CGAL

Computational Geometry Algorithms Library

Monique Teillaud

www.cgal.org

- 2D, 3D, dD “rational” kernels
- 2D circular and 3D spherical kernels

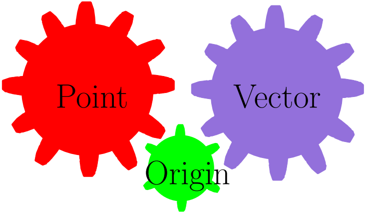
In the kernels

- Elementary geometric objects
- Elementary computations on them

Primitives	Predicates	Constructions
2D, 3D, dD	• comparison	• intersection
• Point	• Orientation	• squared distance
• Vector	• InSphere	. . .
• Triangle	. . .	
• Circle		
. . .		

Affine geometry

Point - Origin $\rightarrow$ Vector
Point - Point $\rightarrow$ Vector
Point + Vector $\rightarrow$ Point



Point + Point illegal

midpoint(a,b) = a + 1/2 x (b-a)

Kernels and number types

Cartesian representation

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

Homogeneous representation

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

Kernels and number types

Cartesian representation

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

Homogeneous representation

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

- ex: Intersection of two lines -

$$\left\{\begin{array}{l} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{array}\right.$$

$$\left\{\begin{array}{l} a_1 hx + b_1 hy + c_1 hw = 0 \\ a_2 hx + b_2 hy + c_2 hw = 0 \end{array}\right.$$

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

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Kernels and number types

Cartesian representation

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

Homogeneous representation

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

- ex: Intersection of two lines -

$$\left\{ \begin{array}{l} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{array} \right.$$

$$\left\{ \begin{array}{l} a_1 hx + b_1 hy + c_1 hw = 0 \\ a_2 hx + b_2 hy + c_2 hw = 0 \end{array} \right.$$

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

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Field operations

Ring operations

The "rational" Kernels

```
CGAL::Cartesian< FieldType >
CGAL::Homogeneous< RingType >
```

⟶ Flexibility

```
typedef double                          NumberType;
typedef Cartesian< NumberType >         Kernel;
typedef Kernel::Point_2                 Point;
```

Arithmetic robustness issues

Rational Kernels:
Predicates = signs of polynomial expressions

Exact Geometric Computation
$\neq$ **exact arithmetics**

Predicates evaluated exactly

Filtering Techniques (interval arithmetics, etc)
exact arithmetics only when needed

`CGAL::Exact_predicates_inexact_constructions_kernel`

Arithmetic robustness issues

```
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

Kernel::Point_2 p(0,0), q(sqrt2,sqrt2);
Kernel::Circle_2 C(p,2); // squared radius 2
```

Arithmetic robustness issues

```
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt( NT(2) );

Kernel::Point_2 p(0,0), q(sqrt2,sqrt2);
Kernel::Circle_2 C(p,2); // squared radius 2

assert( C.has_on_boundary(q) );
```

OK if NT gives exact sqrt
assertion violation otherwise

The circular/spherical kernels

Circular/spherical kernels

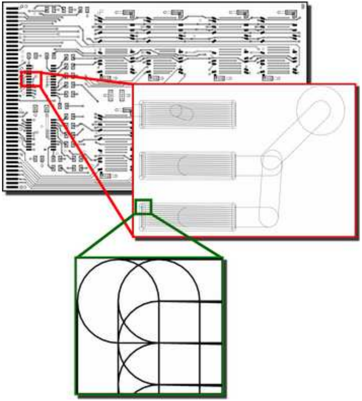
- solve needs for e.g. intersection of circles.
- extend the CGAL (linear) kernels

Exact computations on algebraic numbers of degree 2
= roots of polynomials of degree 2

Algebraic methods reduce comparisons to
computations of signs of polynomial expressions

