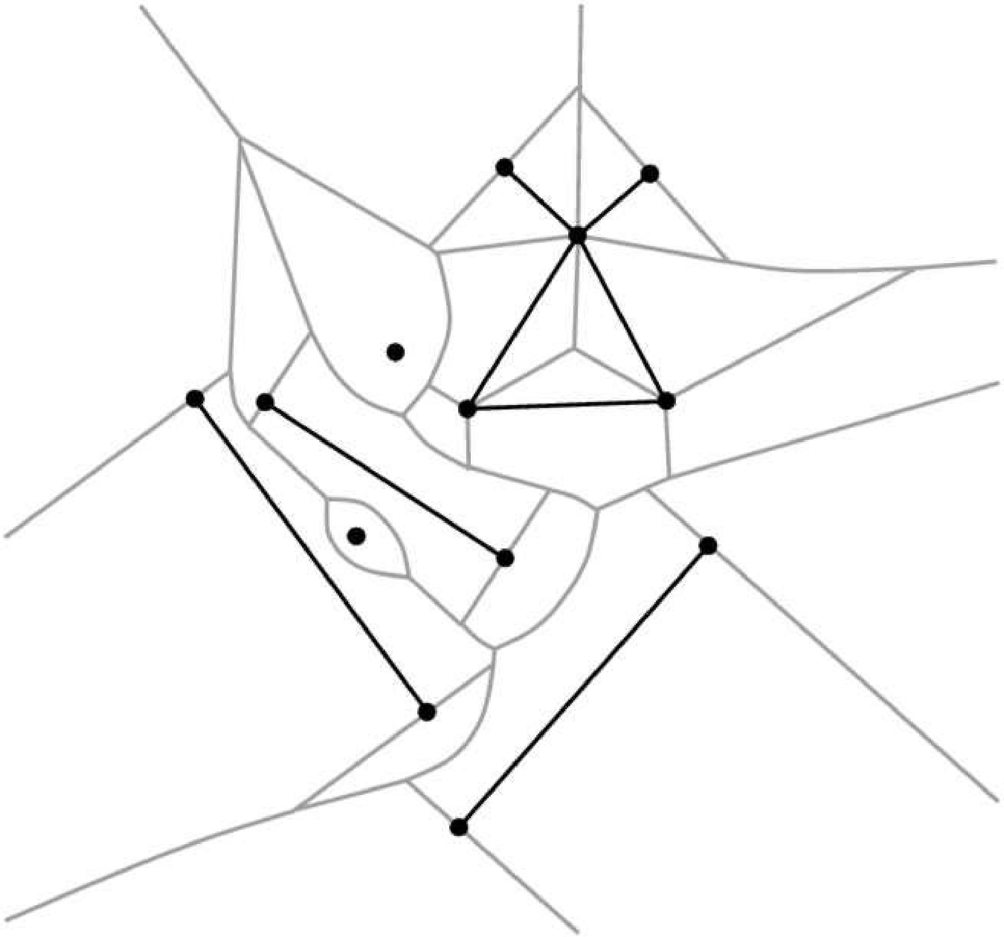
Complement



Support for :

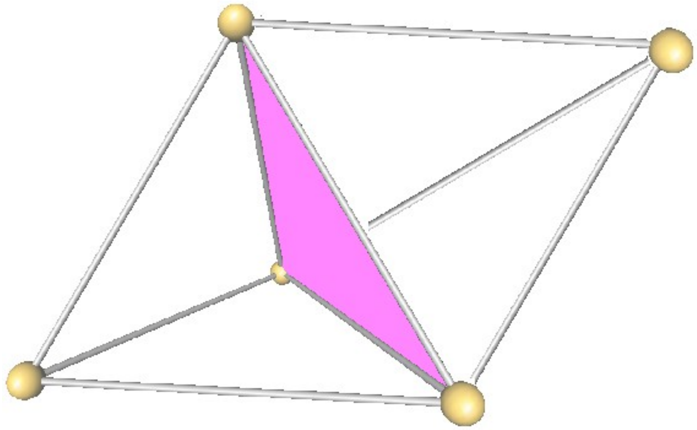
Line segments, Circular arcs, Bezier curves

