Design Goals and Programming Paradigms

Sylvain Pion

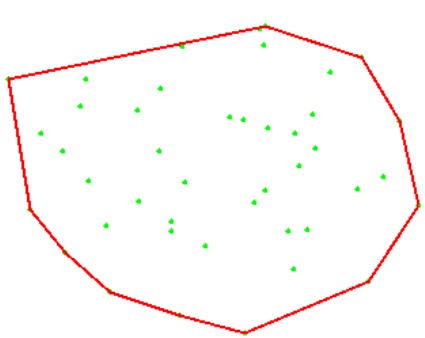
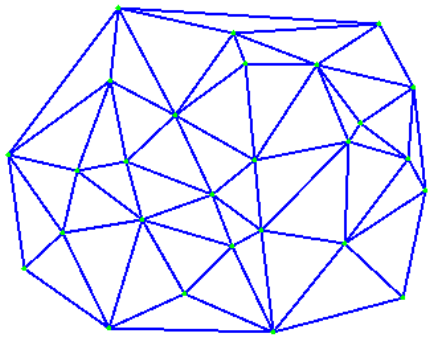
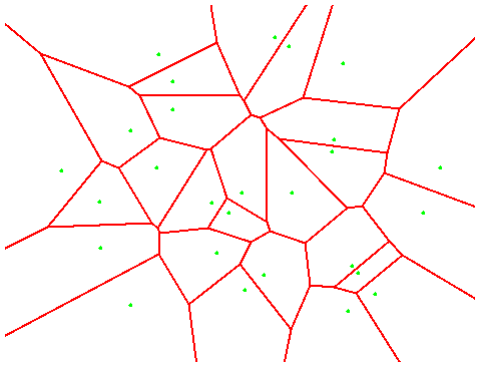
INRIA Sophia Antipolis, France 

LIAMA 10th Anniversary, 2007

Outline

Examples of Computational Geometry Algorithms

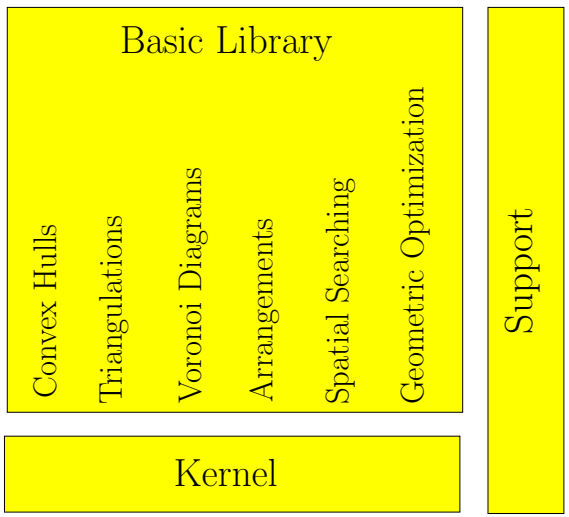
- Convex hulls, triangulations, Voronoi diagrams

- Emphasis on asymptotic complexity (Real-RAM model)
- Algorithms manipulating large numbers of geometric objects

CGAL: Software Architecture

General architecture: kernel, basic library, support library



Kernel of geometric primitives

Algorithms are logically split into:

- a combinatorial part (constructs a graph)
- a numerical part (makes use of coordinates)

The latter calls primitives gathered in the kernel:

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

Kernel of geometric primitives

Algorithms are logically split into:

- a combinatorial part (constructs a graph)
- a numerical part (makes use of coordinates)

The latter calls primitives gathered in the kernel:

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

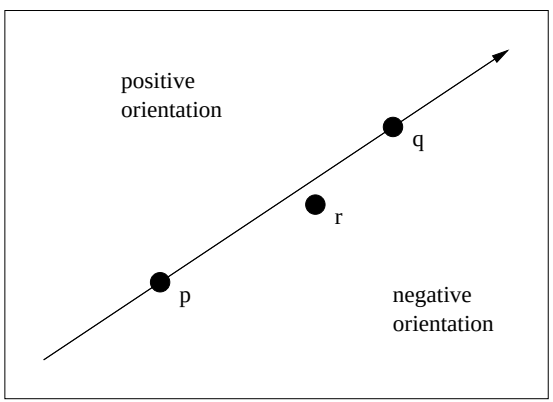
Kernel of geometric primitives

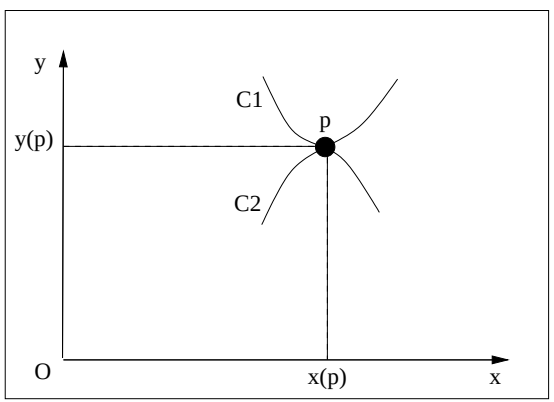
Algorithms are logically split into:

- a combinatorial part (constructs a graph)
- a numerical part (makes use of coordinates)

The latter calls primitives gathered in the kernel:

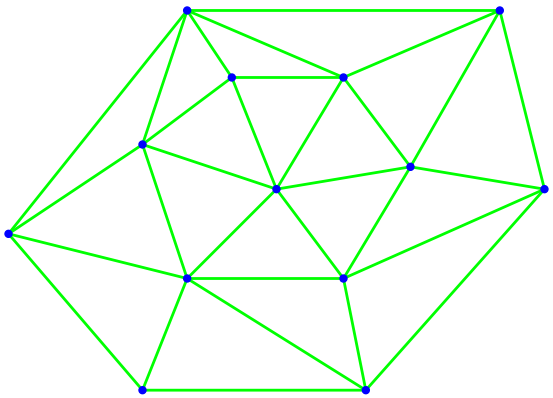
- Basic objects: points, segments, lines, circles...
- Predicates: orientations, abscissa comparisons...
- Constructions: intersection, distance computations...





