CGAL

A C++ Computational Geometric Algorithms Library

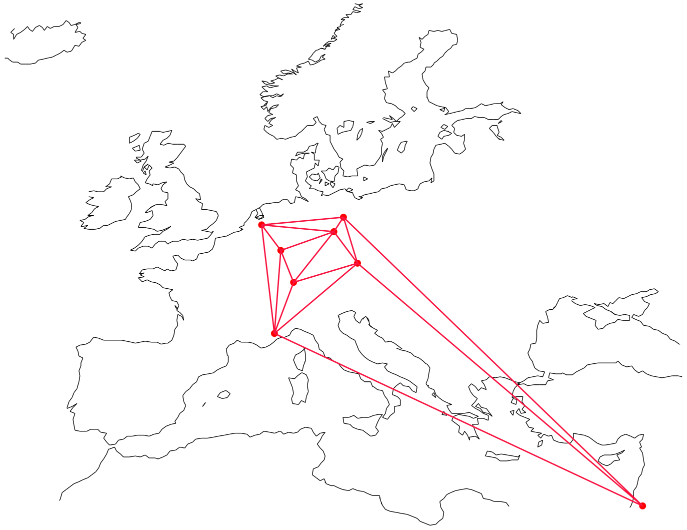
http://www.cgal.org

Mariette Yvinec

INRIA
SOPHIA ANTIPOLIS

CGAL ESPRIT LTR project 21957 http://www.cs.uu.nl/CGAL/

GALIA ESPRIT LTR project 28155 http://www.mpi-sb.mpg.de/GALIA/



RISC Linz

Utrecht University

INRIA Sophia Antipolis

ETH Zürich

MPI Saarbrücken

Tel Aviv University

Free Universität Berlin

Halle university

Computational Geometry

- real RAM model
- general position
- theoretical optimality

- hard to code algorithm
- degeneracies
- numerical inaccuracy

Geometric Computing

CGAL

Robust program

for industrial application

Computational Geometry

Already existing software ?

Workbench for CG	Carleton
XYZ Geobench	Utrecht
Plageo	Zürich
C++GAL	INRIA
CEDA	Sarrebrücken
Gems	Minneapolis

The goals of CGAL's design

- Robustness
- Efficiency
- Flexibility
- Easy to use

Means

- C++, STL
- Generic programming
- STL-like
  templates
  partial specialisation
  traits class

Genericity in STL et CGAL

Templates

◁ Templated containers : vector, list, map, set, ...

◁ Generic algorithms

◁ Iterators

◁ Objects functions

Examples

```
template <class T> class List { ... };
template<class RandomAccessIterator, class Compare>
void sort(RandomAccessIterator first,
          RandomAccessIterator last ,
          Compare comp);
```

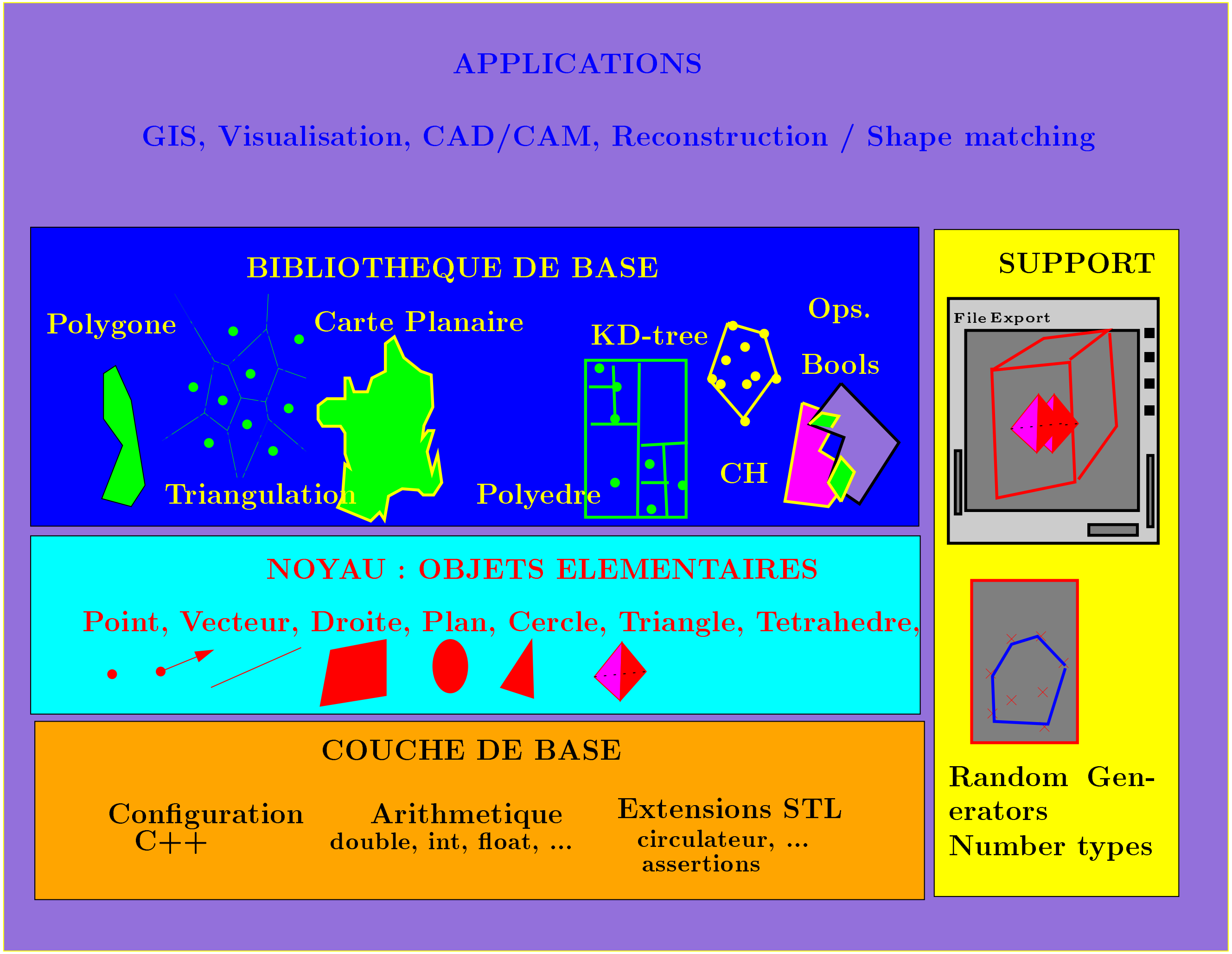
Current release **CGAL 2.4** May 2002

1 200 classes
273 000 lines
50 Mo (without doc)

download from :
http://www.cgal.org/

Supported Platforms

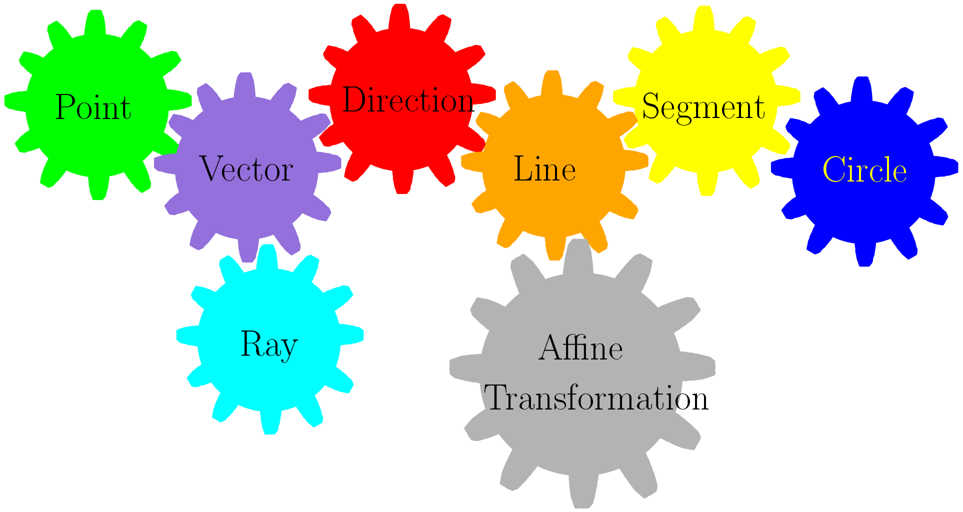
Operating system	**Compiler**
IRIX 6.5	SGI Mips(Pro) CC 7.3 GNU g++
Solaris 2.6 (and later)	GNU g++ KAI C++
Linux 2.x	GNU g++
Windows 95/98/2000/xp/NT4	MS Visual C++ 6.0 MS Visual C++ 7.0 Borland C++ 5.5



APPLICATIONS
GIS, Visualisation, CAD/CAM, Reconstruction / Shape matching
BIBLIOTHEQUE DE BASE
Polygone
Carte Planaire
KD-tree
Ops.
Bools
CH
Triangulation
Polyedre
SUPPORT
File Export
NOYAU : OBJETS ELEMENTAIRES
Point, Vecteur, Droite, Plan, Cercle, Triangle, Tetrahedre,
COUCHE DE BASE
Configuration C++
Arithmetique
double, int, float, ...
Extensions STL
circulateur, ...
assertions
Random Generators
Number types