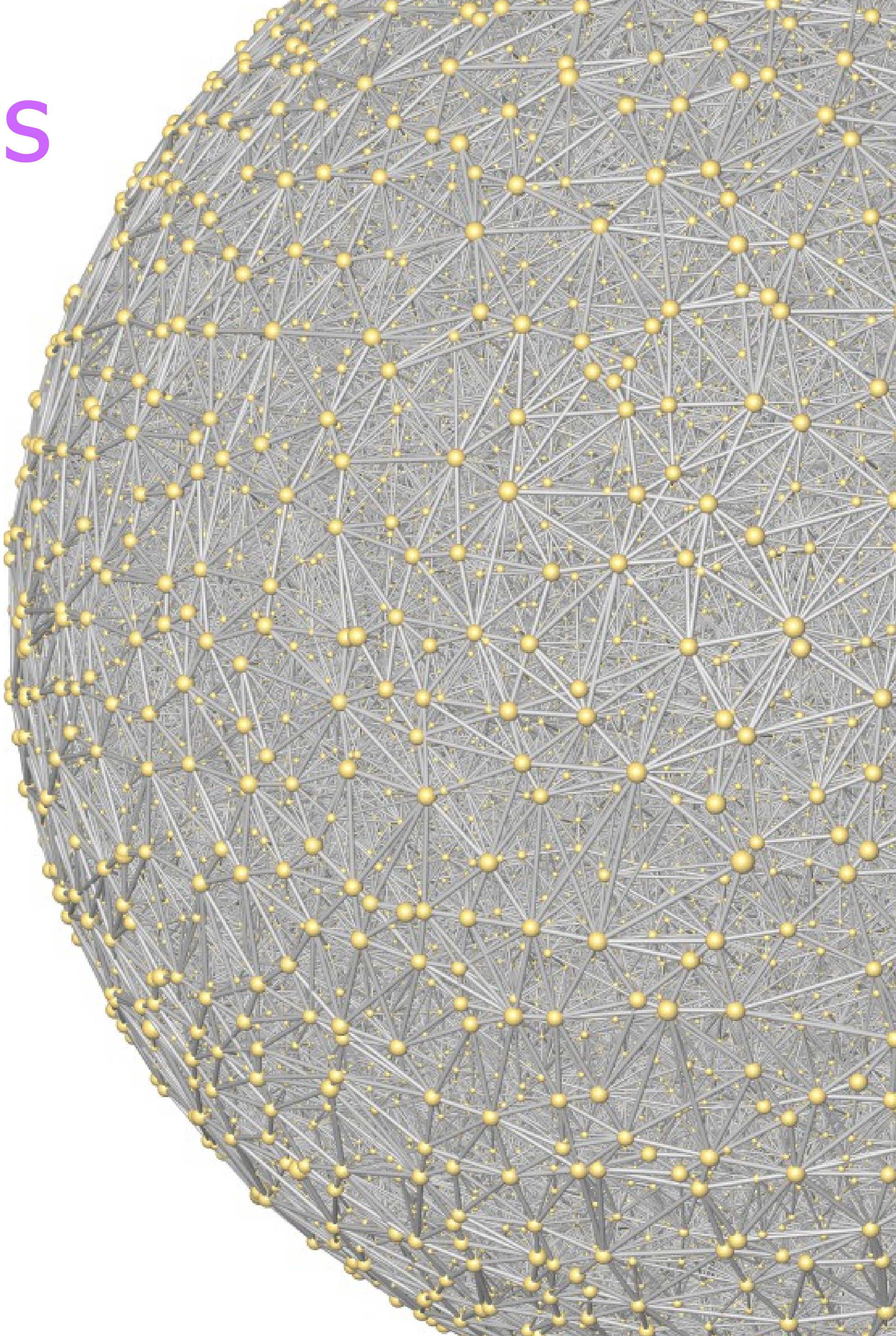
Voronoi diagrams of line segments



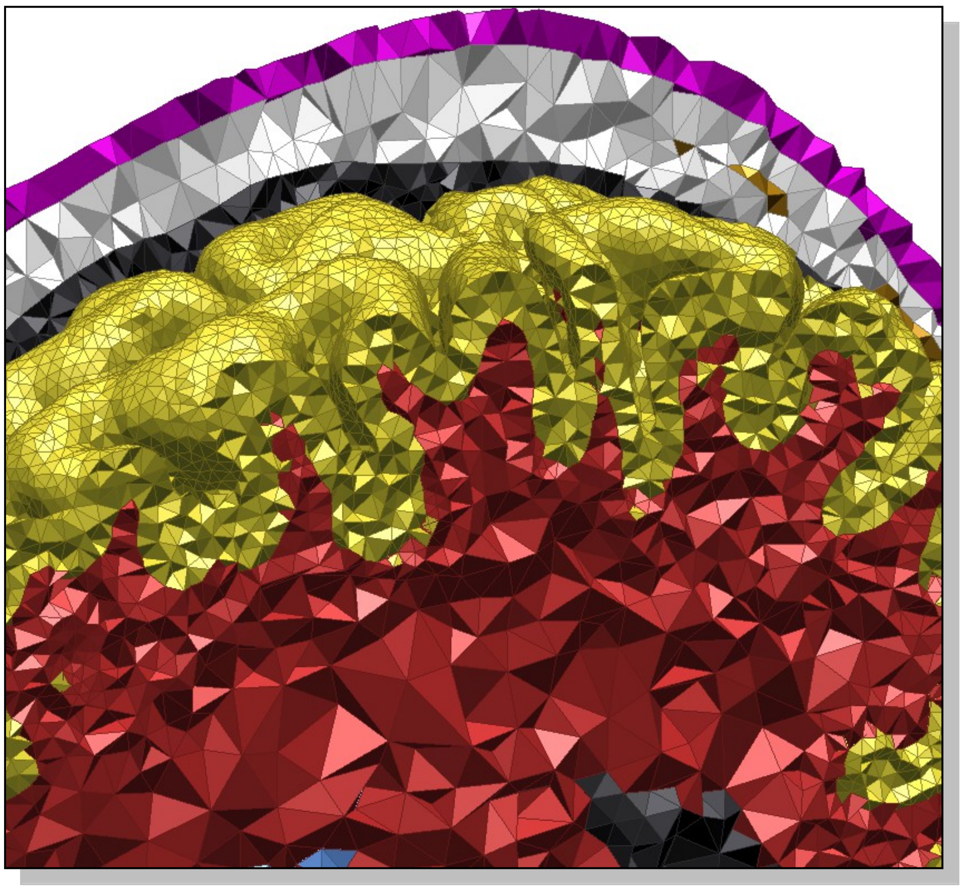
