Efficient Implementation of Exact Geometric Computations in CGAL

Sylvain Pion

INRIA Sophia-Antipolis

October 31, 2006

Plan

Plan

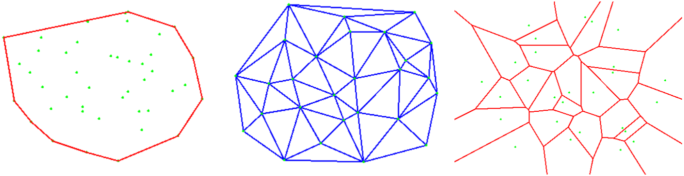
Computational Geometry

- Active research domain since 30 years
- Algorithms handle large number of geometric objects
- Emphasis on asymptotic complexity (Real-RAM model)

Application domains: CAD/CAM, GIS, molecular biology, medical imaging...

Examples

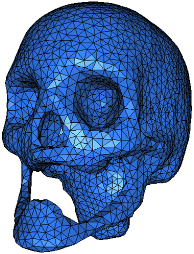
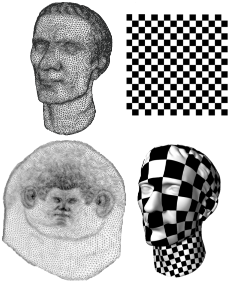
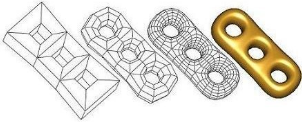
- Convex hulls, triangulations, Voronoi diagrams



- Surface reconstruction, meshing
- Boolean operations on polygons, arrangements
- Geometric optimization
- ...

Examples : applications

- Surface reconstruction and meshing
- Surface parameterization
- Surface subdivision

CGAL: *Computational Geometry Algorithms Library*

Since 1995 : implement Computational Geometry algorithms.

- Criteria : adaptability, efficiency, robustness
- Contributions are reviewed by an Editorial Board
- Chosen language : C++ (generic programming)
- v3.2 : 100 modules, 500.000 code lines, 10,000 downloads/year
- Open Source : LGPL and QPL (commercialized since 2003)

CGAL: *Computational Geometry Algorithms Library*

Since 1995 : implement Computational Geometry algorithms.

- Criteria : adaptability, efficiency, robustness
- Contributions are reviewed by an Editorial Board
- Chosen language : C++ (generic programming)
- v3.2 : 100 modules, 500.000 code lines, 10,000 downloads/year
- Open Source : LGPL and QPL (commercialized since 2003)

CGAL: *Computational Geometry Algorithms Library*

Since 1995 : implement Computational Geometry algorithms.

- Criteria : adaptability, efficiency, robustness
- Contributions are reviewed by an Editorial Board
- Chosen language : C++ (generic programming)
- v3.2 : 100 modules, 500.000 code lines, 10,000 downloads/year
- Open Source : LGPL and QPL (commercialized since 2003)

CGAL: *Computational Geometry Algorithms Library*

Since 1995 : implement Computational Geometry algorithms.

- Criteria : adaptability, efficiency, robustness
- Contributions are reviewed by an Editorial Board
- Chosen language : C++ (generic programming)
- v3.2 : 100 modules, 500.000 code lines, 10,000 downloads/year
- Open Source : LGPL and QPL (commercialized since 2003)

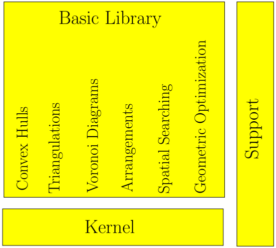
CGAL: *Computational Geometry Algorithms Library*

Since 1995 : implement Computational Geometry algorithms.

- Criteria : adaptability, efficiency, robustness
- Contributions are reviewed by an Editorial Board
- Chosen language : C++ (generic programming)
- v3.2 : 100 modules, 500.000 code lines, 10,000 downloads/year
- Open Source : LGPL and QPL (commercialized since 2003)

CGAL: Architecture

General architecture : kernel, basic library, support library



Kernel of geometric primitives

Algorithms are logically split in :

- a combinatorial part (graph building)
- a numerical part (needs coordinates)

The later calls primitives gathered in the *kernel* :

- Basic objects: points, segments, lines, circles...
- Predicates: orientations, coordinate comparisons...
- Constructions: intersection and distance computations...

