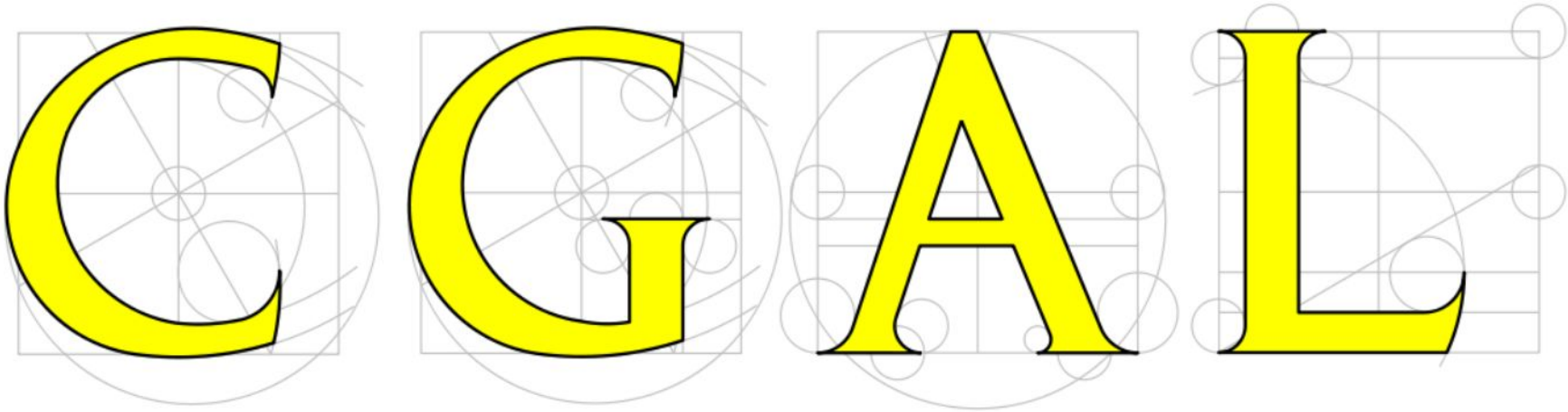


Tutorial

Andreas Fabri
GeometryFactory

Pierre Alliez
Inria

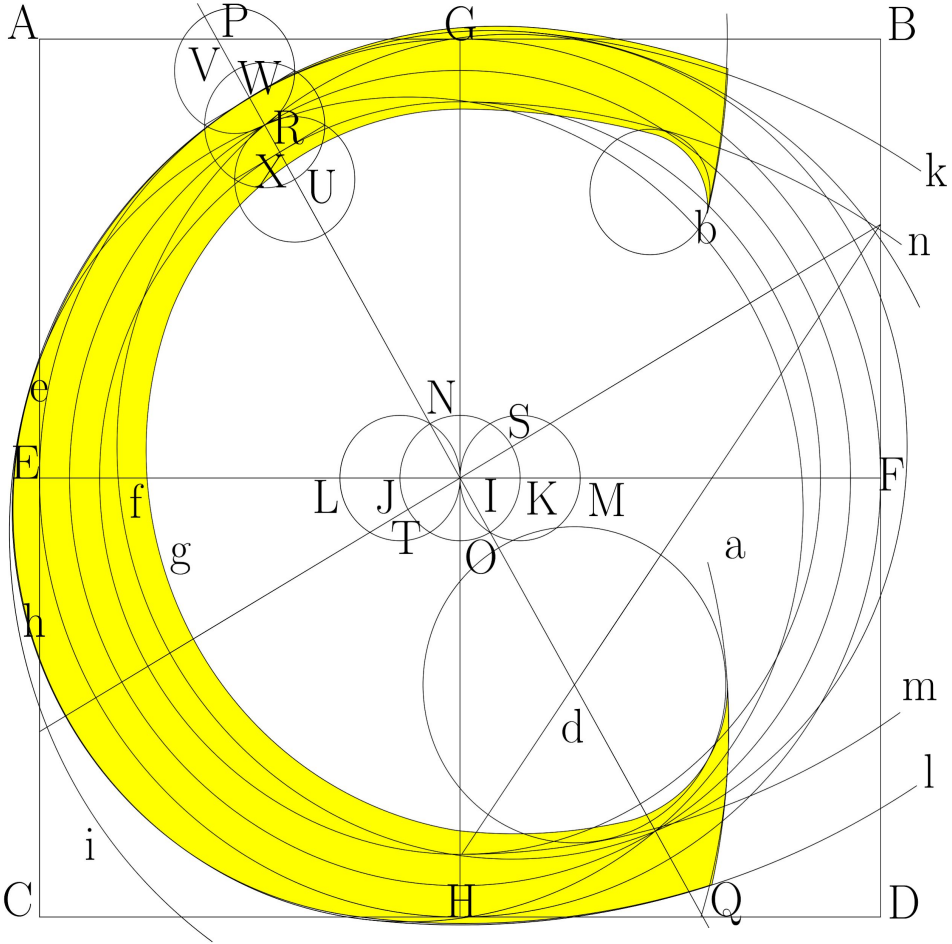
Outline

Today

- Getting Started
- 2D mesh generation, 3D mesh generation, isotropic remeshing

Tomorrow

- Polygon Meshes
- Polygon Mesh Processing
- Point Set Processing



Getting Started

Mission Statement