3D Triangulations

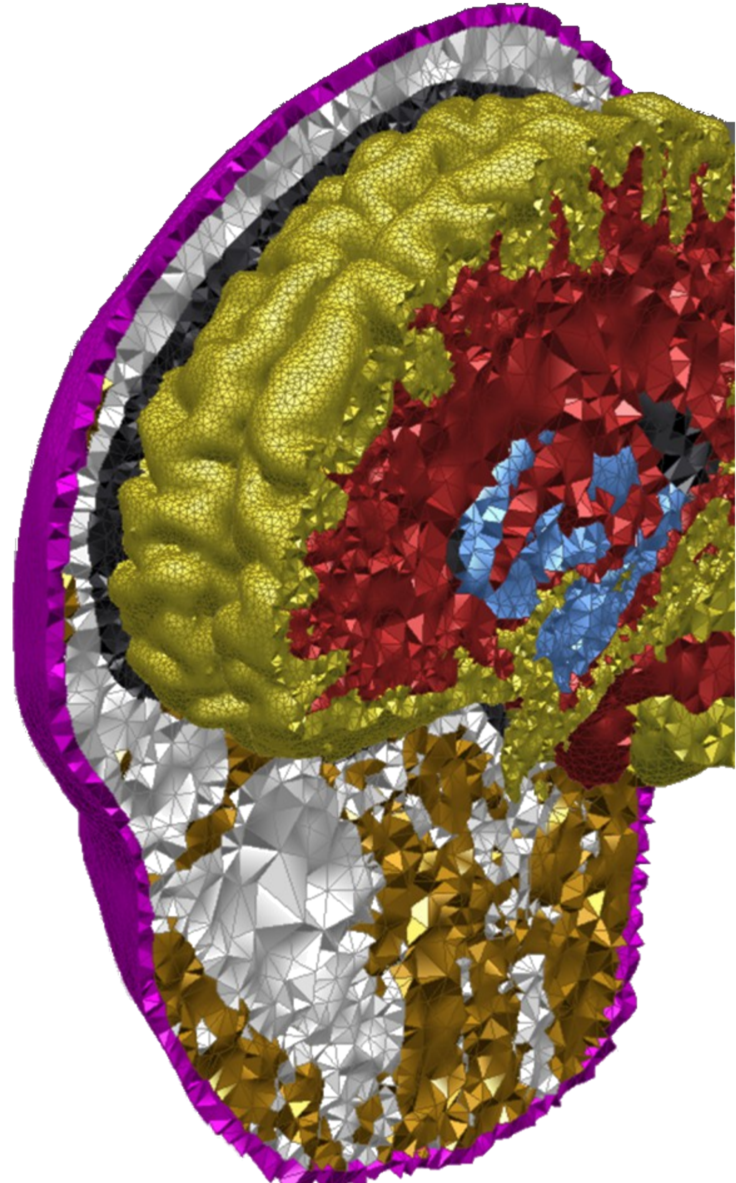
- Delaunay and weighted
- Dynamic structure
- 1M points in 16s