Kernel Concept:

a collection of classes with cooperating functionality



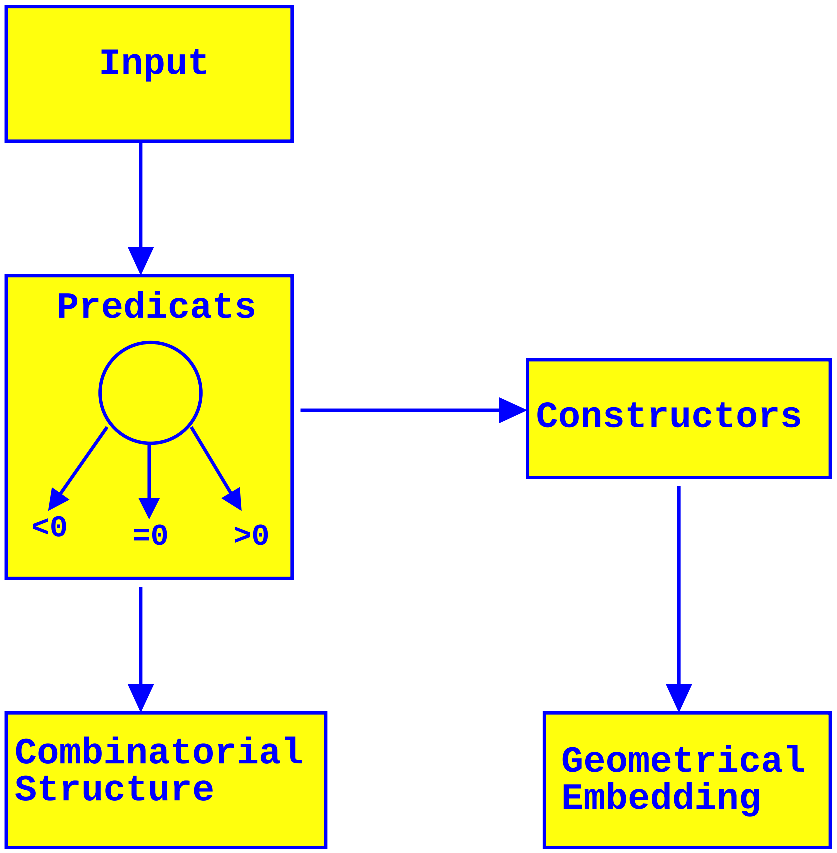
◁ elementary objects

◁ computations on them :
affine transformations,
predicates,
intersections,
distances

Predicates and constructors

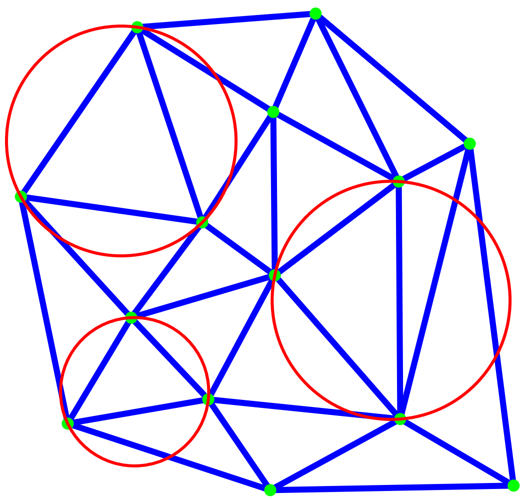
Predicates : numerical tests
- decide branching
- compute combinatorics

Constructors :
- compute the embedding
  of the combinatorial structure

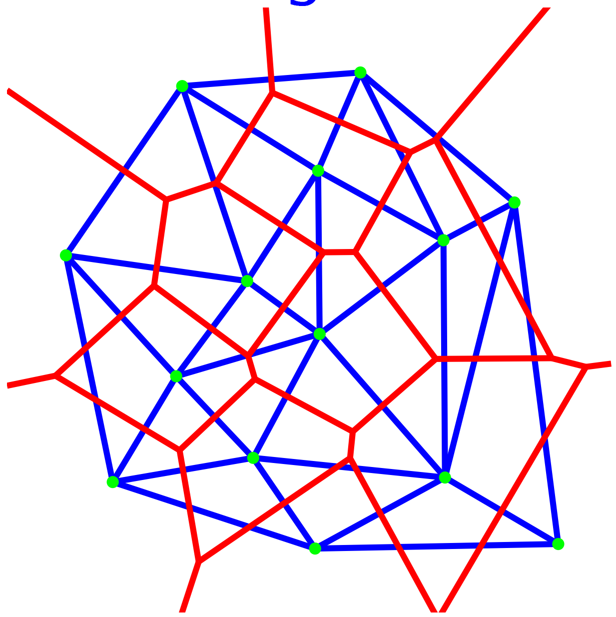


Example

Delaunay Triangulation



Voronoi Diagram



2D Kernel concept

Objects

Point, Vector, Direction,
Droite, Rayon, Segment
Triangle,
Isothetic Rectangle
Circle, Parabola

Predicates

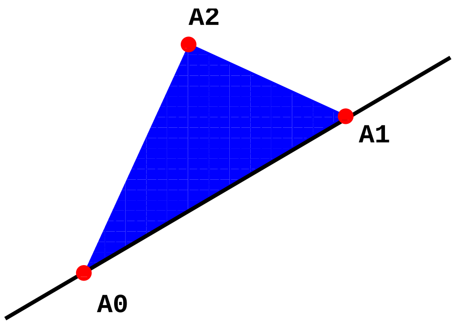
coordinate comparison
orientation test
incircle test
intersections test
distances comparison

Constructors :

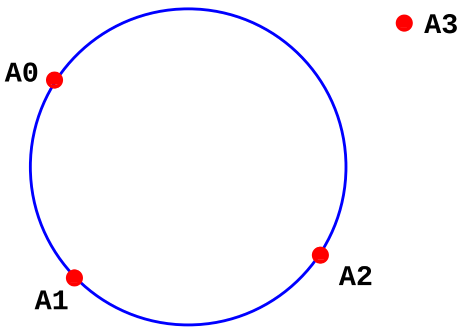
Intersections comput.
Transformations :
tranlations, rotations,
scaling, general

Predicates of the 2D Kernel

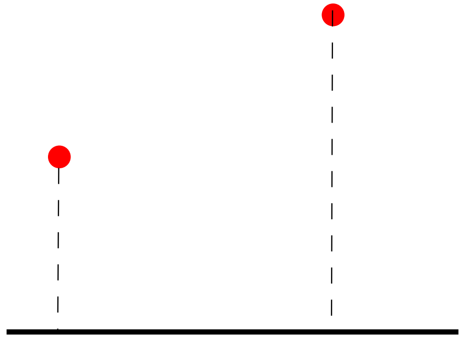
- Orientation test



- Incircle test



- Comparison of coordinates



3D Kernel concept

Objects

- Point, Vector, Direction,
- Droite, Rayon, Segment
- Triangle,
- **Plane**
- **Tetrahedron,**
- **Iso_Rectangle_3d**
- Sphere

Predicates

- coordinate comparison
- orientation test
- incircle test
- intersections test
- distances comparison

Constructors :

- Intersections comput.
- Transformations :
  - tranlations, rotations,
  - scaling, general

The dD Kernel

Objects

Point, Vector, Direction,
Droite, Rayon, Segment
Hyperplane,
Sphere

Predicates

coordinate comparison
orientation test
incircle test
intersections test
distances comparison

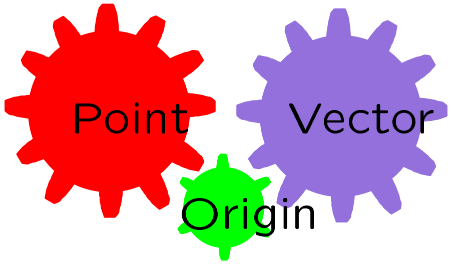
Constructors :

Intersections comput.
Transformations :
tranlations, rotations,
scaling, general

Affine Geometry

The CGAL kernel concept correponds to usual affine geometry

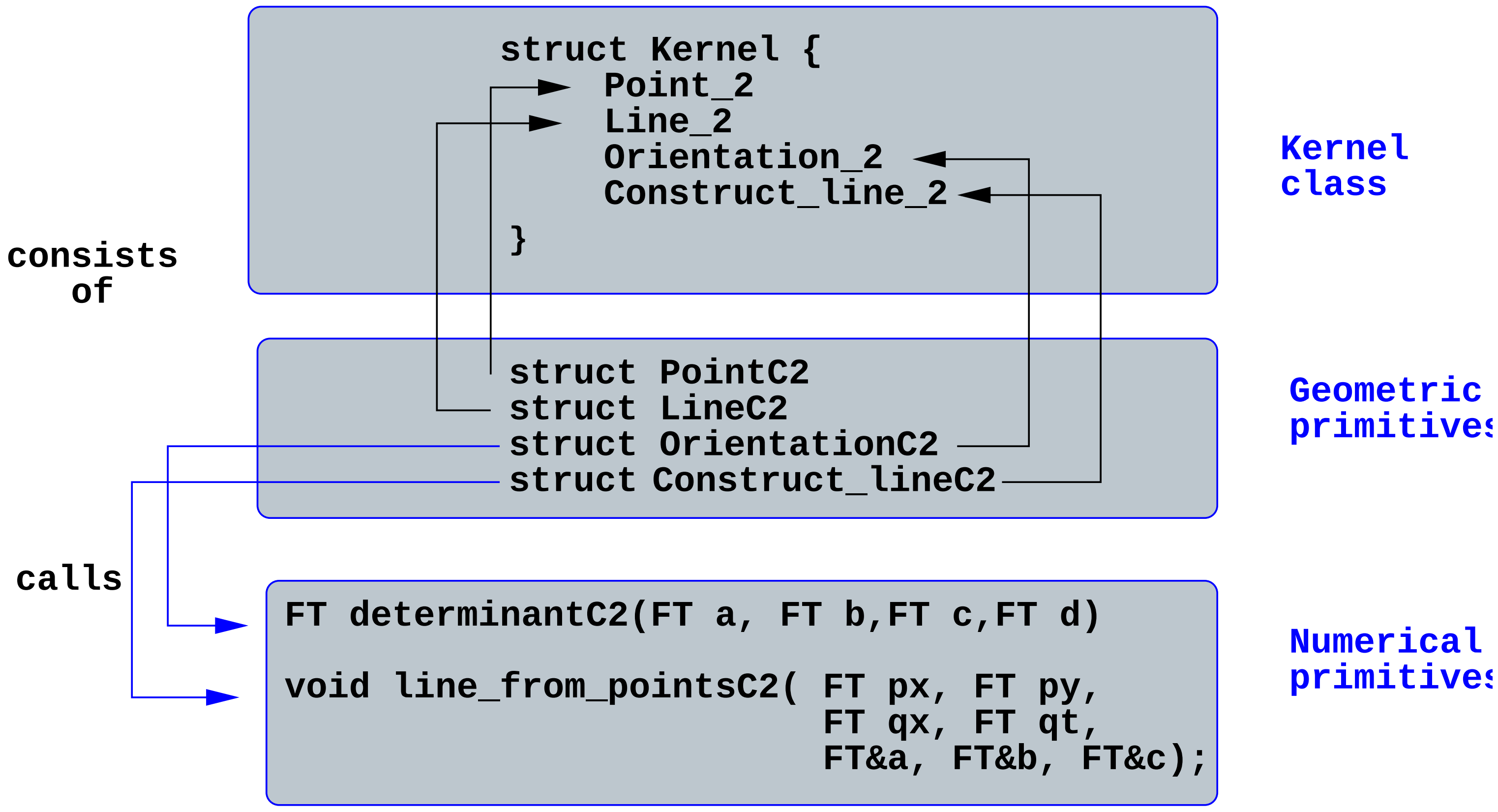
Point - Origin → Vector
Point + Vector → Point
Point - Point → Vector