“Make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications”

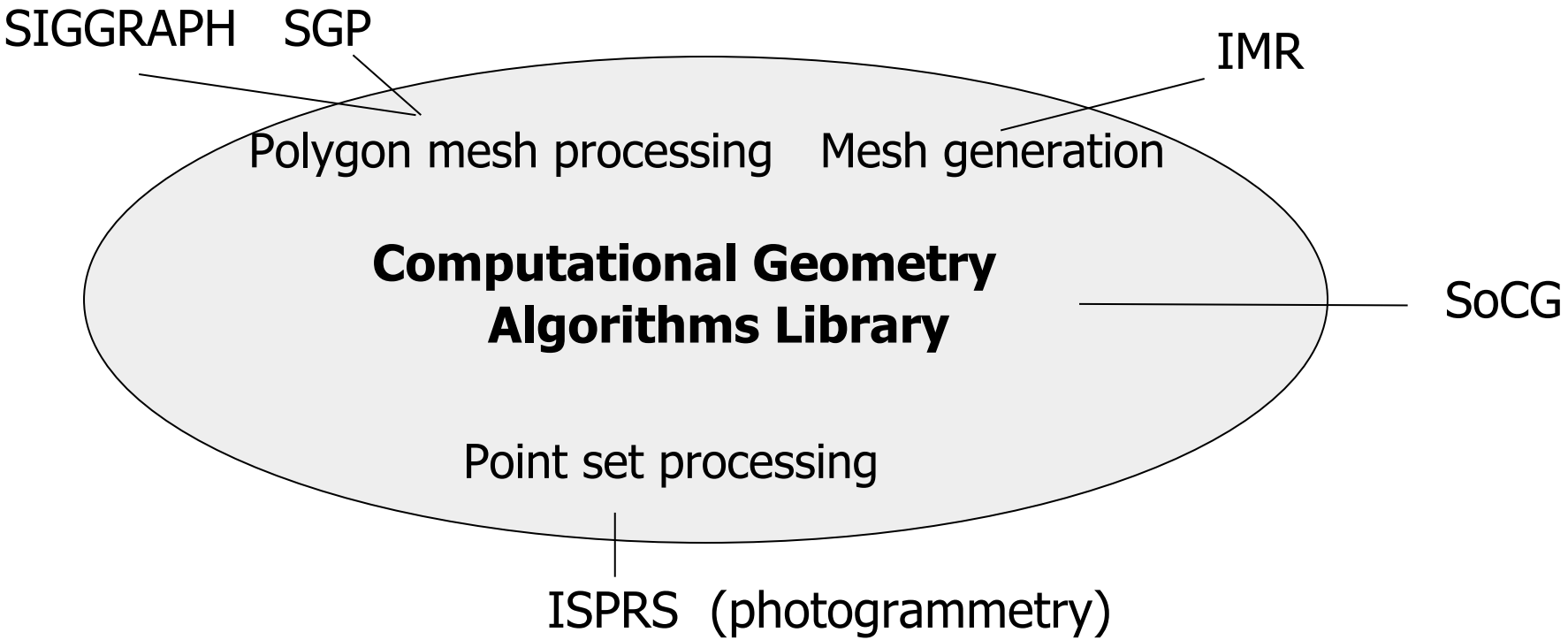
CGAL EU Project Proposal, 1996

The CGAL Library

- C++ template class library
- Header only
- Cross platform: Windows, Linux, macOs, Android
- Supported compilers: VC++, g++, clang
- For several packages Java/Python bindings (using Swig)