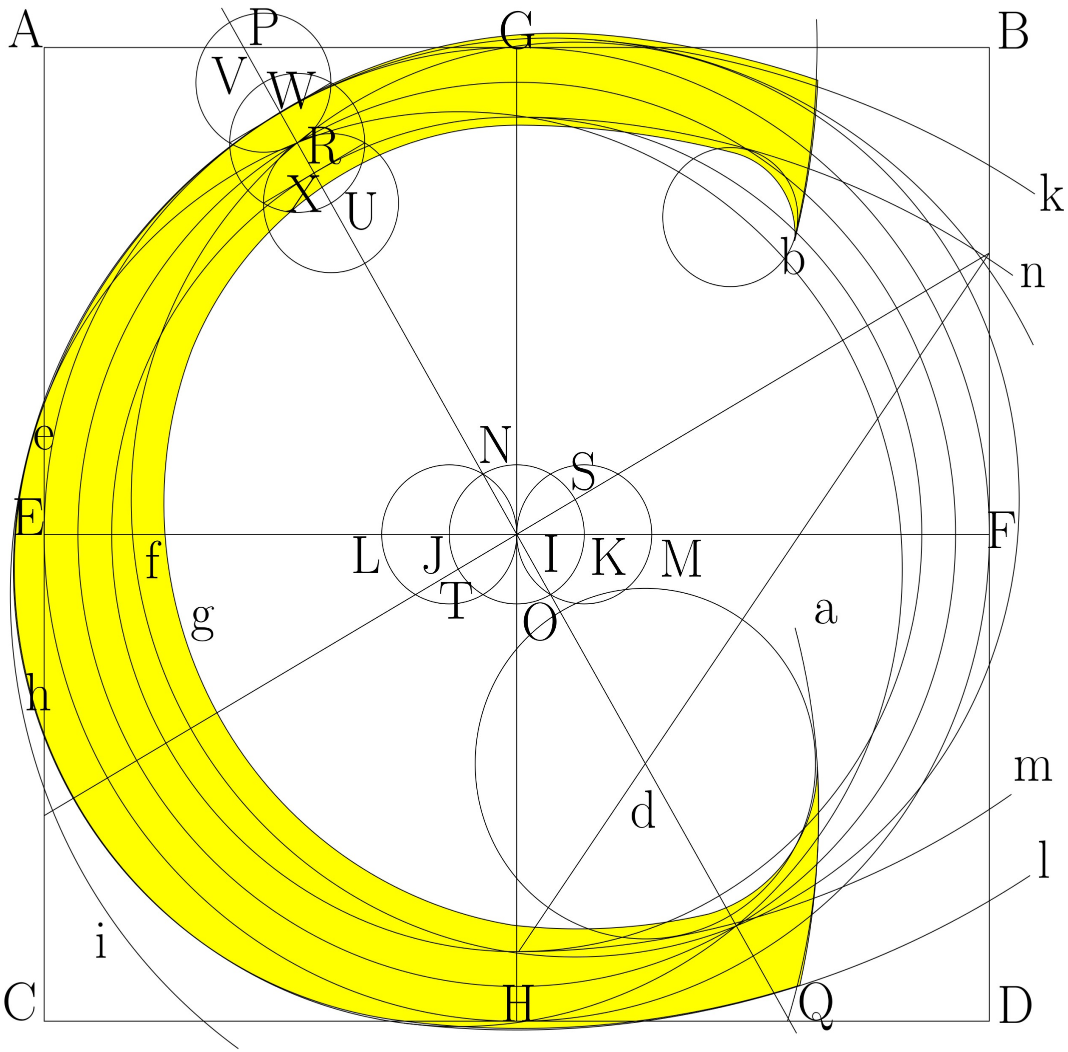




Volumic mesh (segmented image)