Example: Delaunay triangulation

Incremental algorithm in 2 steps: point location and update.

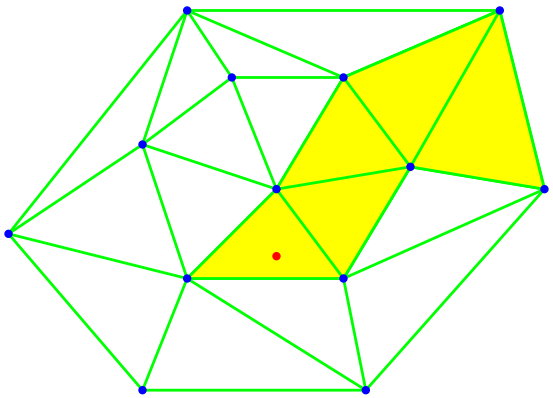


Point location: orientation(p, q, r) predicate, sign of:

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

Example: Delaunay triangulation

Incremental algorithm in 2 steps: point location and update.

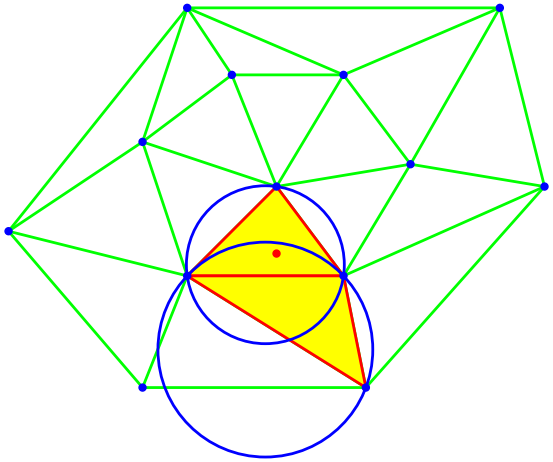


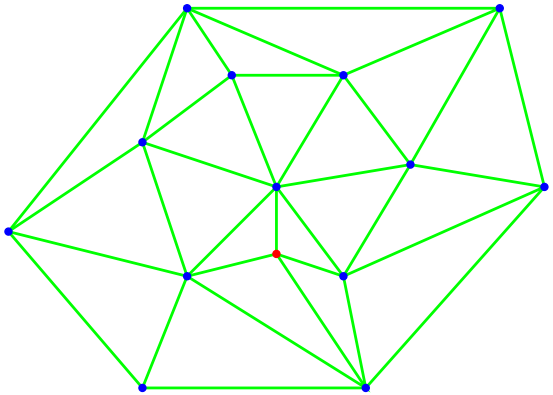
Point location: orientation(p, q, r) predicate, sign of:

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

Delaunay triangulation

Incremental algorithm in 2 steps: point location and update.



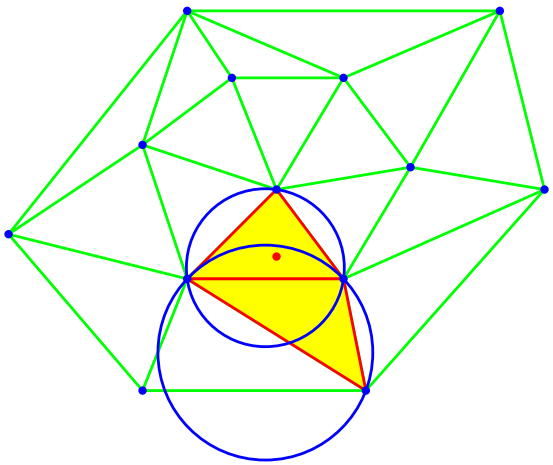


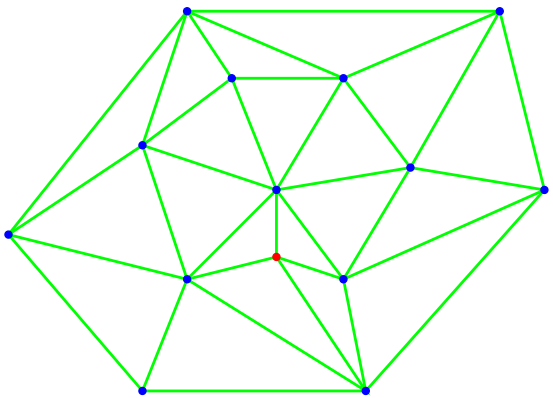
Update: in_circle(p, q, r, s) predicate, sign of:

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

Delaunay triangulation

Incremental algorithm in 2 steps: point location and update.