- Dependencies
  - Boost: selected libraries
  - GMP, Mpfr: exact number types
  - Eigen: algebraic solvers
  - LAStools: I/O for LAS file format
  - OpenCV: random forests

CGAL - Towards a Geometric Computing Library



CGAL in Numbers

700,000	lines of C++ code
10,000	downloads/year (+ package managers)
3,500	manual pages (user and reference manual)
1,000	subscribers to user mailing list
200	commercial users
120	software components
20	active developers
6	months release cycle
2	licenses: Open Source and commercial

The CGAL Project

- Started 1996 as EU Research Project
- Academic project partners make long term commitment

- Editorial Board
  - Steers and animates the project
  - Reviews submissions

- Development infrastructure
  - CGAL on github, doxygen, travis, nightly testsuite (~40 platforms)
  - Two developer meetings per year

GeometryFactory

- 7 engineers with a PhD in CS with focus on geometric computing
  + Rao Fu, Grapes Fellow, + postdoc shared with Inria

- Sales of CGAL software components
- Support to increase customer productivity

- Backed by the CGAL Open Source Project

- Actively involved in the CGAL Project:
  3 EB members (including release management)

Commercial Off The Shelf Components

Academia

publication

prototype

GeometryFactory

Component Repository

CGAL component

CGAL component

CGAL component

integrate

Industrial CGAL User

Complex End User Software