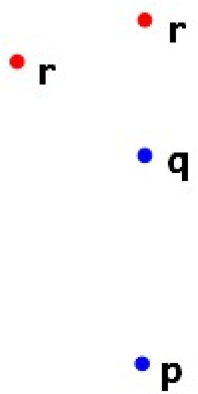




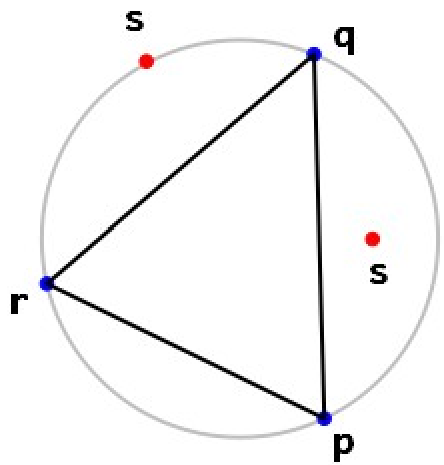


Robustness
-
Exact Geometric Computing

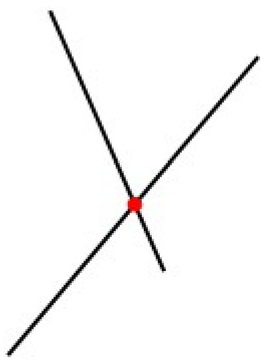
Predicates and Constructions



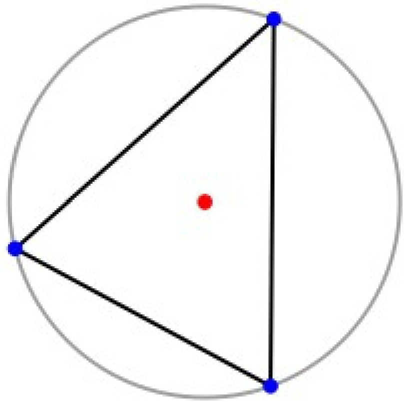
orientation



in_circle



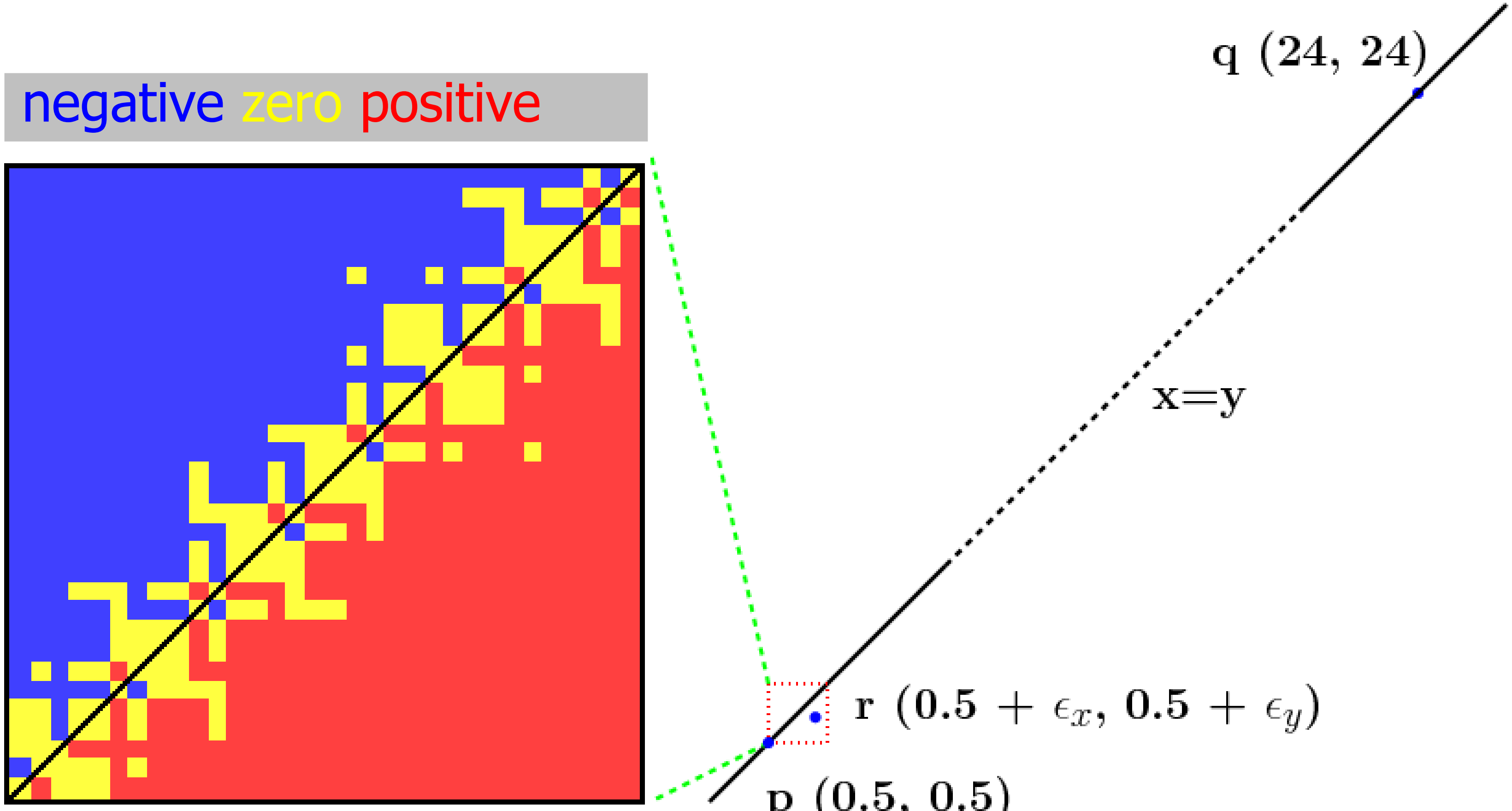
intersection



circumcenter

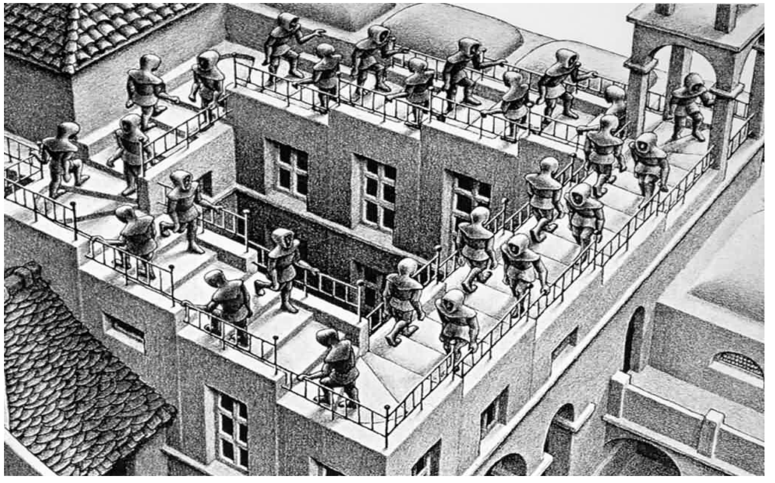
Floating-point direct evaluation

orientation(p,q,r) = $\mathrm{sign}((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x))$



Non-robustness problems

- Direct use of FP arithmetic causes algorithms to