Kernel of geometric primitives

Algorithms are logically split in :

- a combinatorial part (graph building)
- a numerical part (needs coordinates)

The later calls primitives gathered in the *kernel* :

- Basic objects: points, segments, lines, circles...
- Predicates: orientations, coordinate comparisons...
- Constructions: intersection and distance computations...

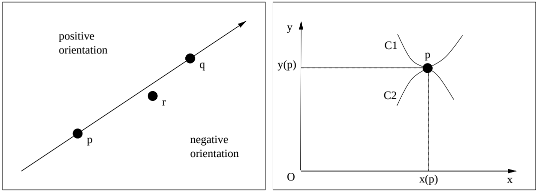
Kernel of geometric primitives

Algorithms are logically split in :

- a combinatorial part (graph building)
- a numerical part (needs coordinates)

The later calls primitives gathered in the *kernel* :

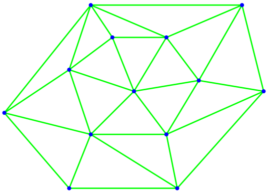
- Basic objects: points, segments, lines, circles...
- Predicates: orientations, coordinate comparisons...
- Constructions: intersection and distance computations...



Plan

Delaunay triangulation

Incremental algorithm in 2 stages: point location and update.

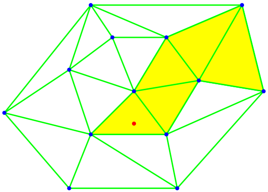


Point location: orientation(p, q, r) predicate, sign of:

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx - px & qy - py \\ rx - px & ry - py \end{vmatrix}$$

Delaunay triangulation

Incremental algorithm in 2 stages: point location and update.

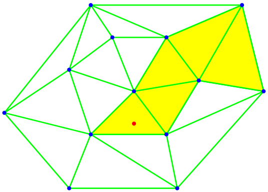


Point location: orientation(p, q, r) predicate, sign of:

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx-px & qy-py \\ rx-px & ry-py \end{vmatrix}$$

Delaunay triangulation

Incremental algorithm in 2 stages: point location and update.

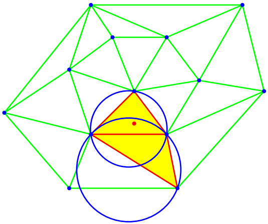


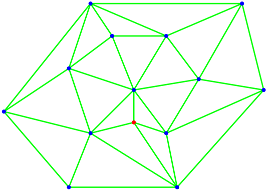
Point location: orientation(p, q, r) predicate, sign of:

$$\begin{vmatrix} 1 & px & py \\ 1 & qx & qy \\ 1 & rx & ry \end{vmatrix} = \begin{vmatrix} qx - px & qy - py \\ rx - px & ry - py \end{vmatrix}$$

Delaunay triangulation

Incremental algorithm in 2 stages: point location and update.



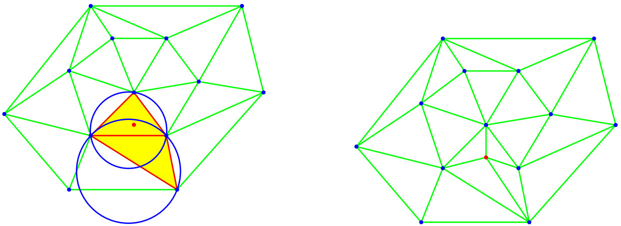


Update: in_circle(p, q, r, s) predicate, sign of:

$$\begin{vmatrix} 1 & px & py & px^2+py^2 \\ 1 & qx & qy & qx^2+qy^2 \\ 1 & rx & ry & rx^2+ry^2 \\ 1 & sx & sy & sx^2+sy^2 \end{vmatrix}$$

Delaunay triangulation

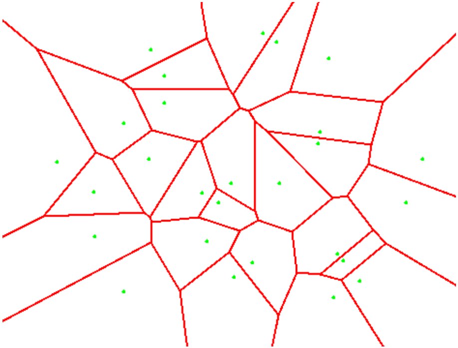
Incremental algorithm in 2 stages: point location and update.