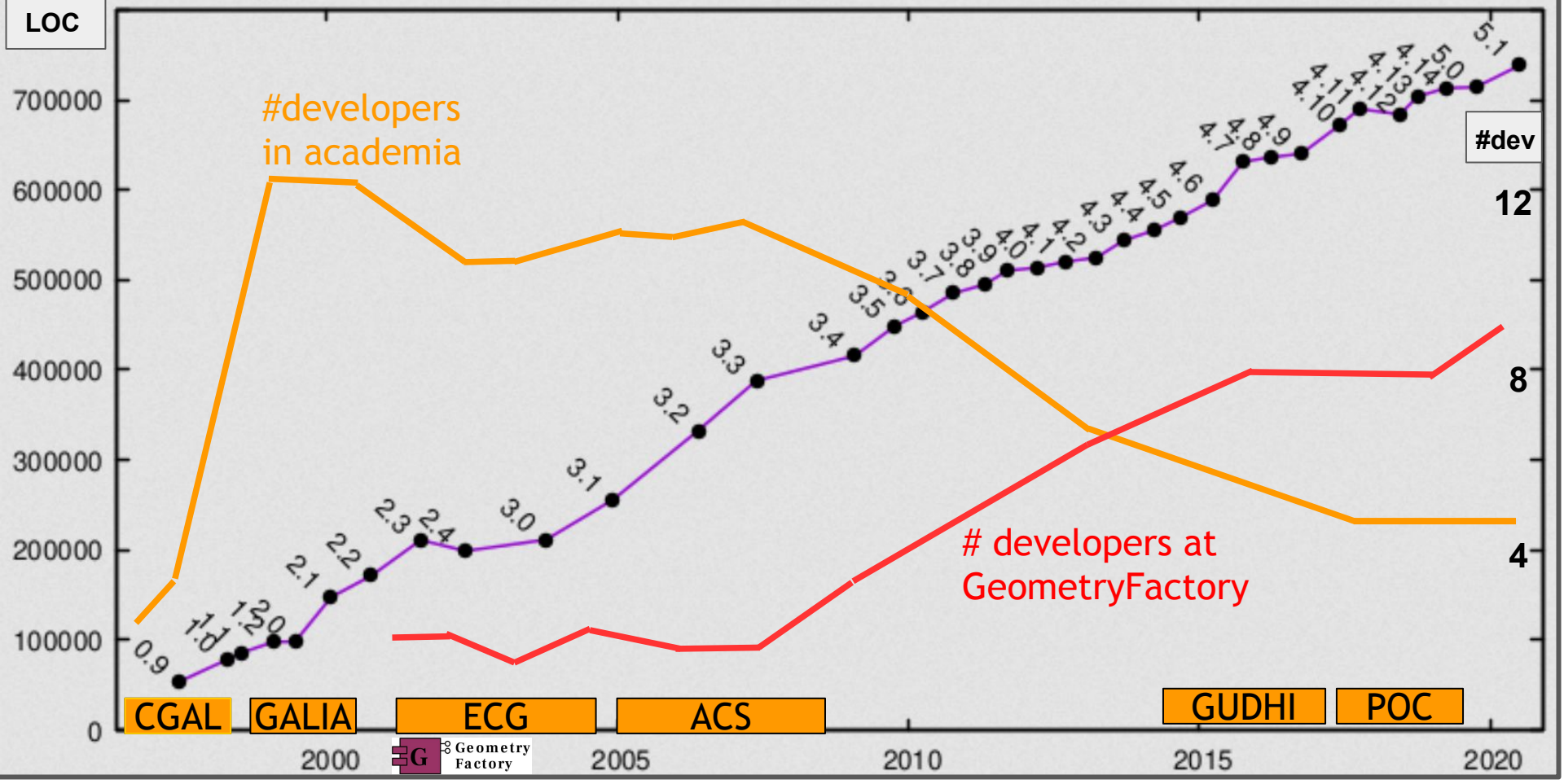
component

component

component

CGAL component

component


LOC
700000
600000
500000
400000
300000
200000
100000
0
#developers
in academia
#dev
12
8
4
developers at
GeometryFactory
0.9
1.0
1.1
1.2
2.0
2.1
2.2
2.3
2.4
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4.0
4.1
4.2
4.3
4.4
4.5
4.6
4.7
4.8
4.9
4.10
4.11
4.12
4.13
4.14
5.0
5.1
CGAL
GALIA
ECG
ACS
GUDHI
POC
2000
2005
2010
2015
2020
Geometry
Factory

CGAL Kernels and *Exact Geometric Computation* [Yap 96]

CGAL Kernels

- Kernel = Foundation layer of all data structures and algorithms
- Constant size geometric objects
  - point, vector, direction
  - segment, ray, line, plane
  - triangle, tetrahedron
  - iso-rectangle, iso-cuboid
  - circle, sphere
- Operations on these types
  - Tests: orientation, intersection, bounded side, etc.
  - Constructions: distance, midpoint, projection, intersection, etc.

CGAL Kernels

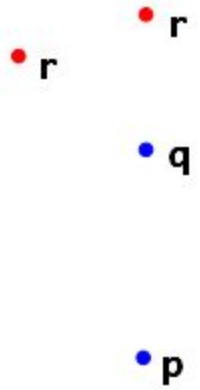
- `Exact_predicates_inexact_constructions_kernel`
- `Exact_predicates_exact_constructions_kernel`
- `Exact_predicates_exact_constructions_kernel_with_sqrt`

- `Cartesian` **vs** `Homogeneous`
- `Cartesian` **vs** `Simple_cartesian`

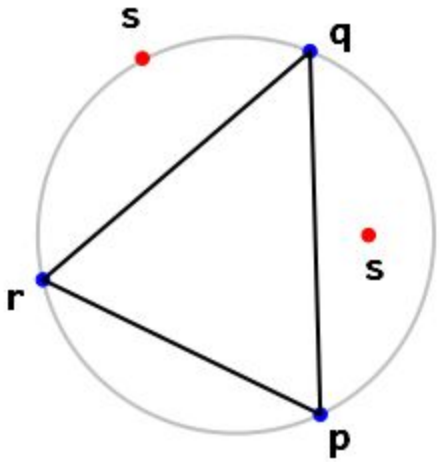
- `Simple_cartesian<double>`

- Different kernels use different number types: double, interval arithmetic, arbitrary precision integer, rational, or algebraic numbers