  - Produce [slightly] wrong output
  - Crash on invariant violation
  - Infinite loop

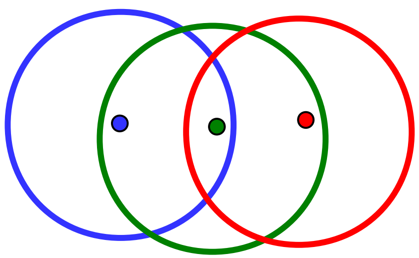


- Geometry is different
  - in theory
  - using FP arithmetic

Partial solution : ε heuristics

« *If value < ε, consider value == 0* »

- Transitivity: p = q, q = r, p != r



- Does not generalize well
- Not beautiful / mathematically appealing

Better solution : Exact Geometric Computing paradigm

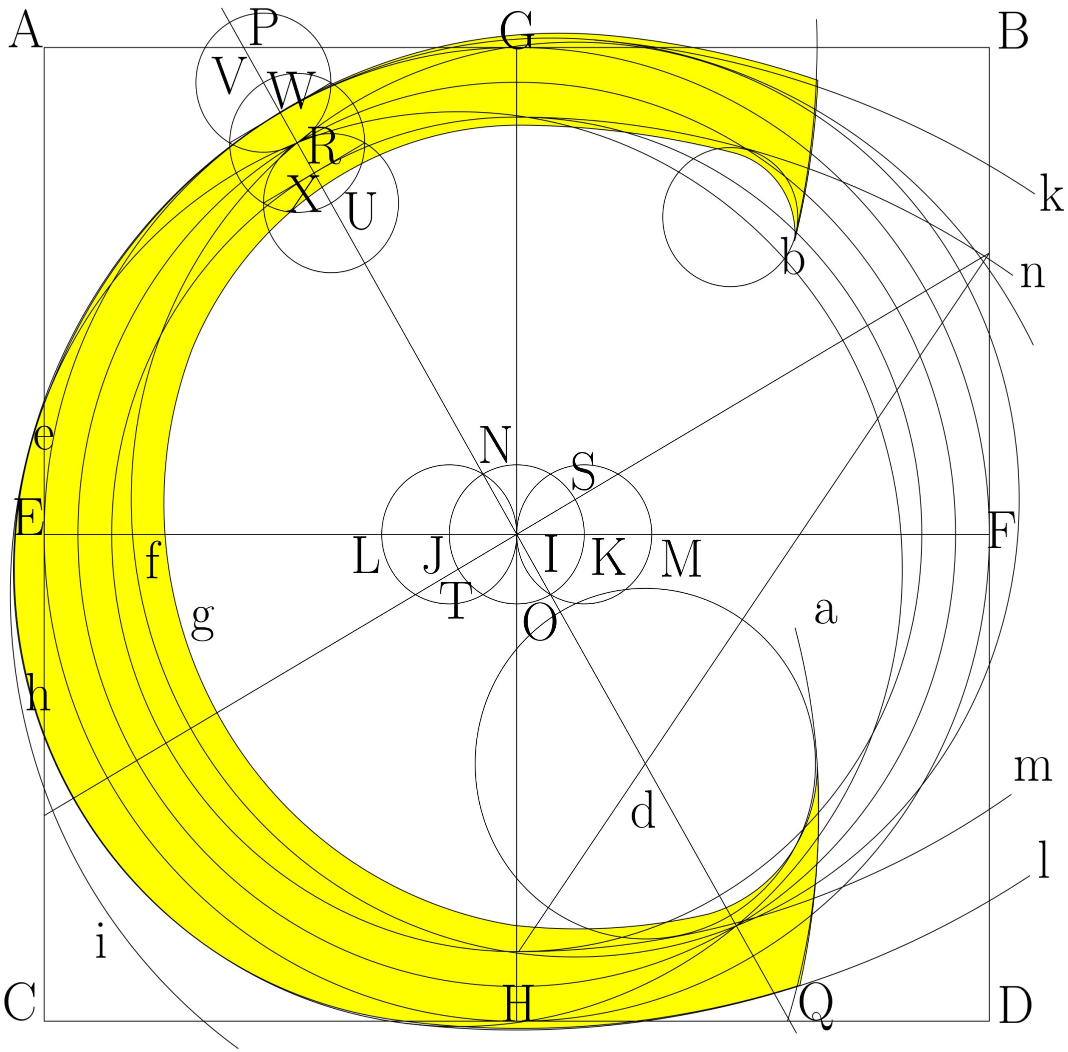
- « *Make sure decisions are taken as if real arithmetic is used* »
  - Theoretical proofs made in the Real-RAM model therefore apply