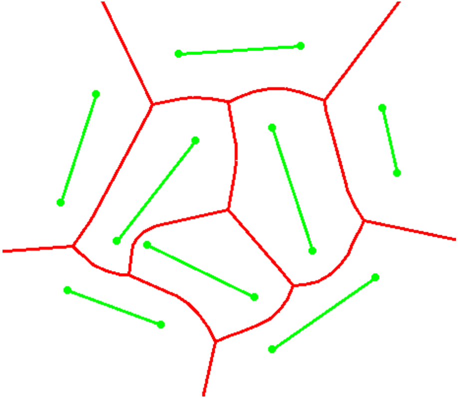
Update: in_circle(p, q, r, s) predicate, sign of:

$$\begin{vmatrix} 1 & px & py & px^2+py^2 \\ 1 & qx & qy & qx^2+qy^2 \\ 1 & rx & ry & rx^2+ry^2 \\ 1 & sx & sy & sx^2+sy^2 \end{vmatrix}$$

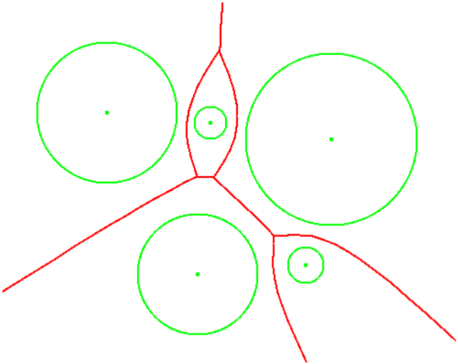
Voronoi diagramms of points



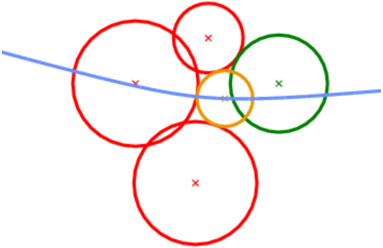
Voronoi diagrams of segments



Voronoi diagrams of circles

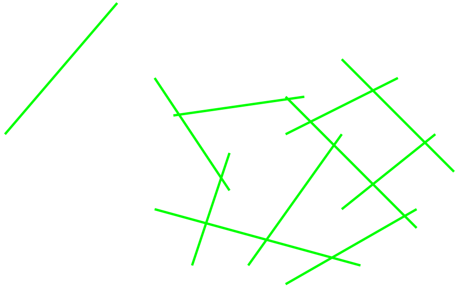


One of the predicates of the Voronoi diagram of circles

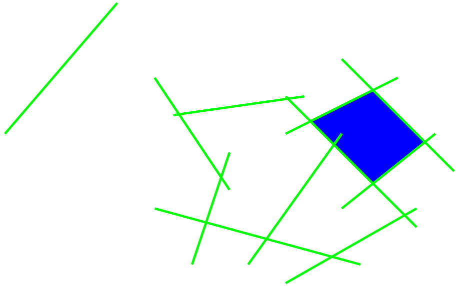


Root comparison techniques [Karavelas, Emiris: SODA'03]