Unsupported
Point + Point → Vector

Point p + Point q - Point r	??
(Point p + Point q) - Point r	Vector
Point p + (Point q - Point r)	Point

midpoint(Point a, Point b) = a + (1/2.)*(b-a)



Kernel architecture

Adaptable and extensible Kernels

objects, functors, access functions

```
struct Kernel {
    typedef MyPoint                  Point_2;
    typedef MyLine                   Line_2;
    ...
    typedef MyConstruct_line         Construct_line_2;
    typedef MyOrientation            Orientation_2;
    .....
    Construct_line_2                 construct_line_2_object();
    Orientation_2                    orientation_2_object();
};
```

Adaptable and extensible Kernels II

Because clases have to know about each others

```
template<class K> struct MyPoint { K::Line_2 ... }
template<class K> struct MyLine  { ... }

struct Kernel {
    typedef MyPoint<Kernel>              Point_2;
    typedef MyLine<Kernel>               Line_2;
    ...
    typedef MyConstruct_line<Kernel>     Construct_line_2;
    typedef MyOrientation <Kernel>       Orientation_2;
    ...
    Construct_line_2                     construct_line_2_object();
    Orientation_2                        orientation_2_object();
};
```

Adaptable and extensible Kernels III

To change something in the Kernel:

```
struct New_kernel : public Kernel {
    typedef NewPoint<New_kernel>        Point_2;
    typedef NewOrientation<New_kernel> Orientation_2;
}
```

Let's try

```
New_kernel::Point_2 p,q;
New_kernel::Construct_line_2 construct_line_2;
New_kernel::Line_2 l = construct_line_2(p,q); // Error
```

Error message :
No instance of function MyConstruct<Kernel>::operator()
matches the argument list
The argument types are : (New_kernel::Point_2, New_kernel::Point_2)

Adaptable and extensible Kernels IIII : : deferred instantiation

```
template < class K >
struct Kernel_base {
    typedef MyPoint<K>             Point_2;
    typedef MyLine<K>              Line_2;
    ...
    typedef MyConstruct_line<K> Construct_line_2;
    typedef MyOrientation<K>      Orientation_2;
    ...
};

struct Kernel : public Kernel_base < Kernel > ;
```

To change something in the Kernel:

```
struct New_kernel : public Kernel_base<New_kernel> {
    typedef NewPoint<New_kernel> Point_2;
    typedef NewOrientation<New_kernel> Orientation_2;
}
```

CGAL Kernel classes

CGAL::Cartesian<NT> ;
CGAL::Homogeneous<NT> ;
CGAL::Simple_cartesian<NT> ;
CGAL::Simple_homogeneous<NT>;

- Cartesian and homogeneous kernels
- Simple and ref_counted kernels
- Templated by a number type

Cartesian Kernels

Point $\left| \begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array} \right.$

compact
simple computations

Algebraic structure

Field type

Homogeneous Kernels

Point $\left| \begin{array}{l} hx \\ hy \\ hw \end{array} \right.$

robust
avoids division

Algebraic structure

Ring type

Example : lines intersection

Cartesian coordinates

$$a_1 x + b_1 y + c_1 = 0$$
$$a_2 x + b_2 y + c_2 = 0$$

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right)$$

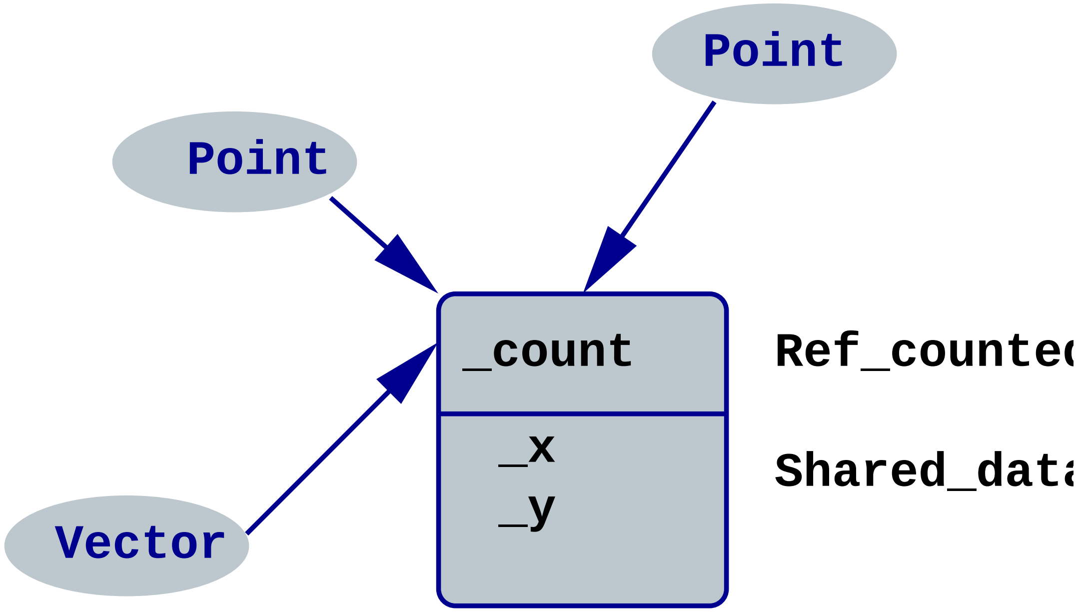
Homogeneous coordinates

$$a_1 hx + b_1 hy + c_1 hw = 0$$
$$a_2 hx + b_2 hy + c_2 hw = 0$$

$$hx = \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$
$$hy = -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}$$
$$hw = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

Simple Kernels

Ref_counted Kernels



Kernel objects are non mutable

Arithmetics : Templated Kernel classes

```
template < class K , class FT>
struct Kernel_base {
    typedef MyPoint<K>                Point_2;
    typedef MyLine<K>                 Line_2;
    ...
    typedef MyConstruct_line<K> Construct_line_2;
    typedef mYOrientation<K>          Orientation_2;
    ...
    Construct_line_2                  construct_line_2_object();
    Orientation_2                     orientation_2_object();
};

template< class FT>
struct Kernel : public Kernel_base < Kernel<FT> ,FT > ;
```

Templated Kernels

template $<$ class NT $>$ class Cartesian ;
template $<$ class NT $>$ class Homogeneous;
template $<$ class NT $>$ class Simple_cartesian ;
template $<$ class NT $>$ class Simple_homogeneous;

Cartesian Kernels : Field type

- *double*
- *quotient<gmpz>*
- *leda_real*

Homogeneous Kernels : Ring type

- *int*
- *gmpz*
- *double*

typedef double coord_types ;
typedef Cartesian<coord_types> Ker;
typedef Ker::Point_2 Point;

Influence of Number Types

Floating point computation	$\Longrightarrow$	speed
Exact number type	$\Longrightarrow$	robustness

```
typedef Cartesian<Nt> Ker;
Nt            sqrt2 = sqrt( N(2) );          // OK for exact numerical type
Ker::Point_2  p(0,0), q(sqrt2,sqrt2);
Ker::Circle_2 C(p, 2);                       // assertion violation otherwise
assert( C.has_on_boundary(q) );
```

Changing arithmetic is easy

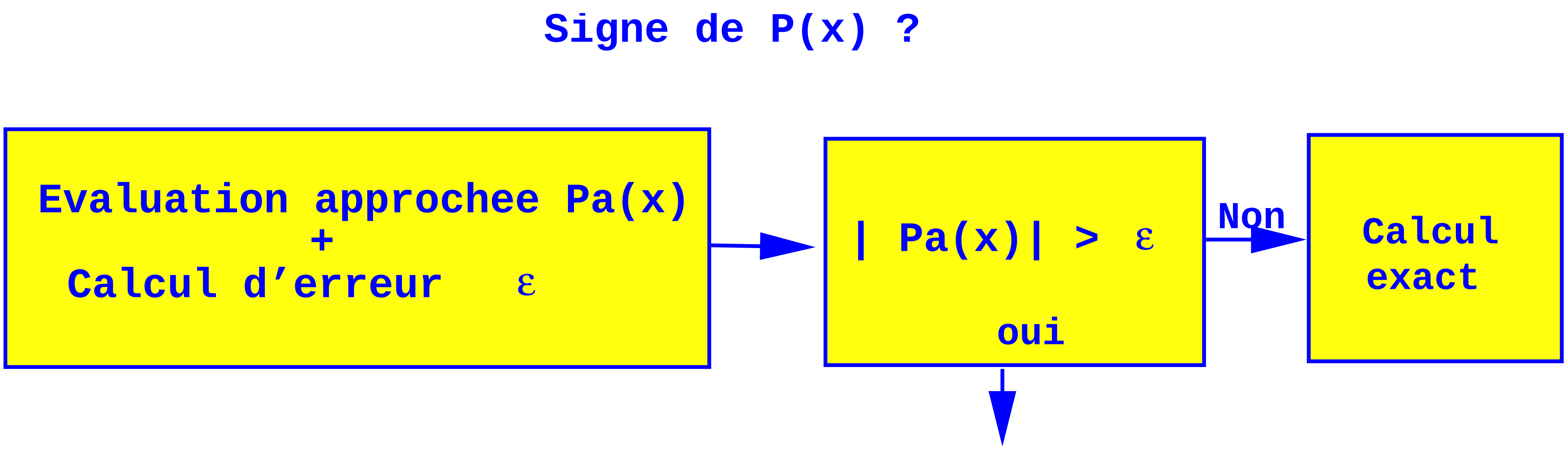
Changing representation is easy

```
typedef double                          coord_types ;
// typedef leda_real                    coord_types ;
typedef Cartesian<coord_types>          Ker;
// Homogeneous<coord_types>             Ker ;
typedef Ker::Point_2                    Point;
...
```