Application of the 2D circular kernel

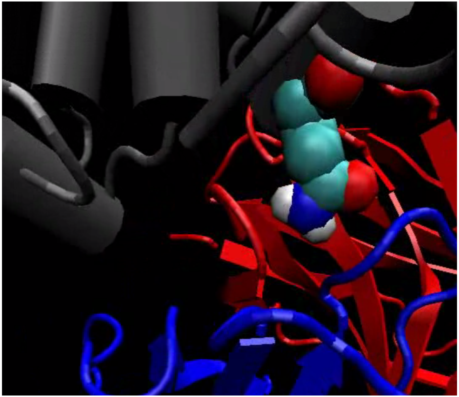
Computation of arrangements
of 2D circular arcs and line segments



Pedro M.M. de Castro, Master internship

Application of the 3D spherical kernel

Computation of arrangements of 3D spheres



Sébastien Loriot, PhD thesis

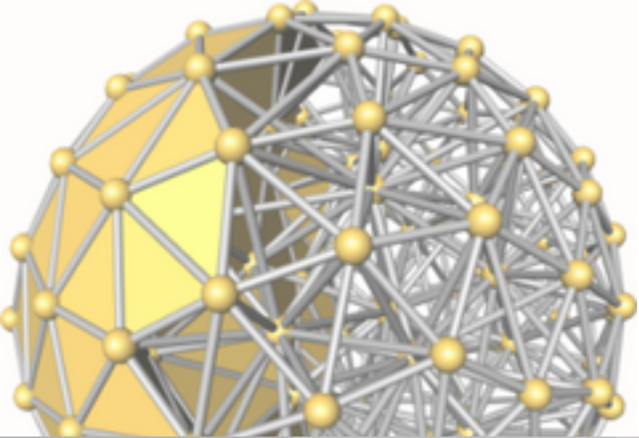
2D, 3D Triangulations in CGAL

2D, 3D Triangulations in CGAL — Introduction

Simplicial complex

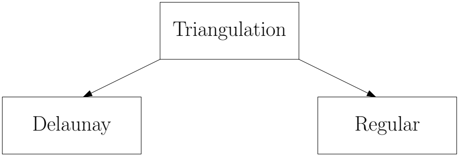
= set $\mathbb{K}$ of 0,1,2,. . . d-faces such that

- each face is a simplex
- $\sigma \in \mathbb{K}, \tau \leq \sigma \Rightarrow \tau \in \mathbb{K}$
- $\sigma, \sigma' \in \mathbb{K} \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$



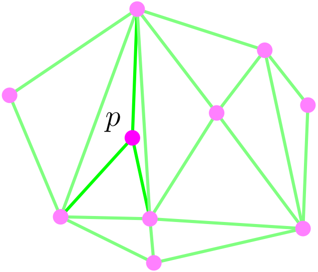
Various triangulations

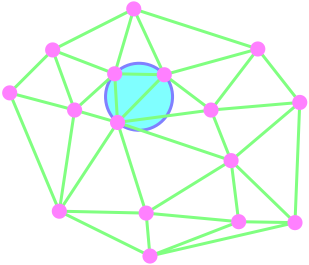
2D, 3D, dD Basic triangulations
2D, 3D, dD Delaunay triangulations
2D, 3D, dD Regular triangulations



Basic and Delaunay triangulations

(figures in 2D)





Basic triangulations : incremental construction

Delaunay triangulations: empty sphere property

2D, 3D Triangulations in CGAL — Functionalities

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells) incident to a vertex
- circulators to visit all cells around an edge

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells)
  incident to a vertex
- circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips

General functionalities

- Traversal of a 2D (3D) triangulation

- passing from a face (cell) to its neighbors
- iterators to visit all faces (cells) of a triangulation
- circulators (iterators) to visit all faces (cells)
  incident to a vertex
- circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips

- is_valid
  checks local validity (sufficient in practice)
  not global validity

Traversal of a 3D triangulation

Iterators

All_cells_iterator	Finite_cells_iterator
All_faces_iterator	Finite_faces_iterator
All_edges_iterator	Finite_edges_iterator
All_vertices_iterator	Finite_vertices_iterator

Circulators

Cell_circulator : cells incident to an edge
Facet_circulator : facets incident to an edge

```
All_vertices_iterator vit;
for (vit = T.all_vertices_begin();
      vit != T.all_vertices_end(); ++vit)
            ...
```