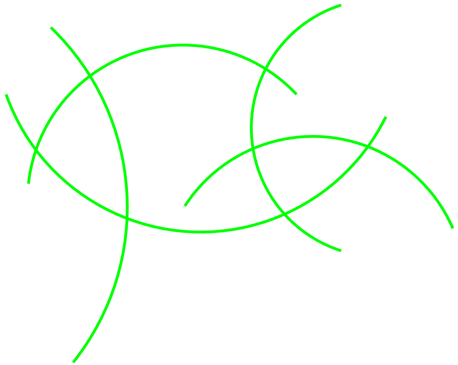
Arrangements of line segments



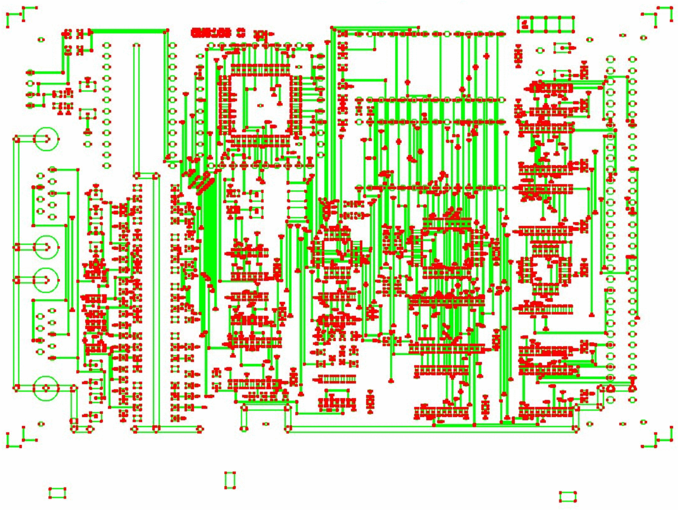
Arrangements of line segments



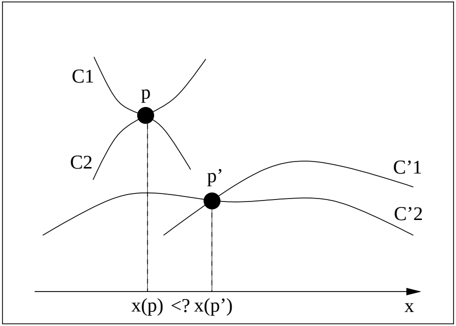
Arrangements of circular arcs



Application: union of polygons in VLSI



Comparison of abscissa of curve intersections

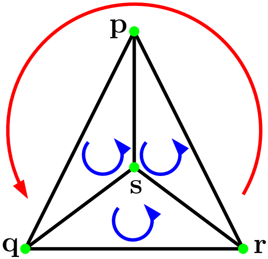


Algebraic curves, comparisons of algebraic numbers

Plan

Robustness

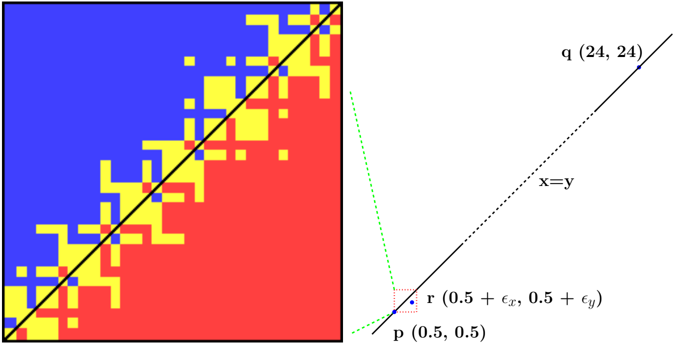
Algorithms rely on mathematic theorems, like:



ccw(s, q, r)
ccw(p, s, r) => ccw(p, q, r)
ccw(p, q, s)

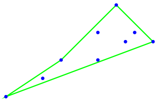
Robustness

Example where floating-point geometry differs from real geometry: orientation of almost collinear points.

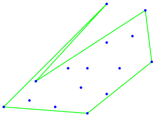


[Kettner, Mehlhorn, Schirra, P., Yap, ESA'04]

Possible consequences on the algorithms

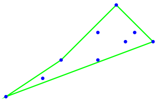


- The result can be slightly off

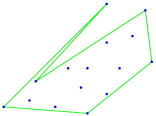


- The result can be completely off
- The algorithm stops because of an unexpected impossible state
- The algorithm loops forever

Possible consequences on the algorithms



- The result can be slightly off



- The result can be completely off
- The algorithm stops because of an unexpected impossible state
- The algorithm loops forever

Robustness: solutions

- Case by case handling : painful, error prone and not mathematically nice
- Use exact predicates (*Exact Geometric Computing*)

Remarks

- Floating-point computing fails on [nearly] degenerate cases.
- These cases happen often in practice.