Number types in CGAL

2 concepts	RingNumberType	FieldNumberType
Built-in	int	float double
CLN Class Library for Numbers	cl_integer	cl_rational
GMP GNU Multiple Precision	Gmpz	
LEDA Library of efficient Data types and Algorithms	leda_integer	leda_rational leda_bigfloat leda_reals
CGAL		Quotient<NT> MP_Float Lazy_exact_nt<NT> Interval_nt Fixed_precision_nt Filtered_exact<CT, ET>

Filtered evaluation of predicates



Static filters

Semi-static filters

Dynamic filters (interval arithmetics)

CGAL::Filtered_exact<CT, ET> number type

Wrapper for the number type CT
The low level predicates of CGAL, templated by the number type, are overloaded for the type Filtered_exact<CT,ET> to yield exact results

CT : number type for constructions and storage
ET : exact number type

Interval_nt	to_interval(CT)	: conversion for filtering
ET	convert_to<ET>(CT)	conversion for exact evaluation

Exemple : filtered orientation predicate

```
template < class CT, class ET >
Orientation
orientation (const Filtered_exact<CT, ET> &px...)
{
  try {
    Protect_FPU_rounding p;
    return orientation(px.interval(), py.interval(),
                       qx.interval(), qy.interval(),
                       rx.interval(), ry.interval());
  }
  catch (...) {
    return orientation(px.exact(), py.exact(),
                       qx.exact(), qy.exact(),
                       rx.exact(), ry.exact());
  }
}
```

CGAL::Filtered_kernel<CK, EK, FK, C2E, C2F>

class CK	construction kernel
class EK	exact kernel
class FK	filtering kernel
class C2E	converter CK -> EK
class C2F	converter CK -> FK

Each predicates of such a kernel :

- convert his arguments from CK to FK using C2F
- call the predicate of FK.
- the predicates of FK throw an exception when they cannot conclude
- exceptions are caught and predicate of EK are called

CGAL::Filtered_kernel<CK, EK, FK, C2E, C2F>

Defaults

class EK = Simple_cartesian<MP_Float>, exact kernel

class FK = Simple_cartesian<Interval_nt_advanced>, filtering kernel

class C2E = Cartesian_converter<CK, EK>,

class C2F = Cartesian_converter<CK, FK,Interval_converter<CK::RT>>

Example

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Filtered_kernel.h>
typedef CGAL::Simple_cartesian<double> CK;
typedef CGAL::Filtered_kernel<CK> K1;
struct K : public K1;
```

Basic Library

• Convex Hulls	2D, 3D, dD
• Polygons	
• Polyhedral Surfaces	
• Nef Polyhedra	boolean operations on polygons
• Planar Maps	2D monotone non intersecting curves
• Arrangement	2D general curves
• Triangulations	2D, 3D
• Alpha shapes	
• Geometric Optimisation	
• Search Structures	

Traits classes

convex_hull_points_2<InputIterator, OutputIterator, Traits>
Polygon_2<Traits, Container>
Polyhedron_3<Traits, HDS>
Planar_map_2<Dcel,Traits>
Arrangement_2<Dcel,Traits,Base_node>
Triangulation_3<Traits, TDS>
Min_circle_2<Traits>
Range_tree_k<Traits>

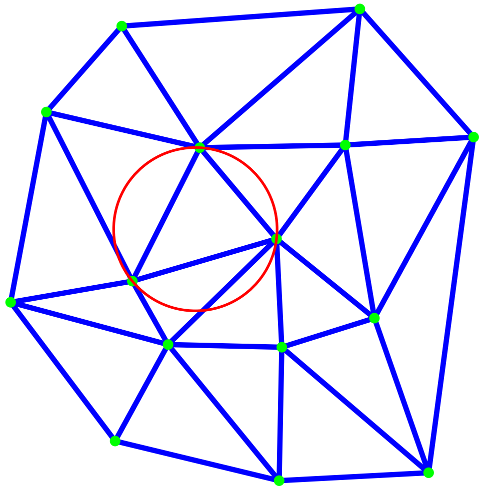
Geometric traits classes provides

Geometric objects + predicates + constructors

Requirements for geometric traits are documented
$\Longrightarrow$ basic library can be used with user defined objects

Default traits classes provided

Example : Delaunay triangulation traits class



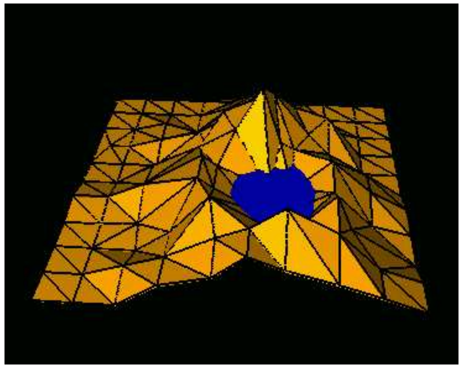
Requirements

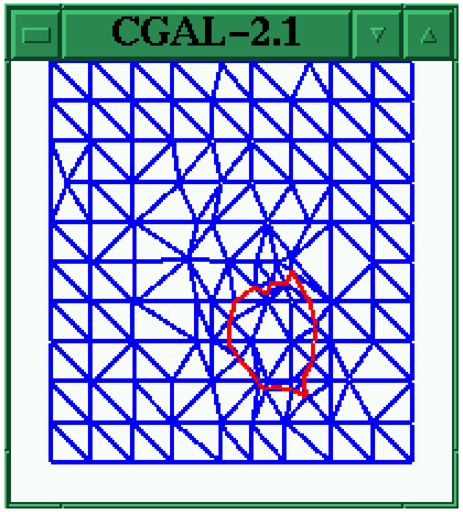
Point, orientation test, in_circle test
Segment, Triangle
Line, Ray, circumcenter, bisector

Default traits class : Triangulation_euclidean_traits_2<Rep>

Definition of a Delaunay triangulation of 2d points