Traversal of a 3D triangulation

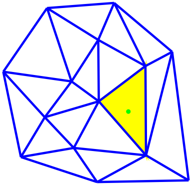
Around a vertex

incident cells and facets, adjacent vertices

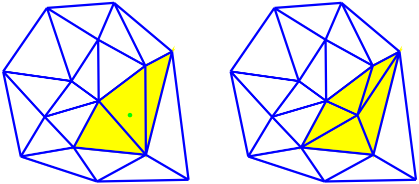
```
template < class OutputIterator >
OutputIterator
     t.incident_cells
          ( Vertex_handle v, OutputIterator cells)
```

Point location, insertion, removal

basic triangulation:

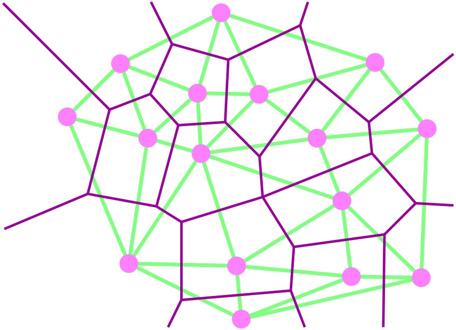


Delaunay triangulation :



Additional functionalities for Delaunay triangulations

Nearest neighbor queries
Voronoi diagram



2D, 3D Triangulations in CGAL — Representation

The main algorithm

Incremental algorithm

- fully dynamic (point insertion, vertex removal)
- any dimension
- easier to implement
- efficient in practice
- ...

Needs

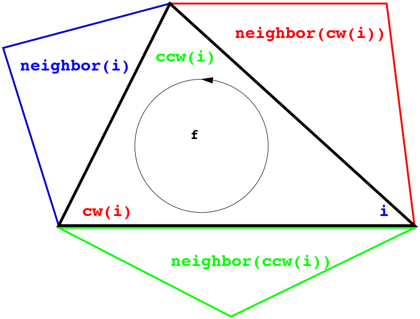
Walking in a triangulation

Access to

- vertices of a simplex
- neighbors of a simplex

in constant time

2D - Representation based on faces



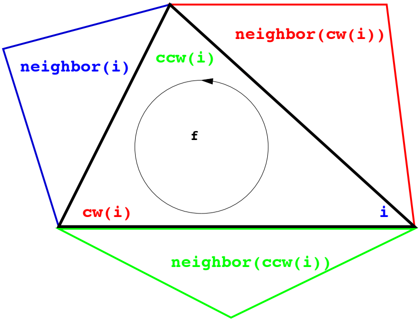
Vertex

Face_handle *v_face*

Face

Vertex_handle *vertex*[3]
Face_handle *neighbor*[3]

2D - Representation based on faces



Vertex

Face_handle *v_face*

Face

Vertex_handle *vertex*[3]
Face_handle *neighbor*[3]

Edges are implicit: `std::pair<` f, i `>`
where f = one of the two incident faces.

more efficient than half-edges

From one face to another

n = f → neighbor(i)
j = n → index(f)

Geometry vs. combinatorics

Each finite vertex stores a point

Arithmetic robustness

see above

Benchmarks
2.3 GHz, 16 GByte workstation
CGAL 3.9 (Release mode)

Arithmetic robustness

see above

Benchmarks

2.3 GHz, 16 GByte workstation

CGAL 3.9 (Release mode)

Delaunay triangulation - 10 Mpoints

Kernel	2D	3D
Cartesian < double >	9.7 sec	75 sec
Exact_predicates_inexact_constructions_kernel	10.6 sec	82 sec