Robustness: solutions

- Case by case handling : painful, error prone and not mathematically nice
- Use exact predicates (*Exact Geometric Computing*)

Remarks

- Floating-point computing fails on [nearly] degenerate cases.
- These cases happen often in practice.

Robustness: solutions

- Case by case handling : painful, error prone and not mathematically nice
- Use exact predicates (*Exact Geometric Computing*)

Remarks

- Floating-point computing fails on [nearly] degenerate cases.
- These cases happen often in practice.

Plan

Number types

Geometric primitives are parameterized by the arithmetic.

- Multi-precision integers [GMP, MPFR, LEDA...]
- Multi-precision rationals
- Multi-precision floating-point
- Interval arithmetic (single or multi-precision bounds)

Algebraic numbers:

- Numeric evaluation with separation bounds [CORE, LEDA]
- Polynomials, Sturm sequences, resultants... [CGAL, CORE, SYNAPS]

Number types

Geometric primitives are parameterized by the arithmetic.

- Multi-precision integers [GMP, MPFR, LEDA...]
- Multi-precision rationals
- Multi-precision floating-point
- Interval arithmetic (single or multi-precision bounds)

Algebraic numbers:

- Numeric evaluation with separation bounds [CORE, LEDA]
- Polynomials, Sturm sequences, resultants... [CGAL, CORE, SYNAPS]

Number types

Geometric primitives are parameterized by the arithmetic.

- Multi-precision integers [GMP, MPFR, LEDA...]
- Multi-precision rationals
- Multi-precision floating-point
- Interval arithmetic (single or multi-precision bounds)

Algebraic numbers:

- Numeric evaluation with separation bounds [CORE, LEDA]
- Polynomials, Sturm sequences, resultants... [CGAL, CORE, SYNAPS]

Number types

Geometric primitives are parameterized by the arithmetic.

- Multi-precision integers [GMP, MPFR, LEDA...]
- Multi-precision rationals
- Multi-precision floating-point
- Interval arithmetic (single or multi-precision bounds)

Algebraic numbers:

- Numeric evaluation with separation bounds [CORE, LEDA]
- Polynomials, Sturm sequences, resultants... [CGAL, CORE, SYNAPS]

Generic programming

Parameterization using templates.

```
template < class T >
T min (T a, T b)
{
  if (a < b)
    return a;
  else
    return b;
}

...

min(1, 2);       // instantiates min() with T = int.
min(1.0, 2.0);  // instantiates min() with T = double.
```

Generic programming in CGAL

Several levels of parameterization :

- Algorithms parameterized by the geometry (kernel)

```
template < class Traits >
class Triangulation_3;
```

- Kernels parameterized by the arithmetic (number types)

```
template < class T >
class Cartesian;
```

Plugging the 2 layers:

```
typedef CGAL::Cartesian<double>         Kernel;
typedef CGAL::Triangulation_3<Kernel>  Triangulation_3;
```

Generic programming in CGAL

Several levels of parameterization :

- Algorithms parameterized by the geometry (kernel)

```
template < class Traits >
class Triangulation_3;
```

- Kernels parameterized by the arithmetic (number types)

```
template < class T >
class Cartesian;
```

Plugging the 2 layers:

```
typedef CGAL::Cartesian<double>          Kernel;
typedef CGAL::Triangulation_3<Kernel>  Triangulation_3;
```

Generic programming in CGAL

Several levels of parameterization :

- Algorithms parameterized by the geometry (kernel)

```
template < class Traits >
class Triangulation_3;
```

- Kernels parameterized by the arithmetic (number types)

```
template < class T >
class Cartesian;
```

Plugging the 2 layers:

```
typedef CGAL::Cartesian<double>         Kernel;
typedef CGAL::Triangulation_3<Kernel>   Triangulation_3;
```

Filtered predicates

Speed-up exact predicates using a filter:

- floating-point evaluation with a certificate
- multi-precision arithmetic only when needed

Examples

- interval arithmetic (dynamic filters), [Burnikel, Funke, Seel – Brönnimann, Burnikel, P.'98]
- or code analysis (static filters) [Fortune'93... Melquiond, P.'05]

Implementation issues:

- automatic generation of filtered predicates
- cascading several methods

Filtered predicates : generic implementation

Predicates as generic functors:

```
template <class Kernel>
class Orientation_2
{
  typedef Kernel::Point_2   Point_2;
  typedef Kernel::FT        Number_type;

  Sign
  operator()(Point_2 p, Point_2 q, Point_2 r) const
  {
    return ...;
  }
};
```

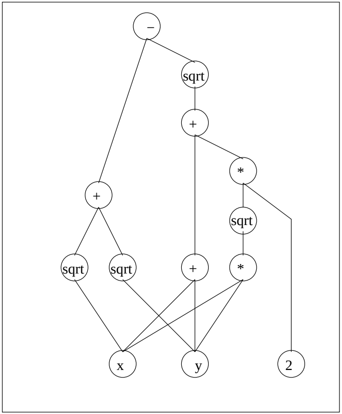
Filtered predicates : generic implementation

```
template <class EP, class AP, class C2E, class C2A>
class Filtered_predicate
{
  AP   approx_predicate;   C2A  c2a;
  EP   exact_predicate;    C2E  c2e;

  typedef EP::result_type  result_type;

  template <class A1, class A2>
  result_type
  operator()(A1 a1, A2 a2) const
  {
    try {
      return approx_predicate(c2a(a1), c2a(a2));
    } catch (Interval::unsafe_comparison) {
      return exact_predicate(c2e(a1), c2e(a2));
    }
  }
};
```

Something similar is done for constructions (harder) [P., Fabri'06]

Filtered number types

Directed Acyclic Graph (DAG) of operations in memory. Ex:
$\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}}$



Approximation and iterative precision refinement, on demand.

Filtered predicates: comparisons

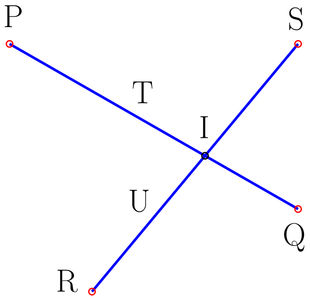
Computation time of a 3D Delaunay triangulation.

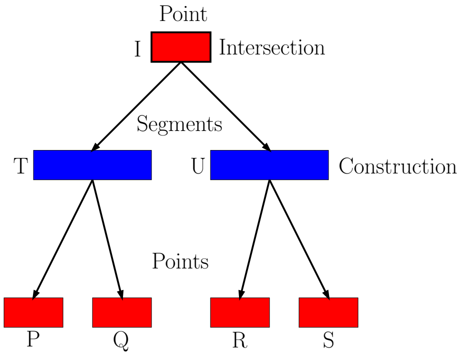
	R5	E	M	B	D
double	40.6	41.0	43.7	50.3	`loops`
MPF	3,063	2,777	3,195	3,472	214
Interval + MPF	137.2	133.6	144.6	165.1	15.8
semi static + Interval + MPF	51.8	61.0	59.1	93.1	8.9
almost static + semi static + Interval + MPF	44.4	55.0	52.0	87.2	8.0
Shewchuk's predicates	57.9	57.5	62.8	71.7	7.2
CORE Expr	570	3520	1355	9600	173
LEDA real	682	640	742	850	125
`Lazy_exact_nt<MPF>`	705	631	726	820	67

Important criterium: failure rate of filters.
User interface in CGAL: choice of different kernels.

Filtered constructions

Additional difficulty: memory storage of geometric objects
Goal: regrouping computations, and less memory





Filtered constructions : benchmarks

Generate 2000 random segments, intersect them, compute all orientations of consecutive intersection points.

Kernel	time g++ 4.1	memory
`SC<Gmpq>`	70	70
`SC<Lazy_exact_nt<Gmpq>>`	7.4	501
`Lazy_kernel<SC<Gmpq>>` (2)	3.6	64
`Lazy_kernel<SC<Gmpq>>`	2.8	64
`SC<double>`	0.72	8.3

Plan

Implementation of EGC

- WIP : Efficient treatment of curved objects of low degree
- WIP : Improvement of the treatment of geometric constructions
- WIP : Geometric rounding with guarantees
- ...

Questions ?