```
typedef Cartesian<double> Rep;
typedef Triangulation_euclidean_traits_2<Rep> Traits;
typedef Delaunay_triangulation_2<Traits> Terrain;
```





Use of traits class : Delaunay terrain

Traits class for a terrain

3D points
orientation
in_circle

on x and y coordinates

▷ Triangulation_euclidean_traits_xy_3<Rep>

Definition of a Terrain

typedef Cartesian<double> Rep;
typedef Triangulation_euclidean_traits_xy_3<Rep> Traits;
typedef Delaunay_triangulation_2<Traits> Triangulation;

Convex hulls

2D, 3D

- static (quickhull) and dynamic convex hulls (triangulation)
- Extremal points
- Strongly convexity checking

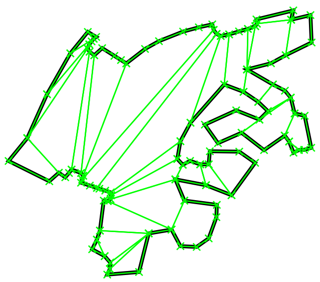
Dim d

incremental convex hull $\Longrightarrow$ (d-1)-Delaunay triangulation
nearest site Voronoi diagram
furthest site Voronoi diagram

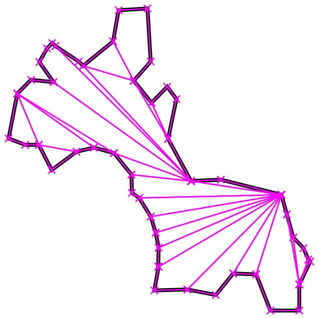
Polygons

Fonctionality : area, orientation
convexity test, simplicity test
point location

Partioning : Convex or monotone partitioning



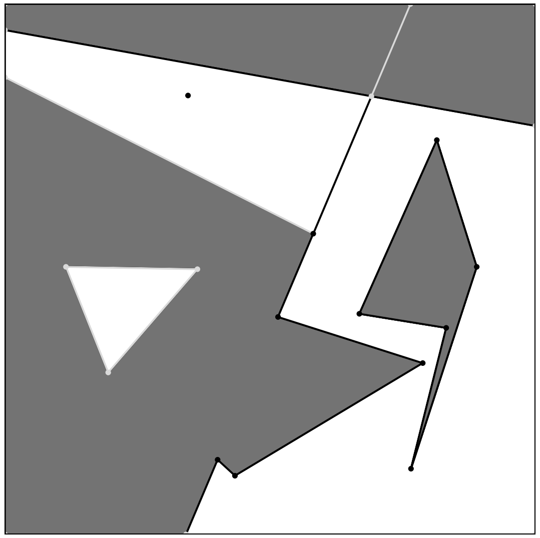
Optimal convex partition



Approx. optimal convex partition

2D Nef polyhedra : boolean operations on polygon

Nef polyhedron : a set obtained from
- a finite set of open halfspaces
- intersections and complements



Functionality
- boolean operations
- interior, boundary, closure
- set regularisation

Planar Map

Planar_map_2<Dcel,Traits>

Planar subdivision induced by
- a set of x-monotone curves
- intersecting only at their endpoints

Fonctionalities

insertion , suppression

localization

vertical ray shooting

Traits class Requirements

Point

x_monotone curve

xy_comparisons

Arrangements

Planar subdivision induced by a set of curves