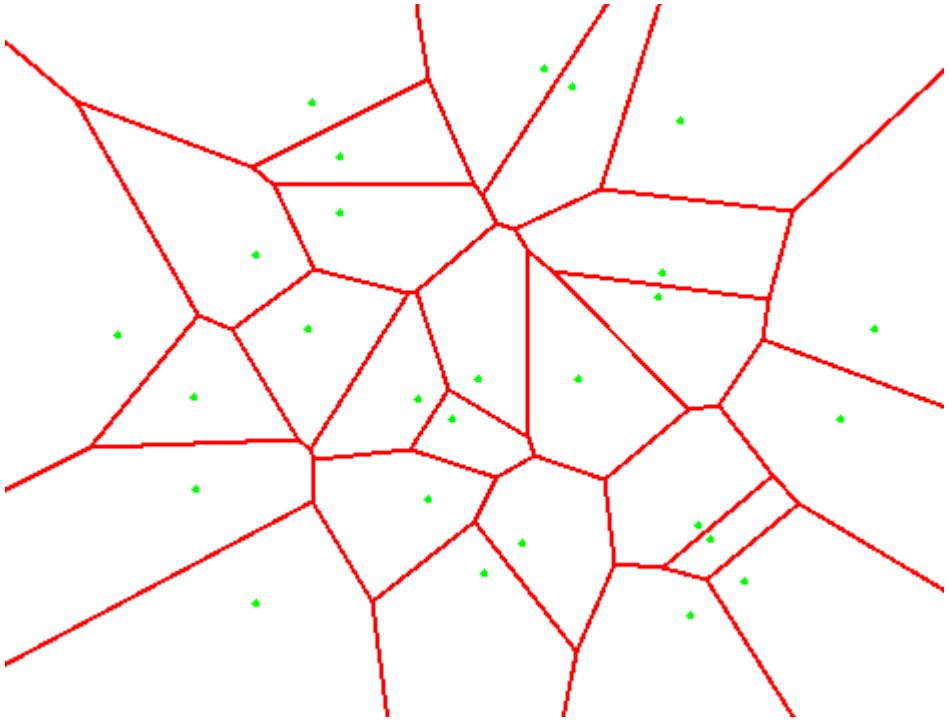




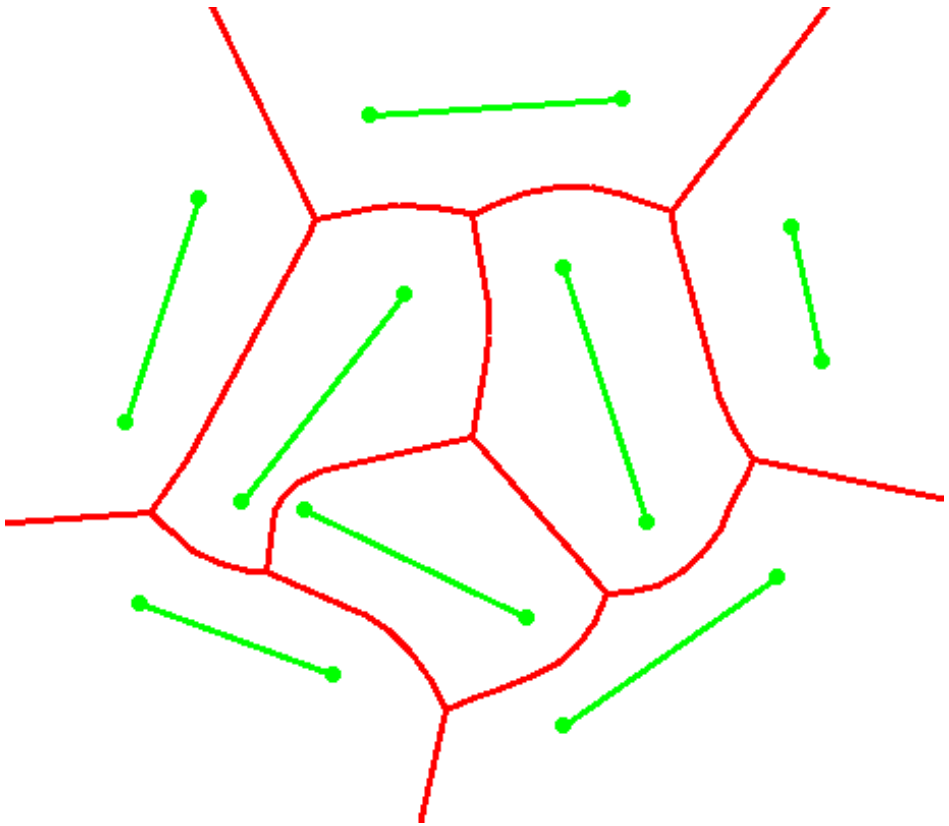
Update: in_circle(p, q, r, s) predicate, sign of:

