Triangulations in CGAL

Monique Teillaud

INRIA
SOPHIA ANTIPOLIS

Outline

- Definition - representation
- Design
- Traits class
- Triangulation data structure
- Degenerate dimensions
- Functionalities - algorithms
- Derived classes
- Future

Packages

2D package

3D package : To appear in next (?) public release

Definitions

Representation

Triangulation

S set of points

Partition of the convex hull

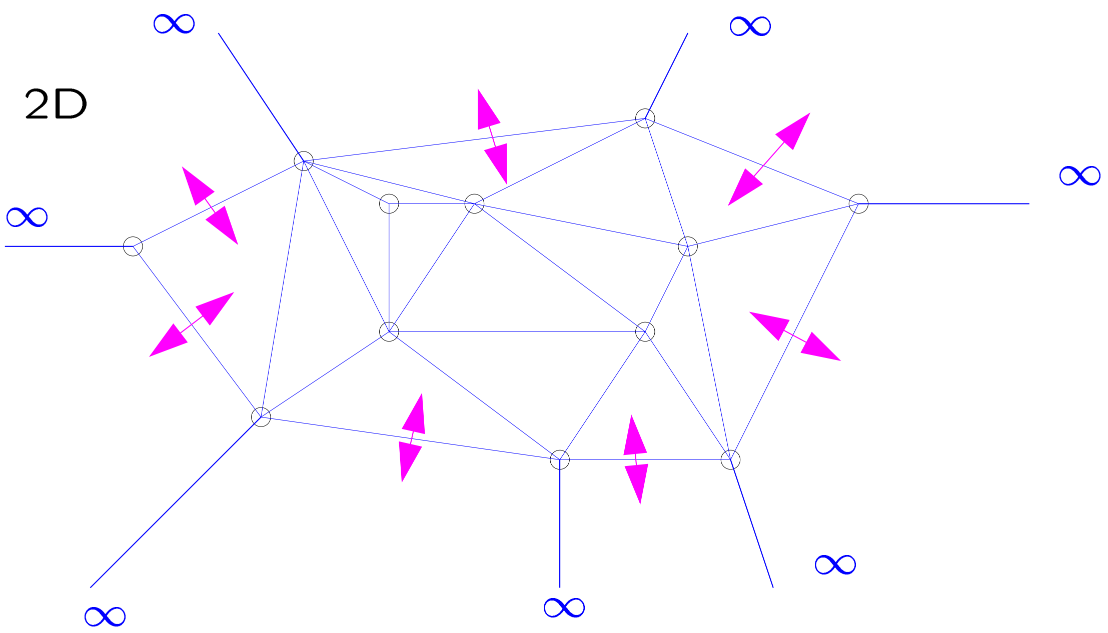
triangulation with boundary
outside : infinite cell