Can deal with non monotone and intersecting or overlapping curves

Arrangement = Planar map + Curve hierarchy

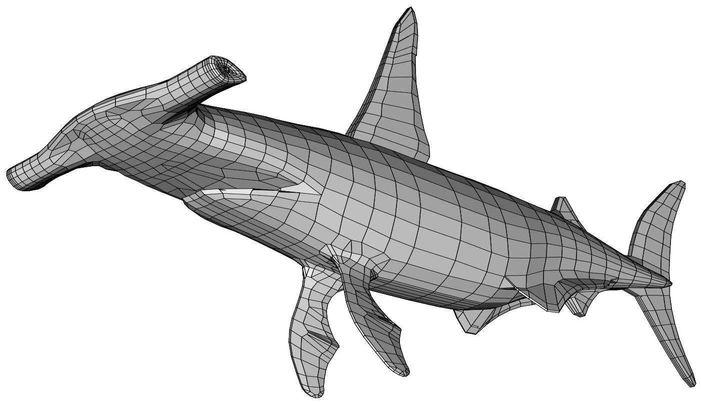
Traits classes exist for :
segments, circular arcs, polyline

Polyhedral surfaces

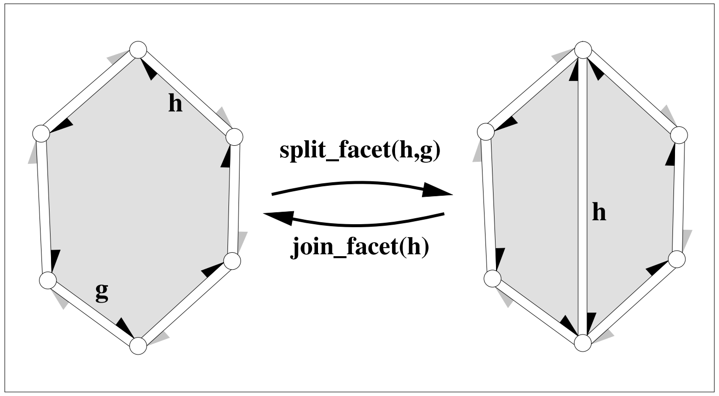
Combinatorial structure :
- Constructions
- Euler operations
- no check of geometry

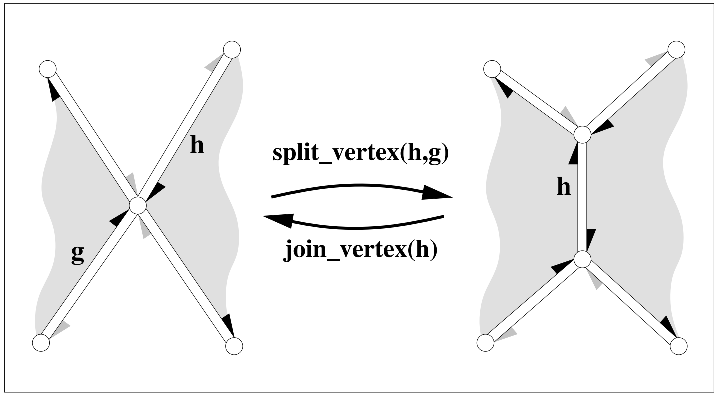
Traits class :
Point, Plane

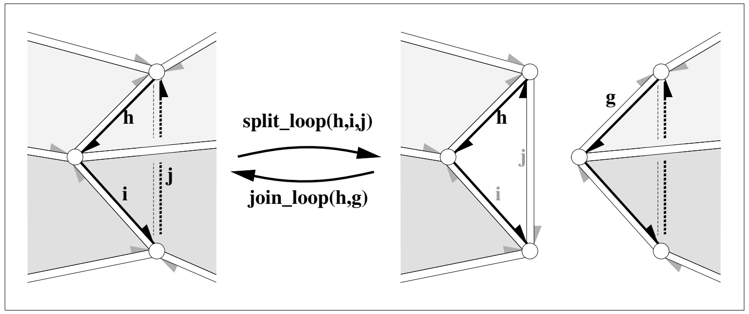
Polyhedron_3<Traits,DCEL>



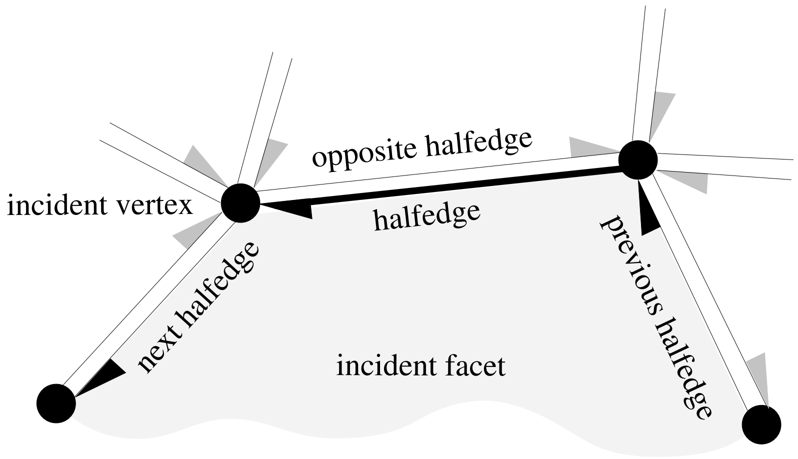
Euler operations on polyhedral surfaces







Halfedge data structure

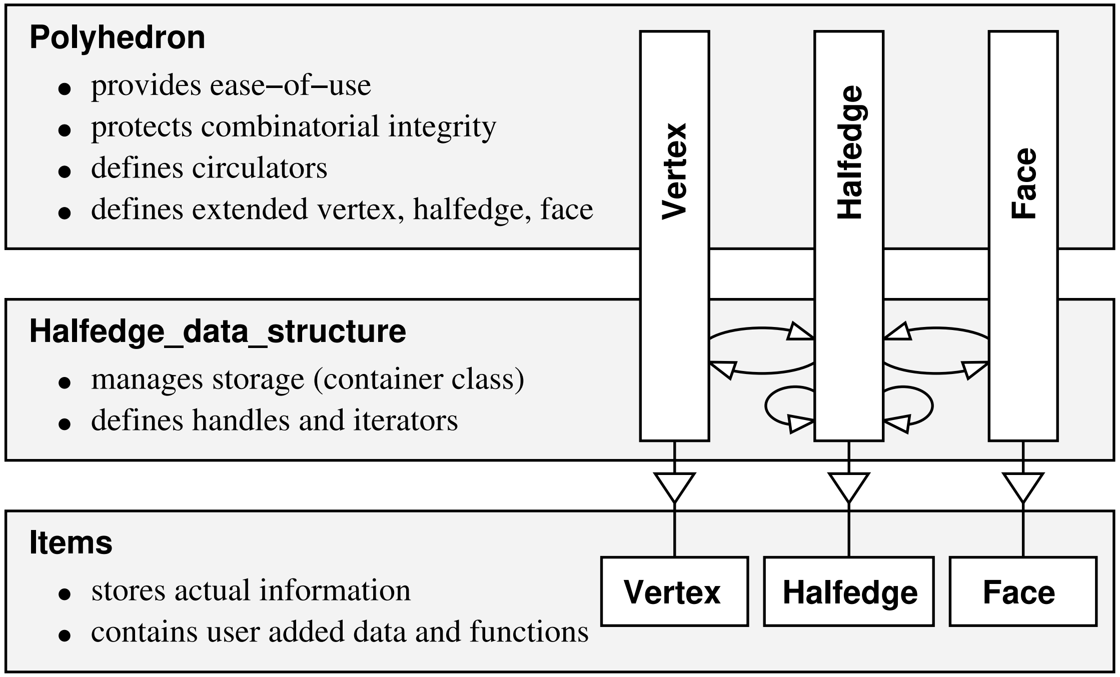


Halfedge
Halfegde opposite()
Halfedge next()
Halfedge prev()
Vertex vertex()
Facet facet()

Vertex
Halfedge halfedge()
Point& point()

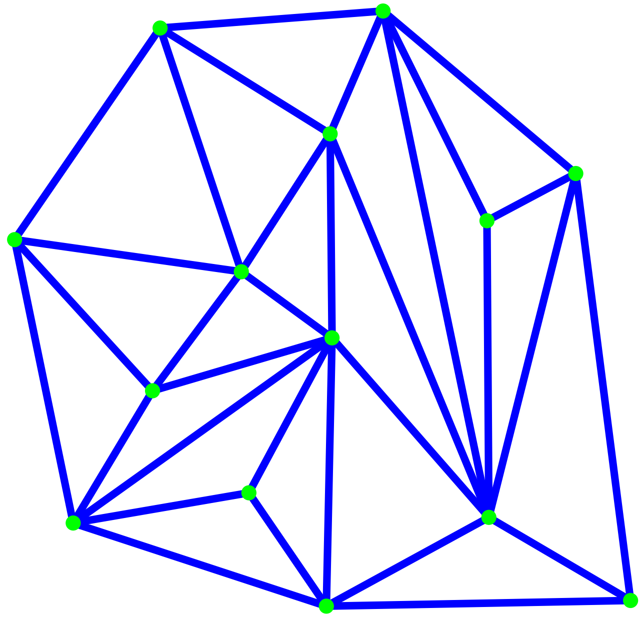
Facet
Halfedge halfedge()
Normal& normal()
Plane& plane()

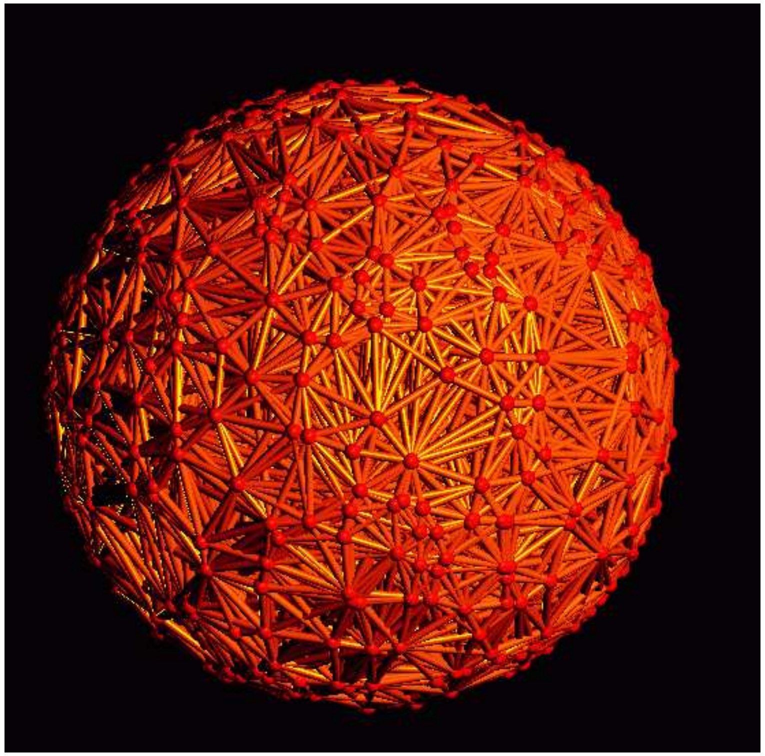
Polyhedral surfaces design



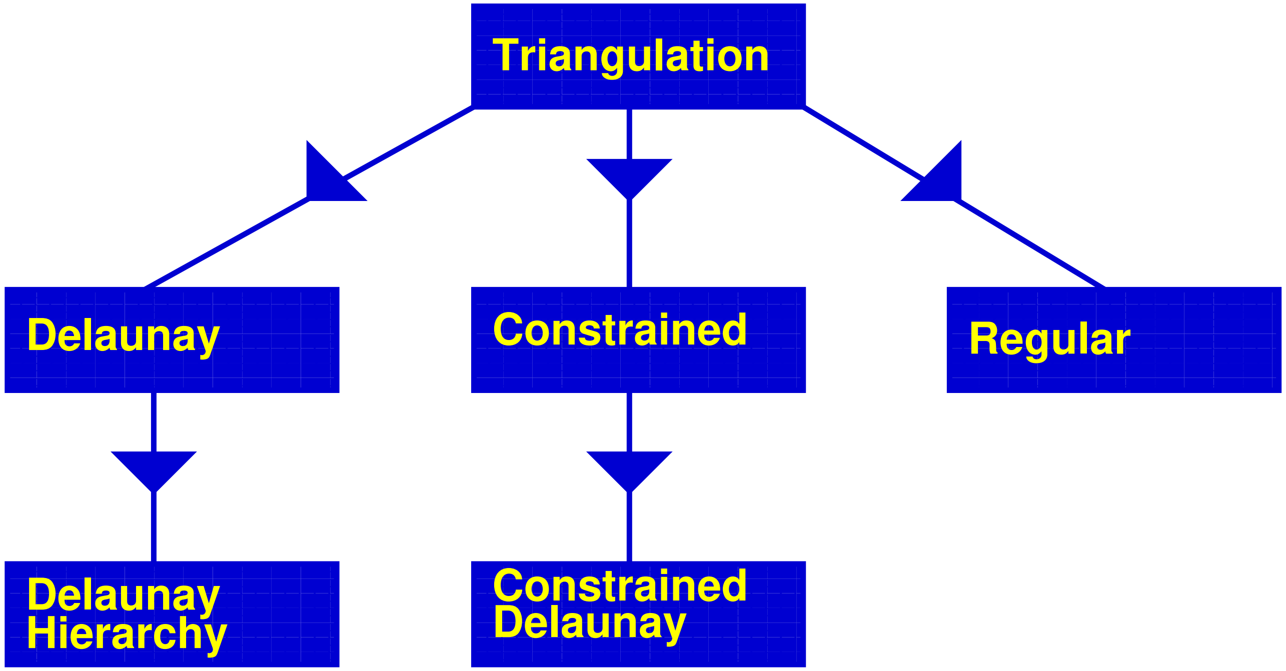
Flexibility : types Vertex, Halfedge and Face can be changed

2D and 3D Triangulations

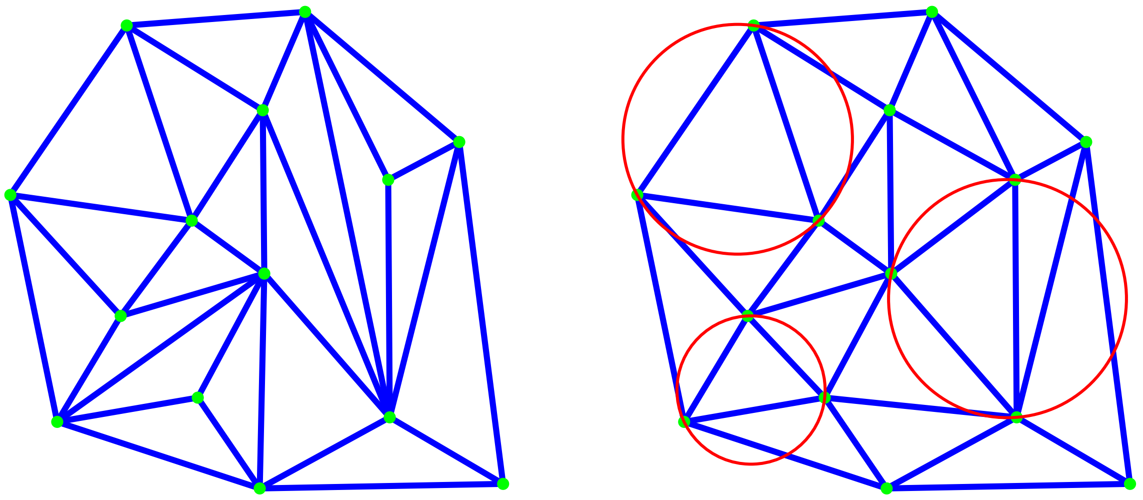




The Triangulations derivation tree



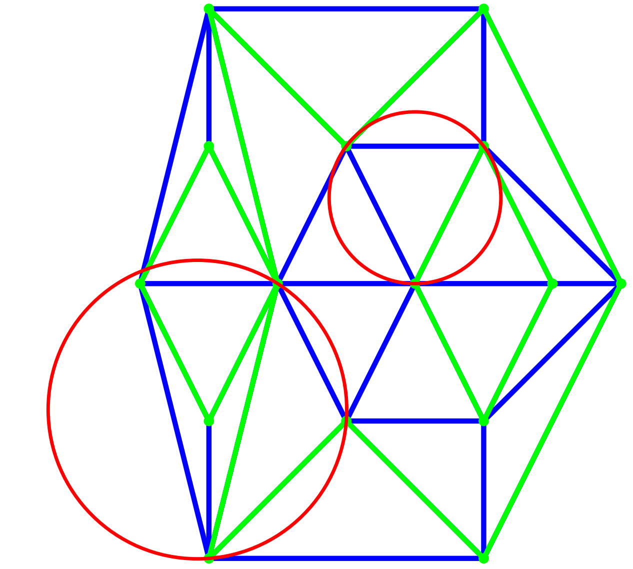
Basic and Delaunay triangulations



Basic triangulations : lazy incremental construction

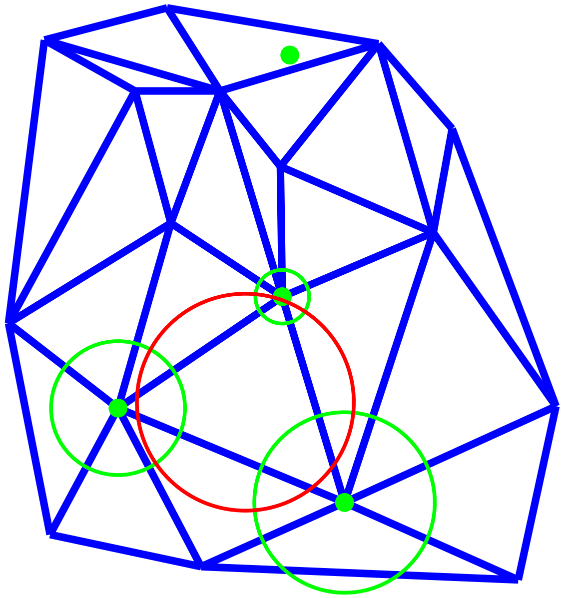
Delaunay triangulations: empty circle property

Constrained triangulations - Constrained Delaunay triangulations



Constrained triangulations : enforced edges
Constrained Delaunay triang : enforced edges +
constrained empty circle property

Regular triangulations



generalisation of Delaunay triangulations :

- defined for a weighted points
- dual of the power diagram.