Exact Geometric Computing

For the simple but common cases of rational expressions

- Use *Multiple Precision* instead of FP
  - Works, but can be >100 times slower

- Increase precision as needed (filtering)
  - Try *Interval Arithmetic* before MP
    - 10 times slower than FP
  - Try *Static Error Analysis* before IA
    - <2 times slower than FP



Arithmetic toolbox

Multiple precision

- Integers, Dyadic (m.2ˆe), Rationals

- Implementations
  - LEDA, GMP
  - CGAL's own MP_Float, Quotient, Gmpzf

- MP_Float
  - Dyadic using signed limbs (no sign bit)
  - Quick implementation (O(n^2) mult...)

Interval Arithmetic

- Double precision mostly
- CGAL's own Interval_nt
  - For portability, integrability, efficiency
  - Decreasing number of rounding mode changes
- Gmpfi too (intervals based on MPFR)

Static Error Analysis

- For predicates that are signs of polynomial expressions, like orientation()
- Start from a bound on the input (known or computed), quickly deduce the maximum error on the final value

- Tradeoffs precision/speed
- Automatization (FPG, C++ ?)
- Formal proofs (COQ+Gappa)

Algebraic numbers

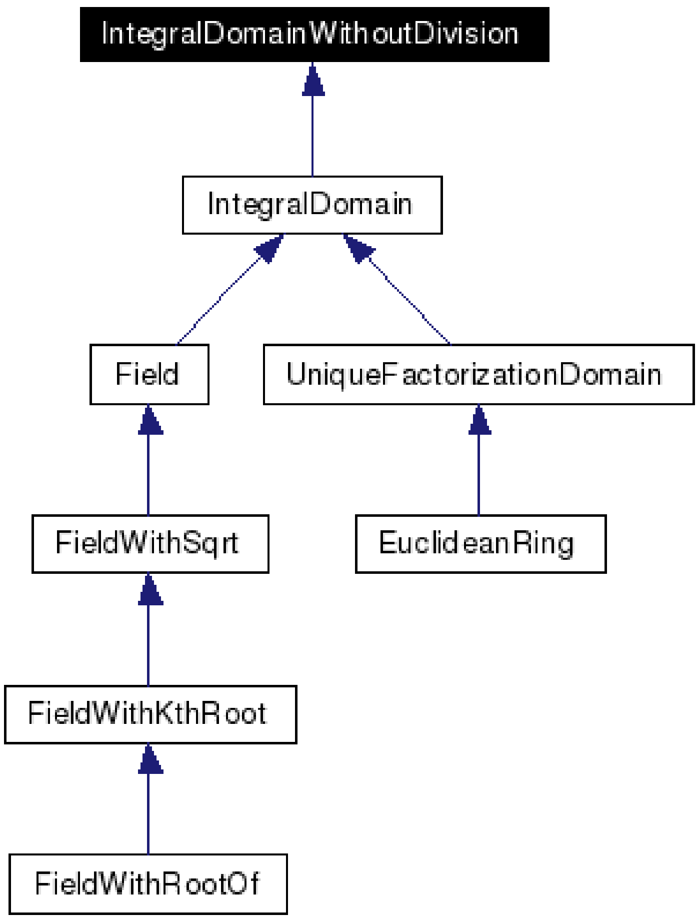
- Example : sign(sqrt(2)^2-2) ?
- Repeated refined numeric approximation with separation bounds [LEDA, CORE]
- More demanding problems : algebraic methods for polynomial systems (not very functional in CGAL) [RS]
- Root_of_2