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

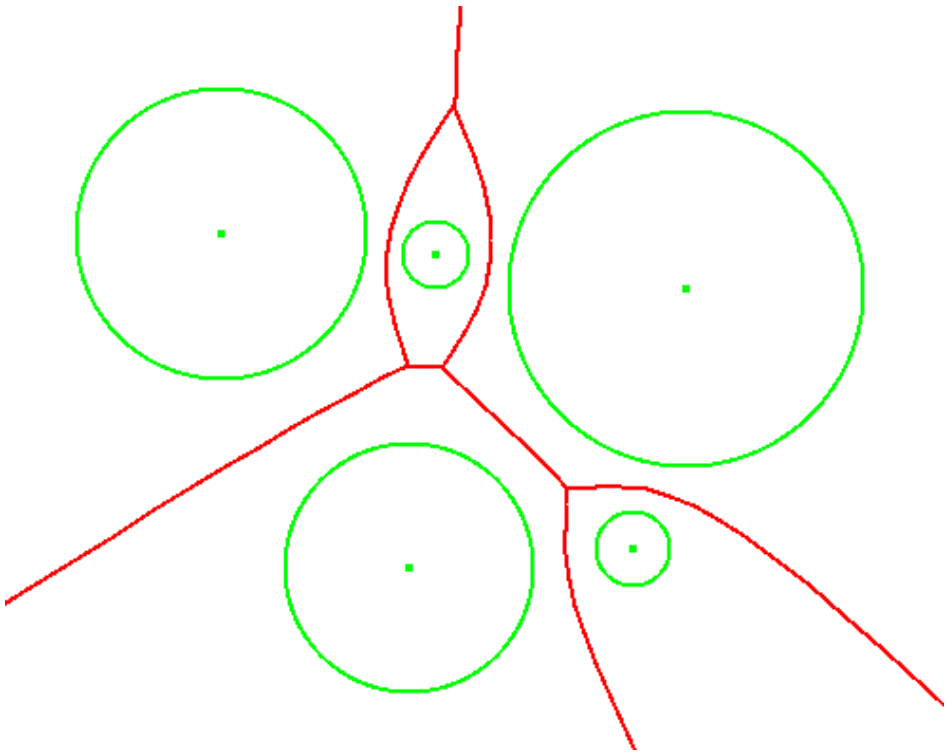
Voronoi diagram of points



Voronoi diagram of line segments

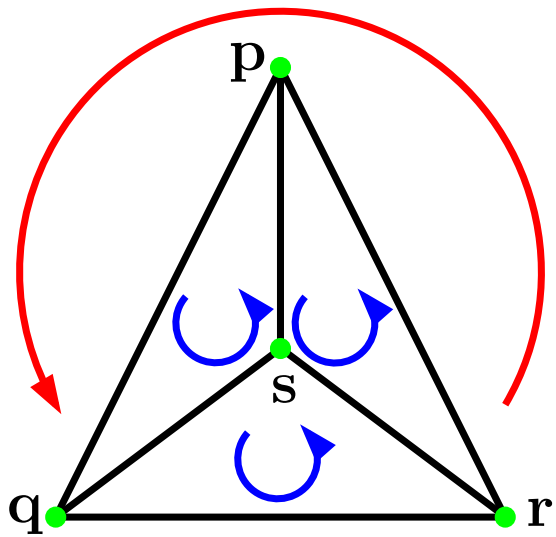


Voronoi diagram of circles



Robustness problems

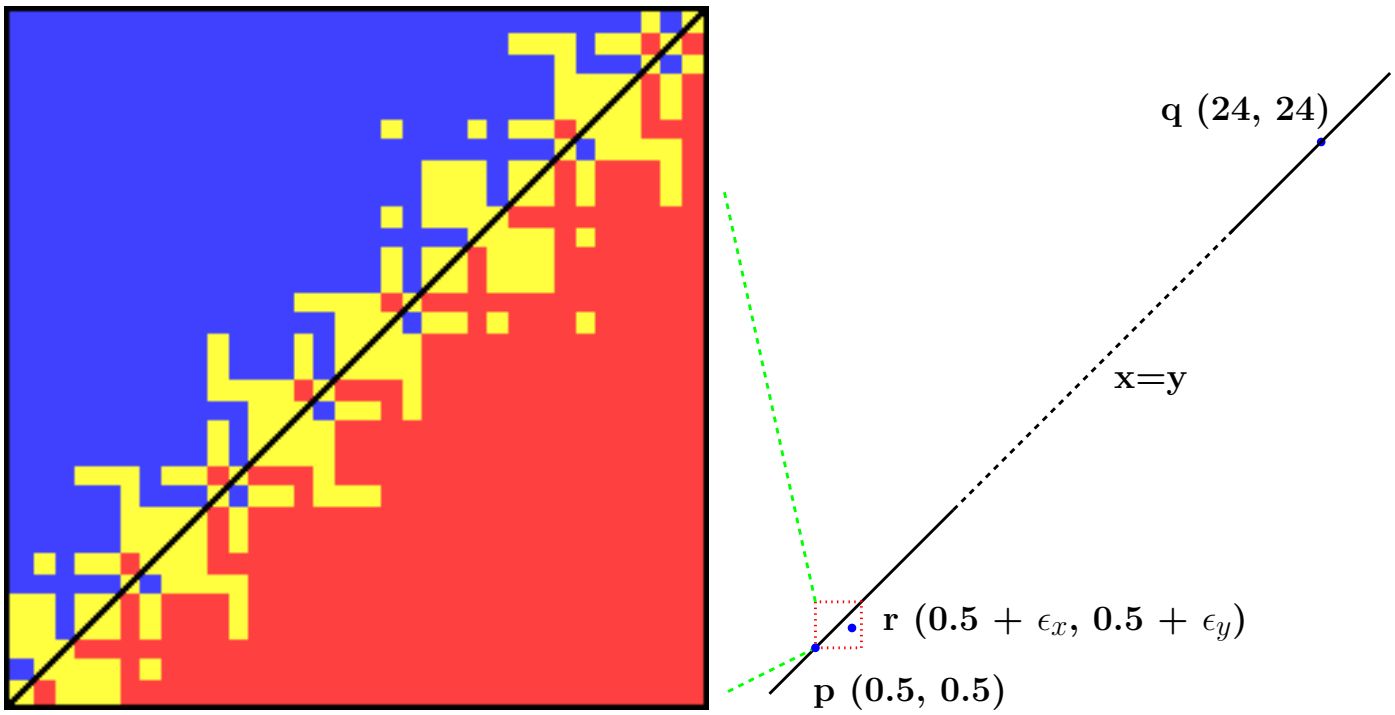
Algorithms rely on mathematical theorems, like:



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

Robustness

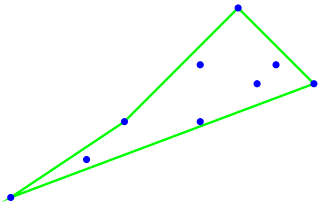
Example where floating-point geometry differs from real geometry:
orientation of almost collinear points.