Predicates and Constructions

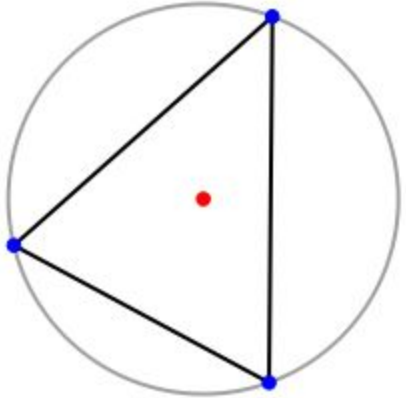


orientation()



in_circle()

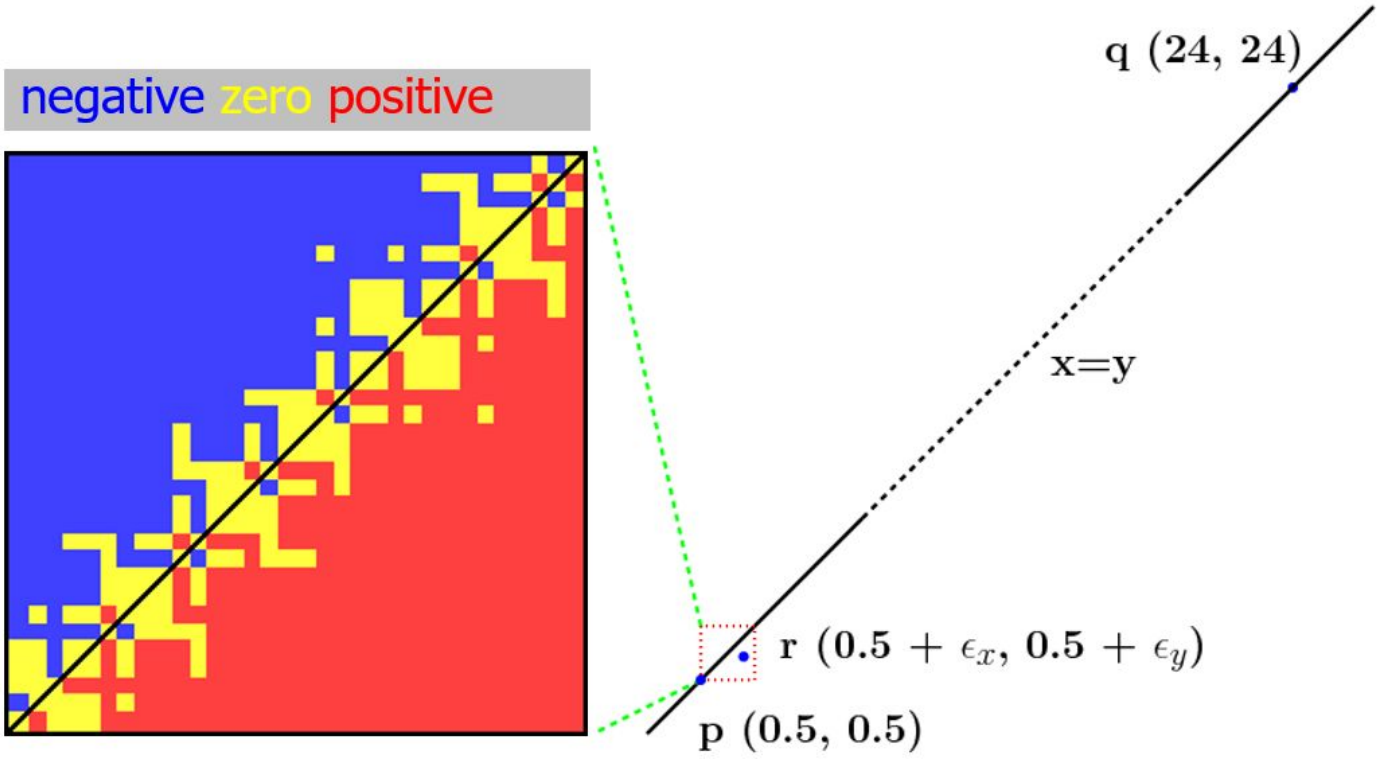
intersection()



circumcenter()