Side note : Modular arithmetic

- Used for quick non-zero testing

Generic programming

- Layers : Arithmetic, Algebra, Geometry



Concrete classes

- Number Types
  - `double, float`
  - `CGAL::Gmpq or leda::rational (rational)`
  - `Core or leda::real (algebraic)`
  - `CGAL::Lazy_exact_nt<ExactNT>`
- Geometry Kernels
  - `CGAL::Simple_cartesian<NT>`
  - `CGAL::Filtered_kernel<Kernel>`
  - `CGAL::Lazy_kernel<NT>`
- Convenience Kernels
  - `Exact_predicates_inexact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel_with_sqrt`

Filtered predicates

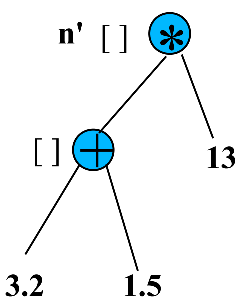
Generic functor adaptor Filtered_predicate<P>

```
try {
    evaluate predicate P<interval arithmetic>;
} catch(UncertaintyException e) {
    evaluate predicate P<exact arithmetic>;
}
```

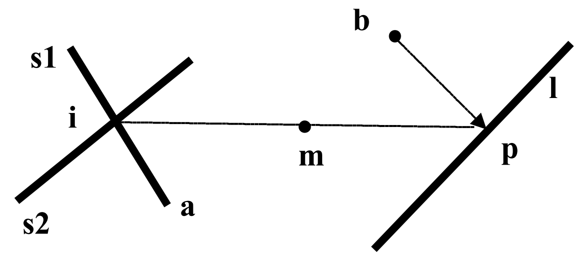
Filtered constructions

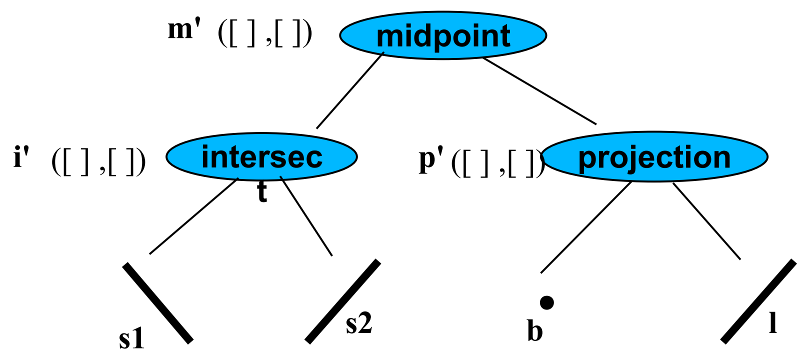
Lazy numbers = interval +
expression tree

$$(3.2 + 1.5) * 13$$



Lazy object = approximate object +
geometric operations tree





Test which can trigger an exact re-evaluation :

if (n' < m')

if (collinear(a',m',b'))

External libraries vs internal

- Reasons/arguments to decide when to use external libraries versus implement internally
  - Availability, portability, ease of installation
  - Efficiency
  - Stability, maturity
  - License, standardizability
  - Inertia, available time/motivation for change, procrastination, whatever...

Conclusion & Open issues

- A mix of FP and EGC works well for most CGAL algorithms

- Some open issues :
  - Depth of cascaded constructions needs to stay reasonable
  - Topology preserving geometric rounding to the FP grid
  - Non-algebraic numbers
  - Static error analysis within C++ (machine precision ball arithmetic?)
  - Formal proofs

Acknowledgements

- Contributors to the CGAL arithmetic toolbox
  - Olivier Devillers, Marc Glisse
  - Michael Hemmer, Luis Peñaranda
  - Monique Teillaud
  - CGAL editors and developers

- Special Thanks
  - Mariette Yvinec, Jean-Daniel Boissonnat
  - Andreas Fabri & GeometryFactory

- Arithmetic libraries : LEDA, GMP, CORE, MPFR, MPFI
- Standards : C++, IEEE-754, IEEE-1788