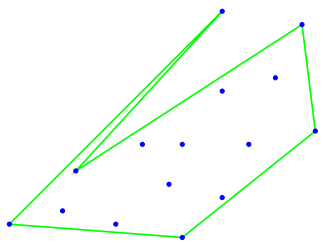


[Kettner, Mehlhorn, Schirra, P., Yap, ESA'04]

Potential consequences on algorithms

- Slightly wrong result

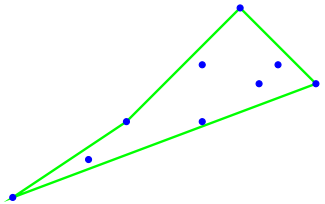


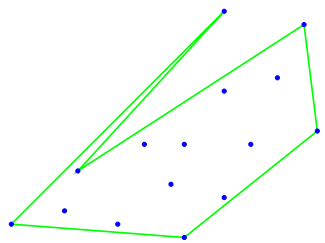


- Completely wrong result
- The algorithm stops on a invalid assertion
- The algorithm loops forever

Potential consequences on algorithms

- Slightly wrong result





- Completely wrong result
- The algorithm stops on a invalid assertion
- The algorithm loops forever

Robustness: solutions

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

Remarks

- Floating-point fails on [almost] degenerate cases.
- These cases happen more of less often in practice.

Robustness: solutions

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

Remarks

- Floating-point fails on [almost] degenerate cases.
- These cases happen more of less often in practice.

Robustness: solutions

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

Remarks

- Floating-point fails on [almost] degenerate cases.
- These cases happen more of less often in practice.

Arithmetic: Number types

Geometric primitives are parameterized by the arithmetic.

- Multiple precision integers [GMP, MPFR, LEDA...]
- Multiple precision rationals
- Multiple precision floating-point
- Interval arithmetic (with `double` or MP bounds)

Algebraic numbers:

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

Arithmetic: Number types

Geometric primitives are parameterized by the arithmetic.

- Multiple precision integers [GMP, MPFR, LEDA...]
- Multiple precision rationals
- Multiple precision floating-point
- Interval arithmetic (with `double` or MP bounds)

Algebraic numbers:

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

Arithmetic: Number types

Geometric primitives are parameterized by the arithmetic.

- Multiple precision integers [GMP, MPFR, LEDA...]
- Multiple precision rationals
- Multiple precision floating-point
- Interval arithmetic (with `double` or MP bounds)

Algebraic numbers:

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

Arithmetic: Number types

Geometric primitives are parameterized by the arithmetic.

- Multiple precision integers [GMP, MPFR, LEDA...]
- Multiple precision rationals
- Multiple precision floating-point
- Interval arithmetic (with `double` or MP bounds)

Algebraic numbers:

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

Filtered predicates

Speed up exact predicates using a filter:

- floating-point evaluation with a certificate
- multiple precision only when needed

Examples

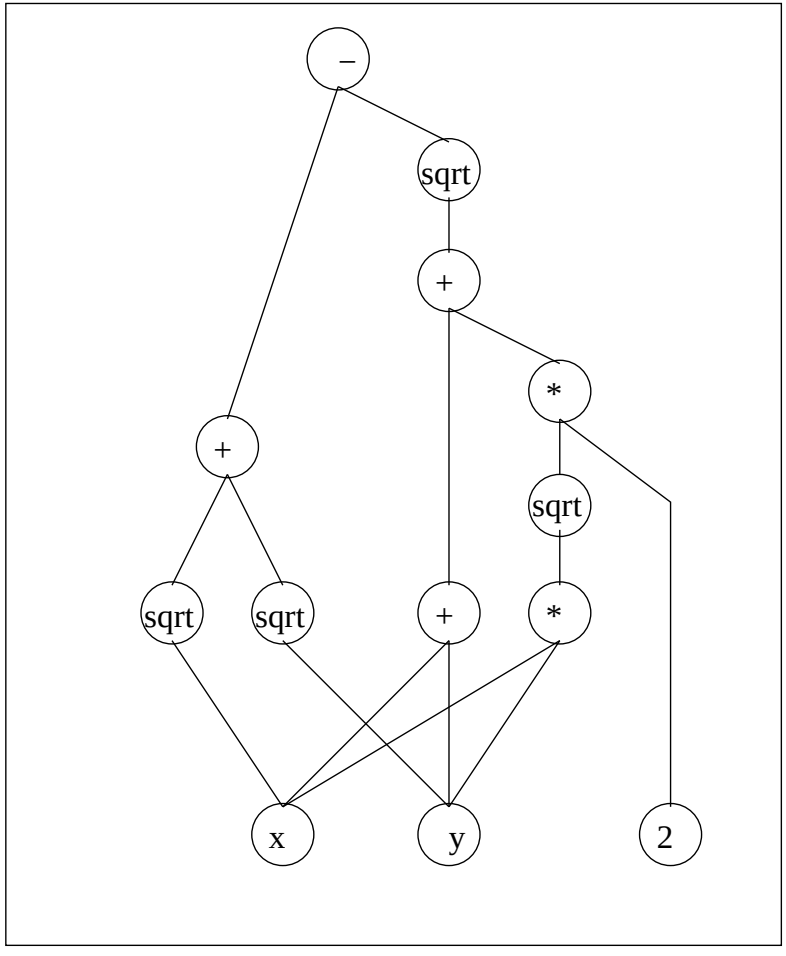
- interval arithmetic (dynamic filters), [Burnikel, Funke, Seel – Brönnimann, Burnikel, P.'98]
- or static code analysis (static filters) [Fortune'93... Melquiond, P.'05]