The Trouble with Double

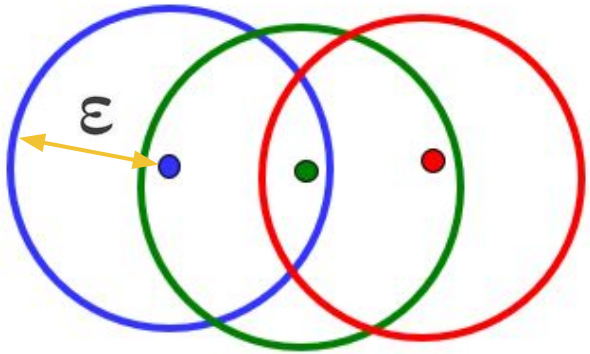
orientation(p,q,r) = sign((px-rx)*(qy-ry) - (py-ry)*(qx-rx))



Heuristic Epsilons

“If a value is almost zero consider it as zero” can lead to problems

For example **P** == **Q** and **Q** == **R** ⇒ **P** == **R** no longer holds



Number Types

Given: orientation(p,q,r) = sign((px-rx)*(qy-ry) - (py-ry)*(qx-rx))

- Built-in type double precision
  - For an expression we can precompute the maximal error that may occur
- Interval arithmetic
  - When doing an arithmetic operation we can ensure that the result encloses the exact value
  - Computing a sign of an interval *[inf,sup]* is possible if *inf > 0* or *sup < 0*
- Arbitrary precision integers, rationals, algebraic numbers
  - Arithmetic operations are always exact but slow

Fast and Exact Predicates

Given: orientation(p,q,r) = sign((px-rx)(qy-ry)-(py-ry)(qx-rx))

Precompute: static error bound for arithmetic expression with double

At runtime:

Compute arithmetic expression with double

If result is uncertain (that is we are in the error interval):

Compute arithmetic expression with interval arithmetic

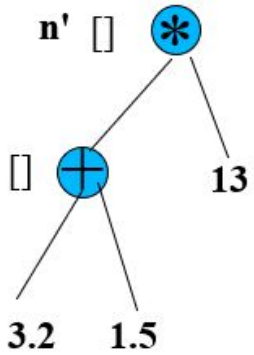
If result is uncertain (that is the interval contains 0):

Compute arithmetic expression with arbitrary precision arithmetic

Fast and Exact Arithmetic

Lazy number = double interval + arithmetic expression tree

*n = (3.2 + 1.5) * 13*

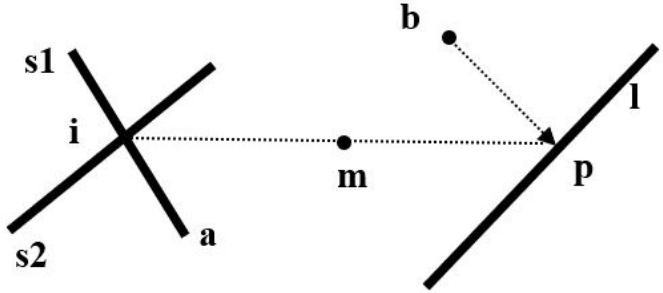


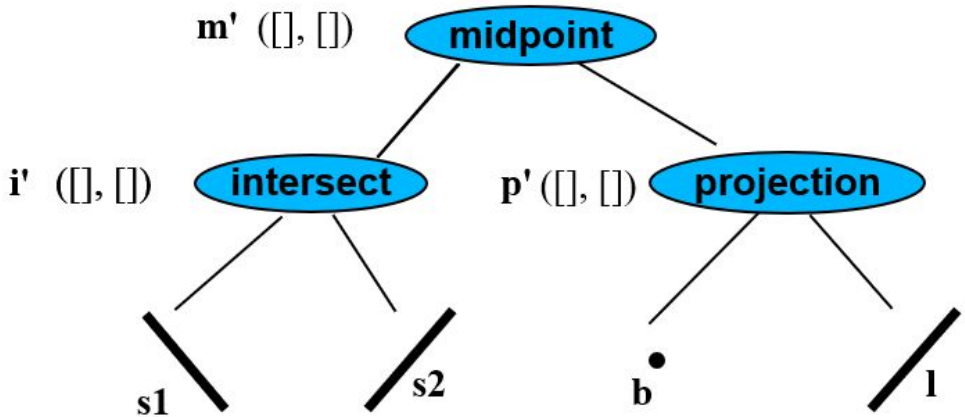
Test that may trigger exact evaluation: *if(n' < m')* namely when intervals overlap