Functionalities of Cgal triangulations

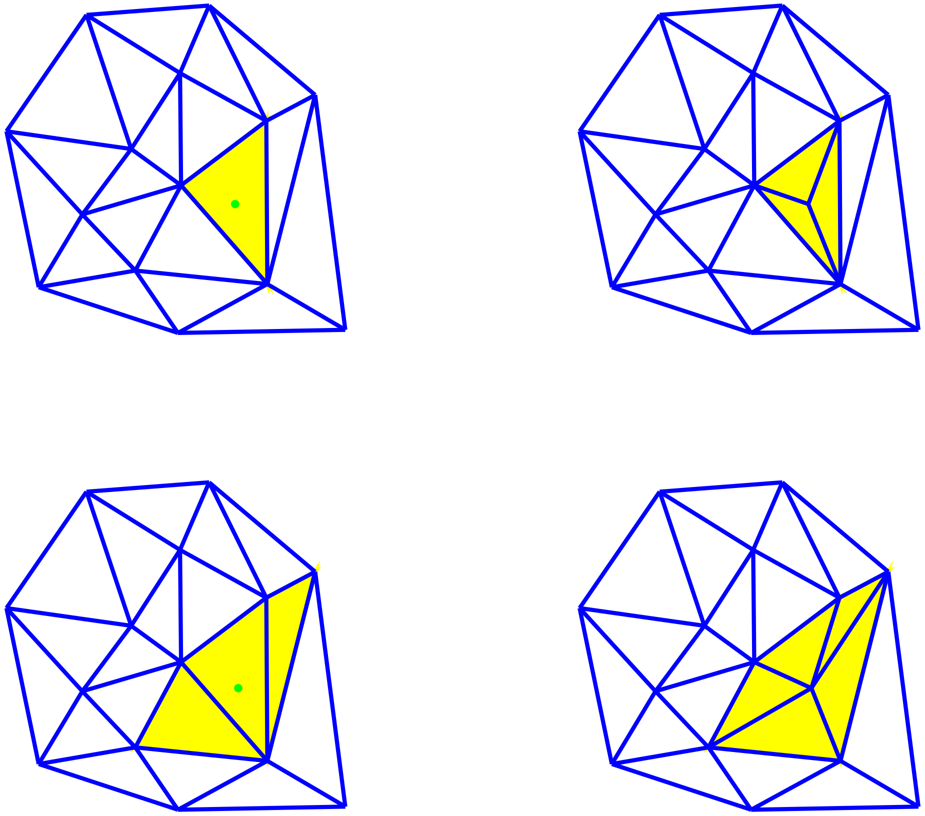
Insertion, suppression

Point location

Traversal of a triangulation

- passing from a face to its neighbors
- iterators to visit all or faces of a triangulation
- circulators to visit all faces around a vertex(2d) or an edge(3d)
- circulators to visit all faces all faces intersected by a line.

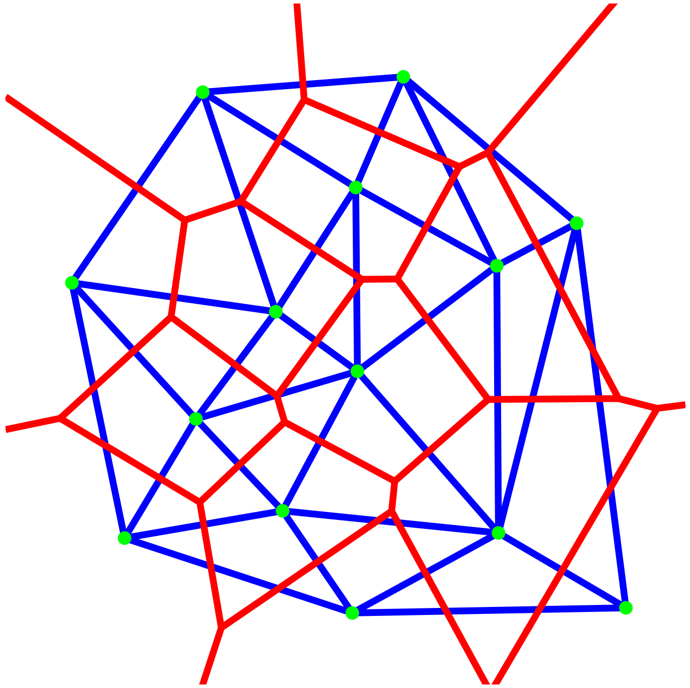
Insertion, removal



Additional functionalities for Delaunay triangulations

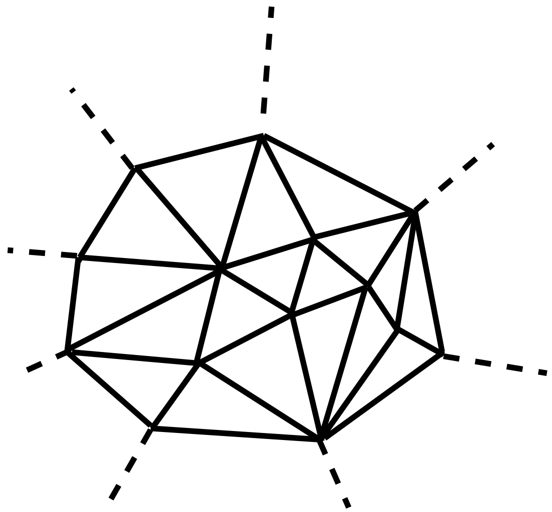
Nearest neighbor queries

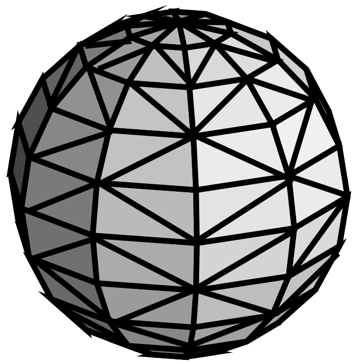
Voronoi diagram



Triangulations

Set of Faces

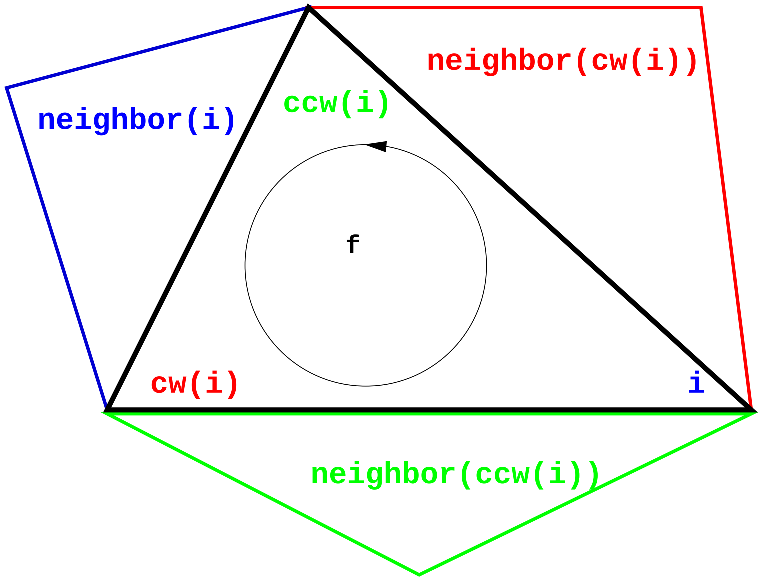




Triangulations in Cgal always covers the whole convex hull of their vertices.
Use constrained triangulations to triangulate polygonal regions.

A fictitious vertex called the *infinite vertex* is added.
Any face is a triangle.
Any edge is incident to two faces.
The set of faces $\equiv$ a 2d topological sphere

The representation is based on faces and vertices.
Edges are only implicitely represented



Vertex

Face* v_face

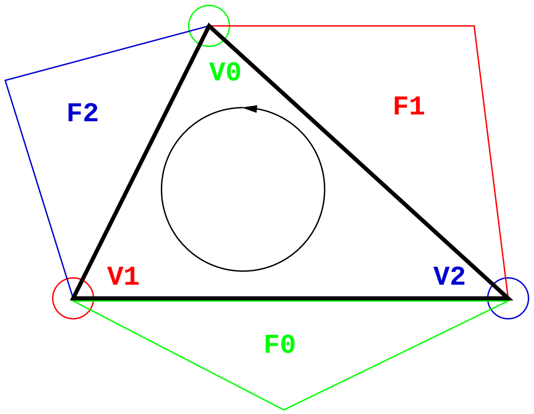
Face

Vertex* $vertex[3]$

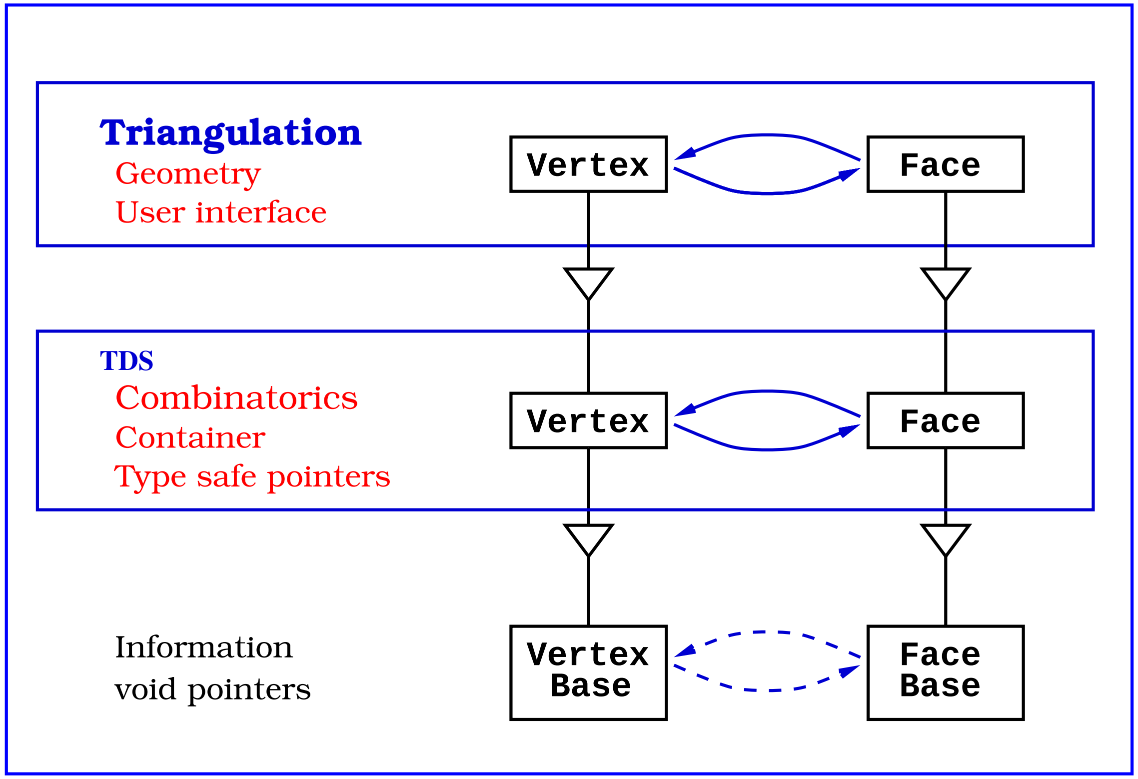
Face* $neighbor[3]$

Representation cont'd

n	vertices
$3n-6$	edges
$2n-4$	faces



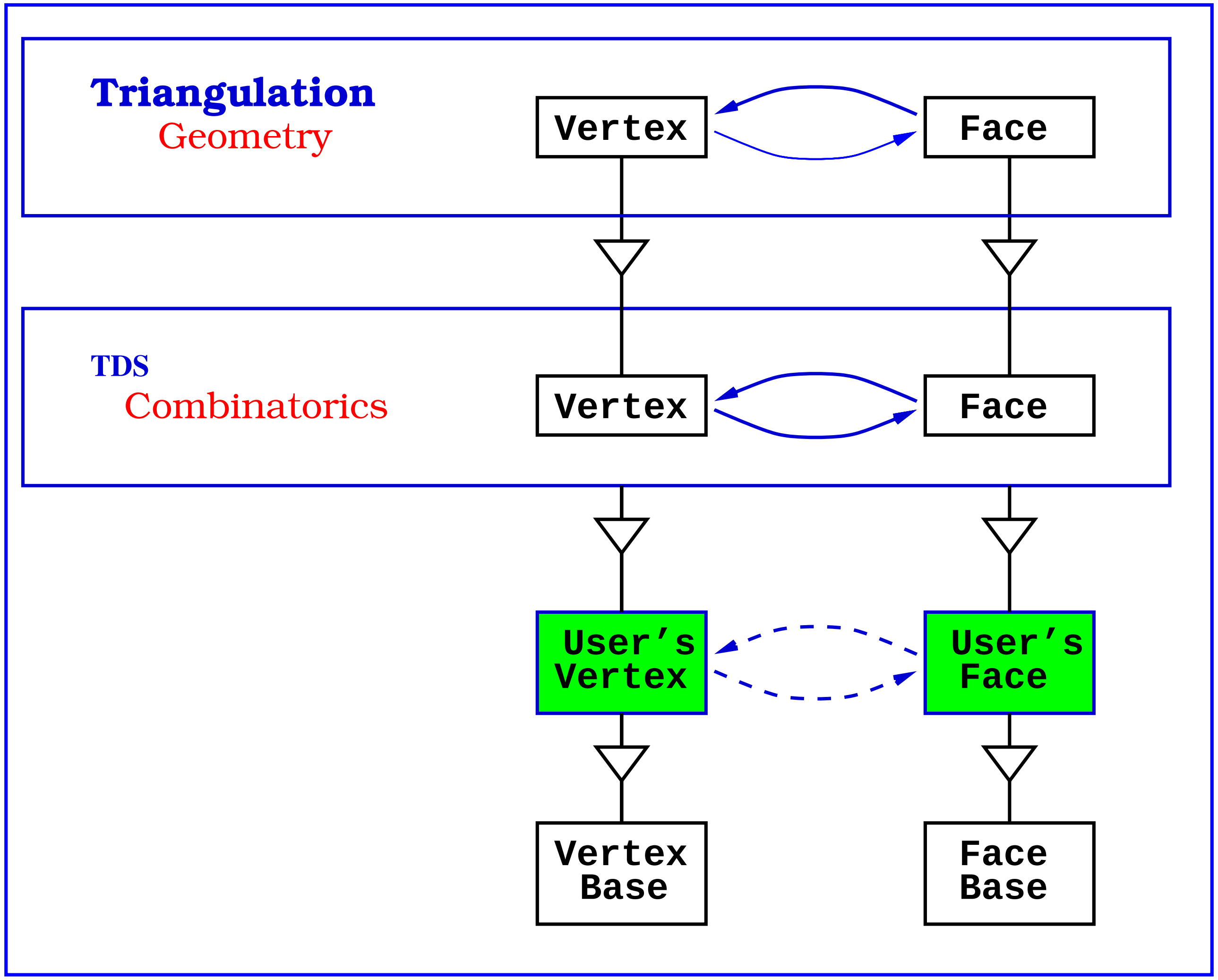
	DCEL	CGAL Tds
Vertex	n	n
Edges	$4 \times 2 \times (3n-6)$	
Faces	$(2n-4)$	$6 \times (2n-4)$
Total	$27n$	$13n$



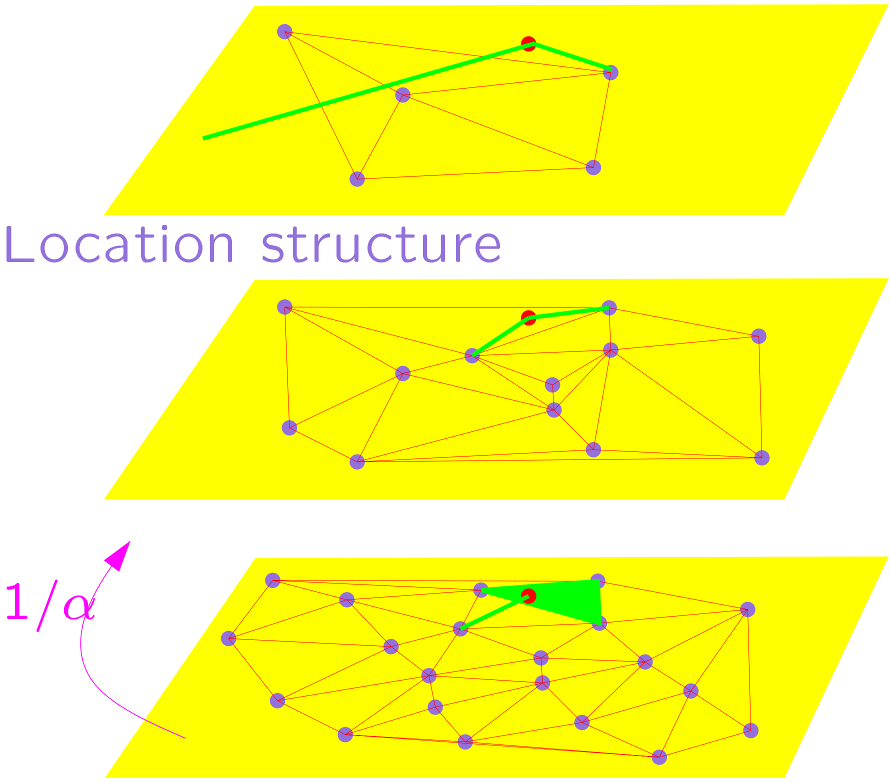
template< class Gt, class Tds> Triangulation_2 ;
template<class Vb, class Fb> Triangulation_data_structure_2;

Triangulations

Software design



Localisation



The Delaunay hierarchy