Programming aspects:

- automatic generation of filtered predicates
- cascading/pipelining various methods

Filtered number types

DAG of the operations in memory, e.g.: $\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}}$



Approximation and iterative precision refinement, on demand.

Filtered predicates: benchmarks

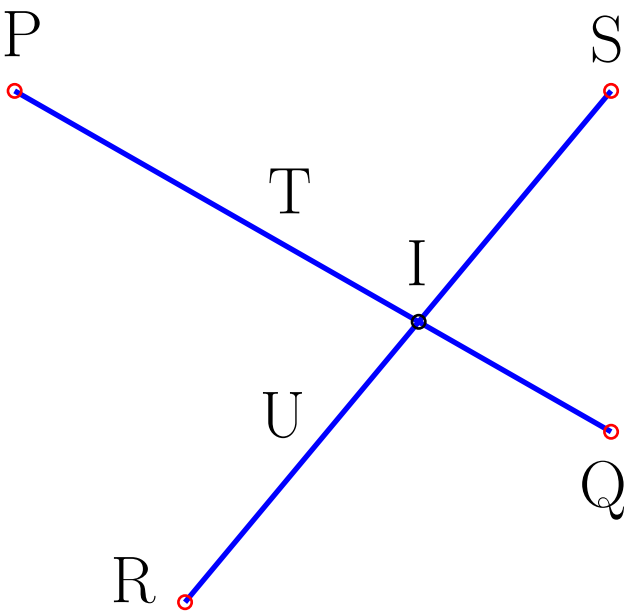
Running time of a 3D Delaunay triangulation.

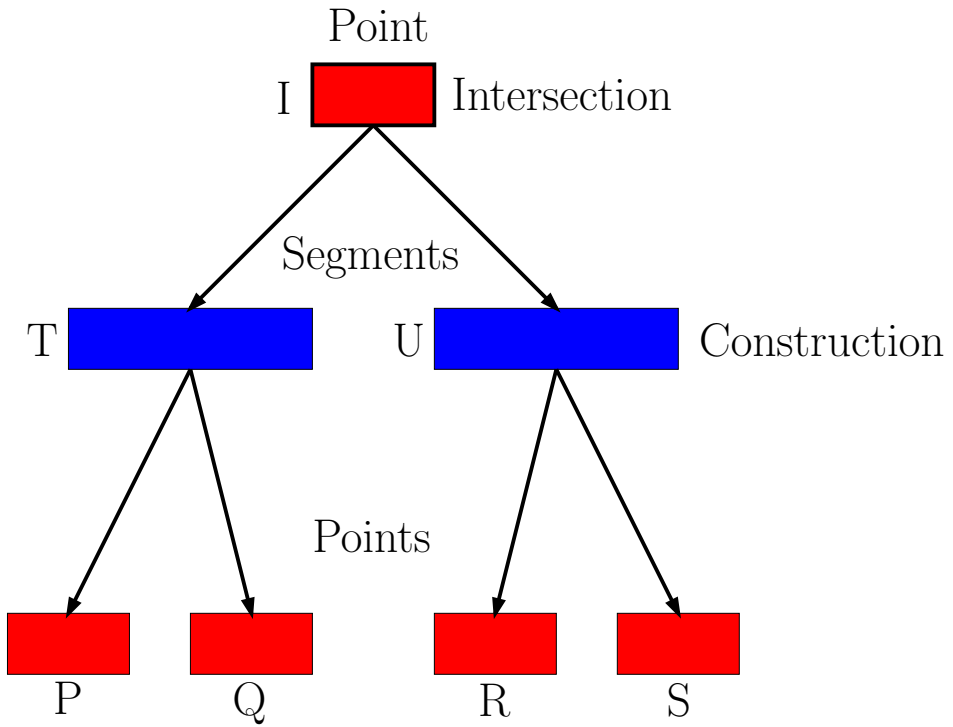
	R5	E	M	B	D
double	40.6	41.0	43.7	50.3	`loops`
MPF	3,063	2,777	3,195	3,472	214
Interval + MPF	137.2	133.6	144.6	165.1	15.8
semi static + Interval + MPF	51.8	61.0	59.1	93.1	8.9
almost static + semi static + Interval + MPF	44.4	55.0	52.0	87.2	8.0
Shewchuk's predicates	57.9	57.5	62.8	71.7	7.2
CORE Expr	570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

An important criteria: failure rate of filters.
User interface in CGAL: choice of different kernels.

Filtered geometric constructions

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





Algorithms and traits classes

Generic programming: algorithms are decoupled from the data structures and the numerics. Similarly to the C++ STL.

Algorithms are parameterized (templates) by geometric traits classes which provide:

- types of the objets manipulated by the algorithm: Point_2, Tetrahedron_3...
- predicates that the algorithm applies to objets: Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

These last two are provided under the form of function objects.
Requirements of algorithms are described under the form of concepts for their template parameters.