[Yap et al 1999]

Fast and Exact Constructions

Lazy object = object with interval coordinates + construction tree





Test that may trigger exact evaluation: if(collinear (a' , m' , b'))

[Pion et al 2011]

A Common Error to Avoid

Do not use a test where you know that the filter will fail

```
Point R = midpoint(P, Q);


if (collinear(P, Q, R) {
…
}
```

Exact Geometric Computing: Merits and Limitations

- If you can stay all the time in the paradigm you are fine
- Otherwise: Robustness inside the black box
- Time penalty of exact predicates is reasonable,
  e.g., 10% for 3D Delaunay triangulation

- Topology preserving rounding is non-trivial
- Construction depth must be reasonable

- Hybrid approach:
  - Use exact predicates where possible
  - Switch to double where necessary
  - Make an exact construction and convert back to double

Concepts and Models

STL Genericity

```
template <class Key, class Compare>
class set {
  Compare compare;

  insert(Key k)
  {
     if (compare(k, treenode.key) == ..)
       insertLeft(k);
     else
       insertRight(k);
  }
};
```

Concept = Named Requirements

C++ named requirements: *Compare*

Compare is a set of requirements expected by some of the standard library facilities from the user-provided function object types.

The return value of the function call operation applied to an object of a type satisfying *Compare*, when contextually converted to `bool`, yields `true` if the first argument of the call appears before the second in the *strict weak ordering relation* induced by this type, and `false` otherwise.

As with any *BinaryPredicate*, evaluation of that expression is not allowed to call non-const functions through the dereferenced iterators.

Requirements

The type T satisfies *Compare* if

- The type T satisfies *BinaryPredicate*, and

Given

- `comp`, an object of type T
- `equiv(a, b)`, an expression equivalent to `!comp(a, b) && !comp(b, a)`

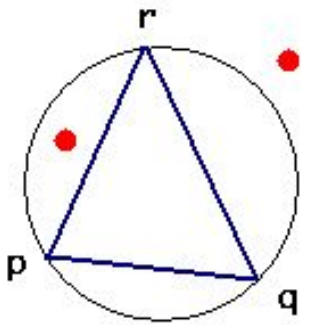
The following expressions must be valid and have their specified effects

Expression	Return type	Requirements
`comp(a, b)`	implicitly convertible to `bool`	Establishes strict weak ordering relation with the following properties • For all a, `comp(a,a)==false` • If `comp(a,b)==true` then `comp(b,a)==false` • if `comp(a,b)==true` and `comp(b,c)==true` then `comp(a,c)==true`
`equiv(a, b)`	`bool`	Establishes equivalence relationship with the following properties • For all a, `equiv(a,a)==true` • If `equiv(a,b)==true`, then `equiv(b,a)==true` • If `equiv(a,b)==true` and `equiv(b,c)==true`, then `equiv(a,c)==true`

Note: `comp` induces a *strict total ordering* on the equivalence classes determined by `equiv`