Geometric Optimisation

Minimum enclosing circle
Minimum enclosing ellipse
Smallest enclosing sphere dDim
Maximal area inscribed k-gons
Maximal perimeter inscribed k-gons
Extremal inscribed k-gons
Min_annulus_d
Polytope_distance_d
Width_3

To come soon

LP solver
QP solver
LPtype solver

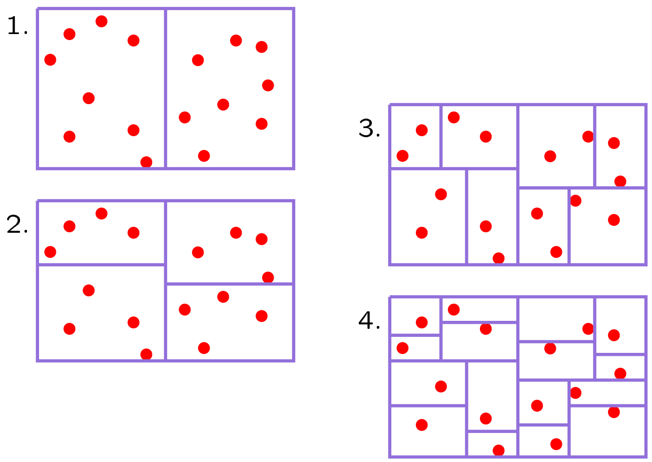
Search structures

Pointset_2 based on Delaunay triangulation

circular, triangular, isorectangular range search

k-nearest neighbor

KD-Trees

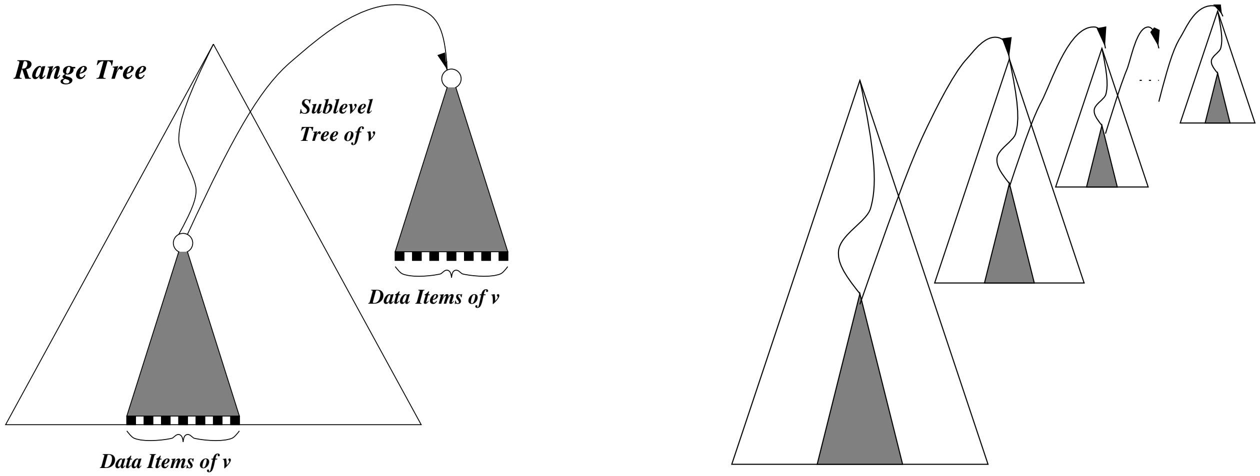


Space $O(nlogn))$

Query time $O(n^{1-\frac{1}{d}}))$

Search structures (cont'd)

Multilayer range-segment trees



Space $O(n \log^{d-1} n)$

Query time $O(\log^d n)$

Support Library

STL extension

Assertions

Circulators

Handles

Random Generators

Random point sets

Random convex point sets

I/O /Visualisation

Geomview_stream

Geowin

Qt_Widget

For CGAL Users

Documentation

- User's maunal (pdf, ps, html)
- Reference Manual (pdf, ps, html)

Web site

http://www.cgal.org

User's mailing lists

cgal-announce-l@postino.mpi-sb.mpg.de

cgal-discuss-l@postino.mpi-sb.mpg.de

CGAL for Developpers

Documentation Developpers'Manual

Web site private pages http://www.cgal.org/Member

Developer's mailing lists cgal-develop-l@postino.mpi-sb.mpg.de

CVS Servor

Testsuite

- run twice a week
- on each supported platform

Installation of CGAL

▷ download the compressed archive file from the CGAL web site
http://www.cgal.org

▷ unpack the file
in the place in which you wish to install it

▷ [optional] read
INSTALL
installation doc in `doc_html`, `doc_ps` or `doc_pdf`

▷ run `./install_cgal -i`

▷ `[optional] get the CGALWin binary and put it into lib/<OS-compiler>.`