Algorithms and traits classes

Generic programming: algorithms are decoupled from the data structures and the numerics. Similarly to the C++ STL.

Algorithms are parameterized (templates) by geometric traits classes which provide:

- types of the objets manipulated by the algorithm: Point_2, Tetrahedron_3...
- predicates that the algorithm applies to objets: Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

These last two are provided under the form of function objects.
Requirements of algorithms are described under the form of concepts for their template parameters.

Algorithms and traits classes

Generic programming: algorithms are decoupled from the data structures and the numerics. Similarly to the C++ STL.

Algorithms are parameterized (templates) by geometric traits classes which provide:

- types of the objets manipulated by the algorithm: Point_2, Tetrahedron_3...
- predicates that the algorithm applies to objets: Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

These last two are provided under the form of function objects.
Requirements of algorithms are described under the form of concepts for their template parameters.

Algorithms and traits classes

Generic programming: algorithms are decoupled from the data structures and the numerics. Similarly to the C++ STL.

Algorithms are parameterized (templates) by geometric traits classes which provide:

- types of the objets manipulated by the algorithm: Point_2, Tetrahedron_3...
- predicates that the algorithm applies to objets: Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

These last two are provided under the form of function objects.

Requirements of algorithms are described under the form of concepts for their template parameters.

Algorithms and traits classes

Generic programming: algorithms are decoupled from the data structures and the numerics. Similarly to the C++ STL.

Algorithms are parameterized (templates) by geometric traits classes which provide:

- types of the objets manipulated by the algorithm: Point_2, Tetrahedron_3...
- predicates that the algorithm applies to objets: Orientation_2, Side_of_oriented_sphere_3...
- constructions : Construct_mid_point_2, Construct_circumcenter_3, Compute_squared_length_2...

These last two are provided under the form of function objects.
Requirements of algorithms are described under the form of concepts for their template parameters.

Kernel interface

CGAL geometric algorithms are parameterized by a geometric traits.

```
template < class TriangulationTraits_3 ,
           class TriangulationDataStructure_3 = ... >
class Triangulation_3 ;
```

It provides a (attempted minimal) set of types and functions for the geometry :

Nested Type	Requirements
Point_3	Assignable, DefaultConstructible
Segment_3	Assignable, DefaultConstructible
Orientation_3	Function object taking 4 Point_3, returning enum...
Circumcenter_3	...

The Kernel concept is a superset for many different geometric traits concepts. There are many different possible (and useful) models of Kernel.

Kernel interface

CGAL geometric algorithms are parameterized by a geometric traits.

```
template < class TriangulationTraits_3 ,
           class TriangulationDataStructure_3 = ... >
class Triangulation_3 ;
```

It provides a (attempted minimal) set of types and functions for the geometry :

Nested Type	Requirements
Point_3	Assignable, DefaultConstructible
Segment_3	Assignable, DefaultConstructible
Orientation_3	Function object taking 4 Point_3, returning enum...
Circumcenter_3	...

The Kernel concept is a superset for many different geometric traits concepts. There are many different possible (and useful) models of Kernel.

Kernel interface

CGAL geometric algorithms are parameterized by a geometric traits.

```
template < class TriangulationTraits_3 ,
           class TriangulationDataStructure_3 = ... >
class Triangulation_3 ;
```

It provides a (attempted minimal) set of types and functions for the geometry :

Nested Type	Requirements
Point_3	Assignable, DefaultConstructible
Segment_3	Assignable, DefaultConstructible
Orientation_3	Function object taking 4 Point_3, returning enum...
Circumcenter_3	...

The Kernel concept is a superset for many different geometric traits concepts. There are many different possible (and useful) models of Kernel.

Kernel interface

CGAL geometric algorithms are parameterized by a geometric traits.

```
template < class TriangulationTraits_3 ,
           class TriangulationDataStructure_3 = ... >
class Triangulation_3 ;
```

It provides a (attempted minimal) set of types and functions for the geometry :

Nested Type	Requirements
Point_3	Assignable, DefaultConstructible
Segment_3	Assignable, DefaultConstructible
Orientation_3	Function object taking 4 Point_3, returning enum...
Circumcenter_3	...

The Kernel concept is a superset for many different geometric traits concepts. There are many different possible (and useful) models of Kernel.

Kernel interface

CGAL geometric algorithms are parameterized by a geometric traits.

```
template < class TriangulationTraits_3 ,
           class TriangulationDataStructure_3 = ... >
class Triangulation_3 ;
```

It provides a (attempted minimal) set of types and functions for the geometry :

Nested Type	Requirements
Point_3	Assignable, DefaultConstructible
Segment_3	Assignable, DefaultConstructible
Orientation_3	Function object taking 4 Point_3, returning enum...
Circumcenter_3	...

The Kernel concept is a superset for many different geometric traits concepts. There are many different possible (and useful) models of Kernel.

Kernels

The kernel can be used as a model for the geometric traits class parameter in numerous algorithms.

Basic kernels, parameterized by number types:
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> is a model of the TriangulationTraits_3 concept.

Kernel functionality is also available as free functions:

```
CGAL::orientation(p, q, r)
```

Kernels

The kernel can be used as a model for the geometric traits class parameter in numerous algorithms.

Basic kernels, parameterized by number types:
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> is a model of the TriangulationTraits_3 concept.

Kernel functionality is also available as free functions:

```
CGAL::orientation(p, q, r)
```

Kernels

The kernel can be used as a model for the geometric traits class parameter in numerous algorithms.