Arithmetic robustness

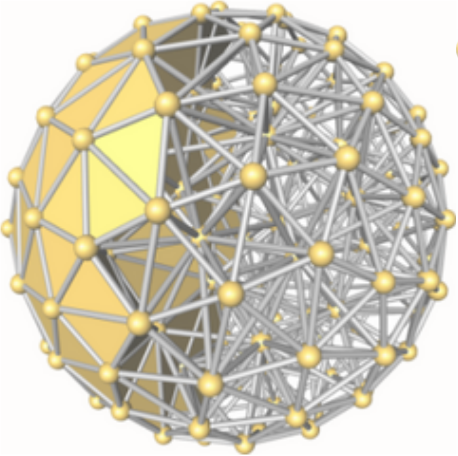
see above

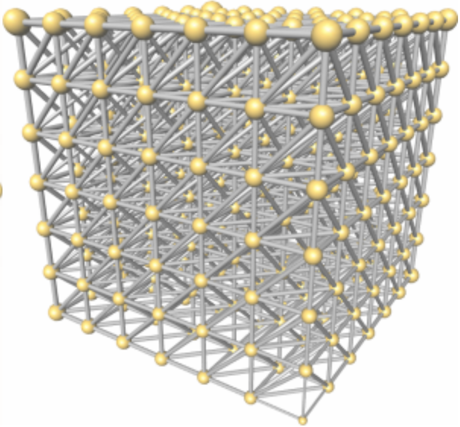
Benchmarks
2.3 GHz, 16 GByte workstation
CGAL 3.9 (Release mode)

Delaunay triangulation - 10 Mpoints

Kernel	2D	3D
`Cartesian < double >` may loop (or crash) !		
`Exact_predicates_inexact_constructions_kernel`	10.6 sec	82 sec

Robustness





Pictures by Pierre Alliez

2D, 3D Triangulations in CGAL — Software Design

Traits class

Triangulation_2<Traits, TDS>

Geometric traits classes provide:

Geometric objects + predicates + constructors

Flexibility:

- The Kernel can be used as a traits class for several algorithms
- Otherwise: Default traits classes provided
- The user can plug his/her own traits class

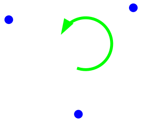
Traits class

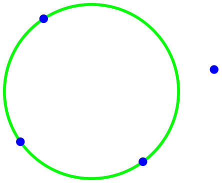
Generic algorithms

`Delaunay_Triangulation_2<Traits, TDS>`

Traits parameter provides:

- Point
- orientation test, in_circle test



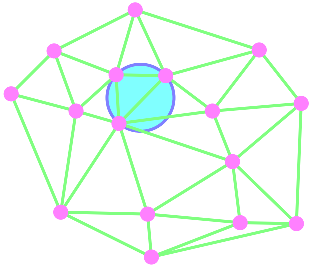


Traits class

2D Kernel used as traits class

```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2< K > Delaunay;
```

- 2D points: coordinates (x, y)
- orientation, in_circle

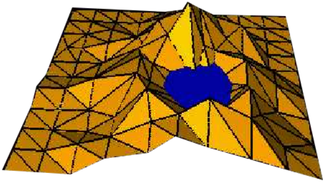


Traits class

Changing the traits class

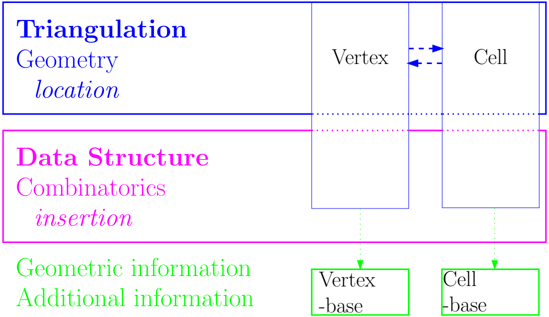
```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Projection_traits_xy_3< K > Traits;
typedef CGAL::Delaunay_triangulation_2< Traits > Terrain;
```

- 3D points: coordinates (x, y, z)
- orientation, in_circle:
  on x and y coordinates only



Layers

Triangulation_3< Traits, TDS >



Triangulation_data_structure_3< Vb, Cb> ;
Vb and Cb have default values.

Layers

The base level

Concepts VertexBase and CellBase.

Provide

- Point + access function + setting
- incidence and adjacency relations (access and setting)

Several models, parameterised by the traits class.

CGAL

demos

web site www.cgal.org