CGAL Genericity

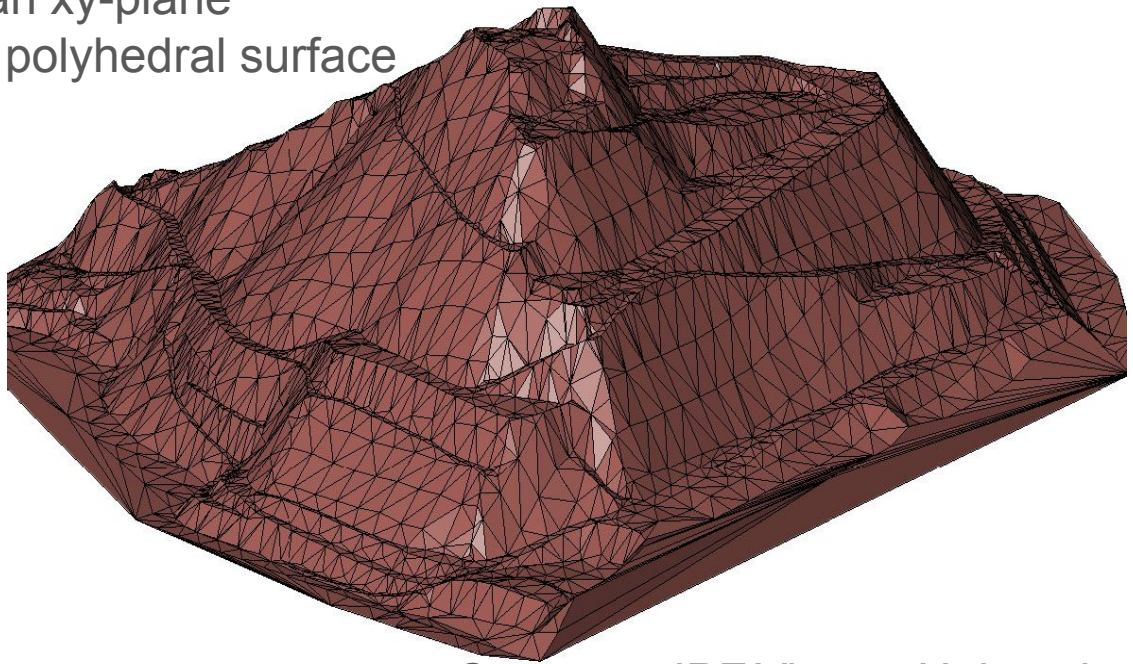
```
template < class Geometry >
class Delaunay_triangulation_2 {
    Geometry::Orientation orientation;
    Geometry::In_circle in_circle;

    void insert(Geometry::Point t) {
        ...
        if(in_circle(p,q,r,t)) {...}
        ...
        if(orientation(p,q,r){...}
    }
};
```



CGAL Genericity

Without explicit conversion to points in the plane

- Triangulate the terrain in an xy-plane
- Triangulate the faces of a polyhedral surface



Courtesy: IPF,Vienna University of Technology & Inpho GmbH

CGAL Genericity

Documentation of `CGAL::Delaunay_triangulation_2< Traits, Tds>`:

- `Traits` a model of `DelaunayTriangulationTraits_2`

Documentation of `DelaunayTriangulationTraits_2`:

- Has Models

  All CGAL kernels

  `CGAL::Projection_traits_xy_3<K>`

  `CGAL::Projection_traits_3<K>`

Reference Manual Page of the Concept

CGAL 5.2 - 2D Triangulation

List of all members

DelaunayTriangulationTraits_2 Concept Reference

2D Triangulation Reference » Concepts

Definition

In addition to the requirements of the concept `TriangulationTraits_2` the concept `DelaunayTriangulationTraits_2` requires a predicate to check the empty circle property. The corresponding predicate type is called type `Side_of_oriented_circle_2`.

The additional types `Line_2`, `Ray_2` and the constructor objects `Construct_ray_2`, `Construct_circumcenter_2`, `Construct_bisector_2`, `Construct_midpoint` are used to build the dual Voronoi diagram and are required only if the dual functions are called. The additional predicate type `Compare_distance_2` is required if the method `nearest_vertex()` is used.

Refines:

`TriangulationTraits_2`

Has Models:

CGAL kernels

`CGAL::Projection_traits_xy_3<K>` (not for dual Voronoi functions)

`CGAL::Projection_traits_yz_3<K>` (not for dual Voronoi functions)

`CGAL::Projection_traits_xz_3<K>` (not for dual Voronoi functions)

See also

`TriangulationTraits_2`

Types

typedef unspecified_type	Line_2	The line type. More...
typedef unspecified_type	Ray_2	The type for ray. More...
typedef unspecified_type	Side_of_oriented_circle_2	A function object to perform an incircle test for a point and three other points. More...
typedef unspecified_type	Compare_distance_2	A function object to compare two distances for three points. More...
typedef unspecified_type	Construct_circumcenter_2	A function object to construct the circumcenter of three points. More...
typedef unspecified_type	Construct_bisector_2	A function object to construct the bisector of two points. More...