Basic kernels, parameterized by number types:
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> is a model of the TriangulationTraits_3 concept.

Kernel functionality is also available as free functions:

```
CGAL::orientation(p, q, r)
```

Kernels

The kernel can be used as a model for the geometric traits class parameter in numerous algorithms.

Basic kernels, parameterized by number types:
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> is a model of the TriangulationTraits_3 concept.

Kernel functionality is also available as free functions:

```
CGAL::orientation(p, q, r)
```

Kernels

The kernel can be used as a model for the geometric traits class parameter in numerous algorithms.

Basic kernels, parameterized by number types:
Cartesian<FT>
Homogeneous<RT>

Ex : Triangulation_3<Cartesian<double> >

Cartesian<double> is a model of the TriangulationTraits_3 concept.

Kernel functionality is also available as free functions:

```
CGAL::orientation(p, q, r)
```

Number types

Valid parameters for the Cartesian kernel...

FP :

double, float

Multiple precision :

Gmpz, Gmpq, CGAL::MP_Float, leda::integer...

Number types that include filtering:

leda::real, CORE::Expr, CGAL::Lazy_exact_nt<>

Number types

Valid parameters for the Cartesian kernel...
FP :
double, float
Multiple precision :
Gmpz, Gmpq, CGAL::MP_Float, leda::integer...
Number types that include filtering:
leda::real, CORE::Expr, CGAL::Lazy_exact_nt<>

Number types

Valid parameters for the Cartesian kernel...
FP :
double, float
Multiple precision :
Gmpz, Gmpq, CGAL::MP_Float, leda::integer...
Number types that include filtering:
leda::real, CORE::Expr, CGAL::Lazy_exact_nt<>

Number types

Valid parameters for the Cartesian kernel...
FP :
double, float
Multiple precision :
Gmpz, Gmpq, CGAL::MP_Float, leda::integer...
Number types that include filtering:
leda::real, CORE::Expr, CGAL::Lazy_exact_nt<>

Filtered kernels

Internal tools:
Interval arithmetic: CGAL::Interval_nt, boost::interval

Filtered predicates generator using C++ exceptions: CGAL::Filtered_predicate<>

CGAL::Filtered_kernel< *K* > provides some predicates based on static filtering, and all others dynamic (IA).

Recommended kernels:
CGAL::Exact_predicates_exact_constructions_kernel
CGAL::Exact_predicates_inexact_constructions_kernel

Filtered kernels

Internal tools:
Interval arithmetic: CGAL::Interval_nt, boost::interval

Filtered predicates generator using C++ exceptions: CGAL::Filtered_predicate<>

CGAL::Filtered_kernel< *K* > provides some predicates based on static filtering, and all others dynamic (IA).

Recommended kernels:
CGAL::Exact_predicates_exact_constructions_kernel
CGAL::Exact_predicates_inexact_constructions_kernel

Filtered kernels

Internal tools:
Interval arithmetic: CGAL::Interval_nt, boost::interval

Filtered predicates generator using C++ exceptions: CGAL::Filtered_predicate<>

CGAL::Filtered_kernel< *K* > provides some predicates based on static filtering, and all others dynamic (IA).

Recommended kernels:
CGAL::Exact_predicates_exact_constructions_kernel
CGAL::Exact_predicates_inexact_constructions_kernel

Example 1

```
template < typename K >
struct My_orientation_2
{
  typedef typename K::RT       RT;
  typedef typename K::Point_2  Point_2;

  CGAL::Orientation
  operator()(const Point_2 &p, const Point_2 &q,
             const Point_2 &r) const
  {
    RT prx = p.x() - r.x();   RT pry = p.y() - r.y();
    RT qrx = q.x() - r.x();   RT qry = q.y() - r.y();
    return static_cast<CGAL::Orientation> (
             CGAL::sign( prx*qry - qrx*pqy ) );
  }
};
```

Example 2

```
// We use it:
typedef CGAL::Cartesian<double>  Kernel;
Kernel::Point_2 p(1, 2), q(2, 3), r(4, 5);
My_orientation_2<Kernel> orientation;
CGAL::Orientation ori = orientation(p, q, r);
```

Using `Filtered_predicate`

```
typedef CGAL::Simple_cartesian<double> K;
typedef CGAL::Simple_cartesian<CGAL::Interval_nt_advanced> FK;
typedef CGAL::Simple_cartesian<CGAL::MP_Float> EK;
typedef CGAL::Cartesian_converter<K, EK> C2E;
typedef CGAL::Cartesian_converter<K, FK> C2F;

typedef CGAL::Filtered_predicate<My_orientation_2<EK>,
                                 My_orientation_2<FK>,
                                 C2E, C2F>  Orientation_2;

...
  K::Point_2 p(1,2), q(2,3), r(3,4);
  Orientation_2 orientation;
  orientation(p, q, r);
  return 0;
```

C++ is not everything: what about other languages?

We have started to write CGAL wrappers in other languages:

- Scilab, Matlab: not object-oriented, complete rethinking needed
- Python, Java: object-oriented, similar interface and paradigms

Python interface

Python is:

- object-oriented (syntax similar to C++)
- interpreted (precompiled, no templates)
- general purpose, used more and more for scientific tasks
- C++ wrappers generators exist
- cgal-python has already wrapped several CGAL packages
  `http://cgal-python.gforge.inria.fr/`

Java interface

Java is:

- as important as C++
- simpler (garbage collection...)
- syntax is in the same family as C++
- status: the 2D triangulation class has been wrapped.

Conclusion

Thank you for your attention.