Partition of the whole space

triangulation without boundary

additional vertex : ∞

infinite cell triangulated

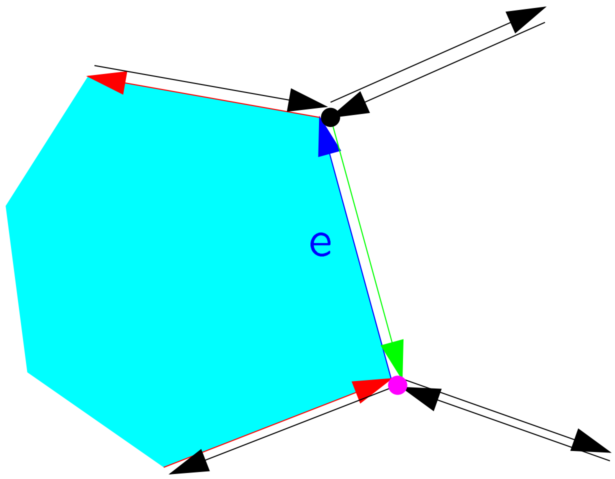


Representation

2D

DECL

Doubly connected edge list



Vertex

Halfedge* *v_edge*

Face

Halfedge* *f_edge*

Edge

Face* *fg*

Vertex* *source*

Halfedge* *opposite*

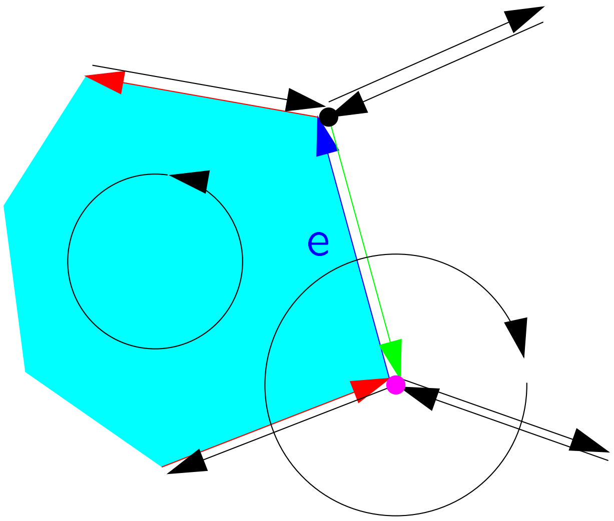
Halfedge* *prev*

Halfedge* *next*

Representation

2D

DECL



Half-edges incident to a face `f`:

`e = f_edge(f)`

counterclockwise	`suiv(e)`
clockwise	`prev(e)`

Half-edges incident to a vertex `v`:

`e = v_edge(v)`

counterclockwise	`opposite(prev(e))`
clockwise	`suiv(opposite(e))`

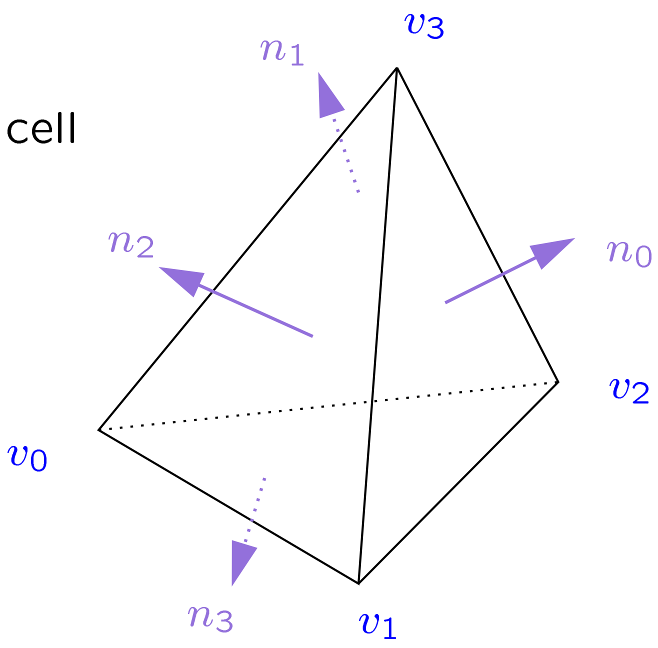
Representation

CGAL

Based on cells and vertices   Facets and edges implicit

- **Cell :**
  four vertices
  four neighbors

- **Vertex :**
  one incident cell



Adjacency relations
Incidence relations

Comparison

2D Triangulation

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

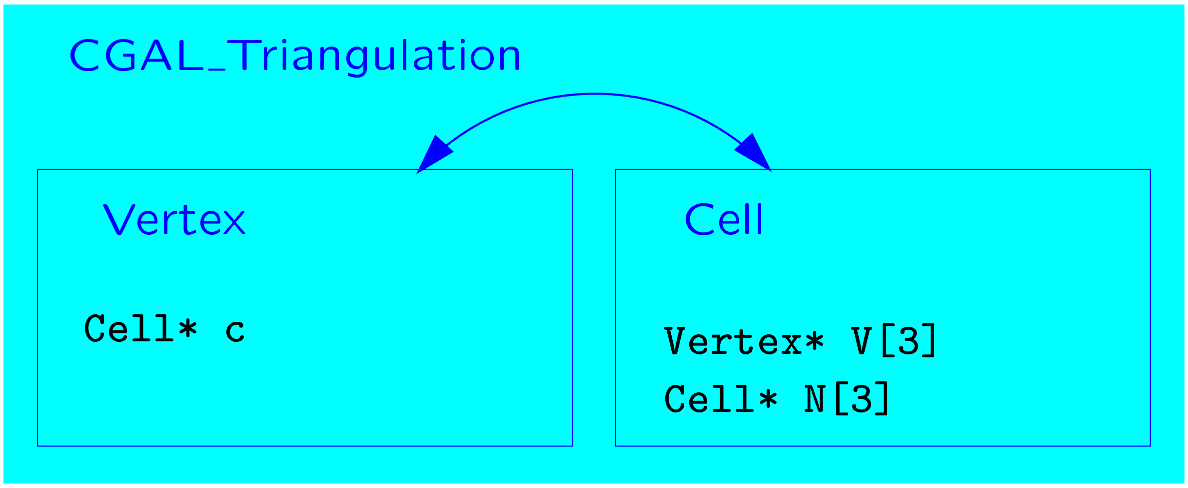
Storage

	DCEL	CGAL
Vertices	n	n
Edges	$4.2.(3n-6)$	
Faces	$2n-4$	$6.(2n-4)$
Total	$27n$	$13n$

3D G-maps

Design

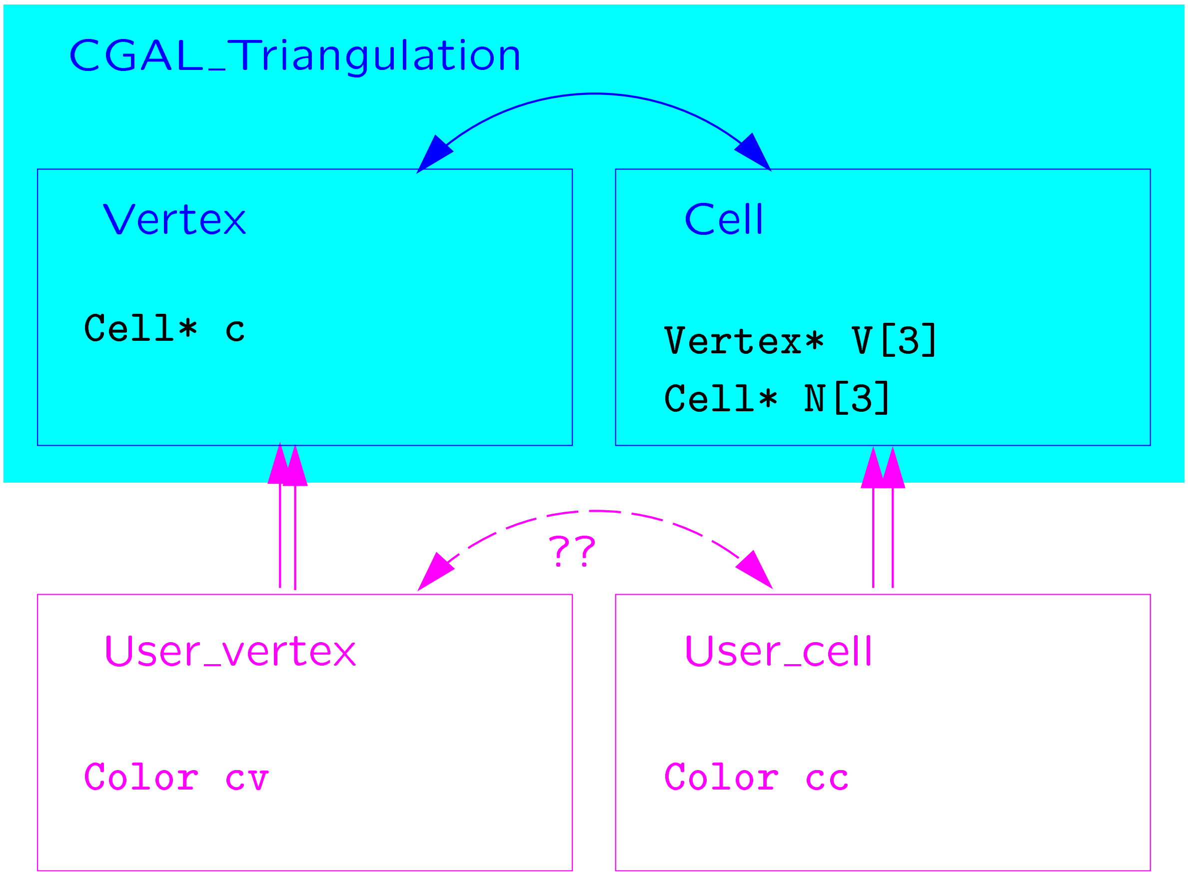
Naive solution



Naive solution

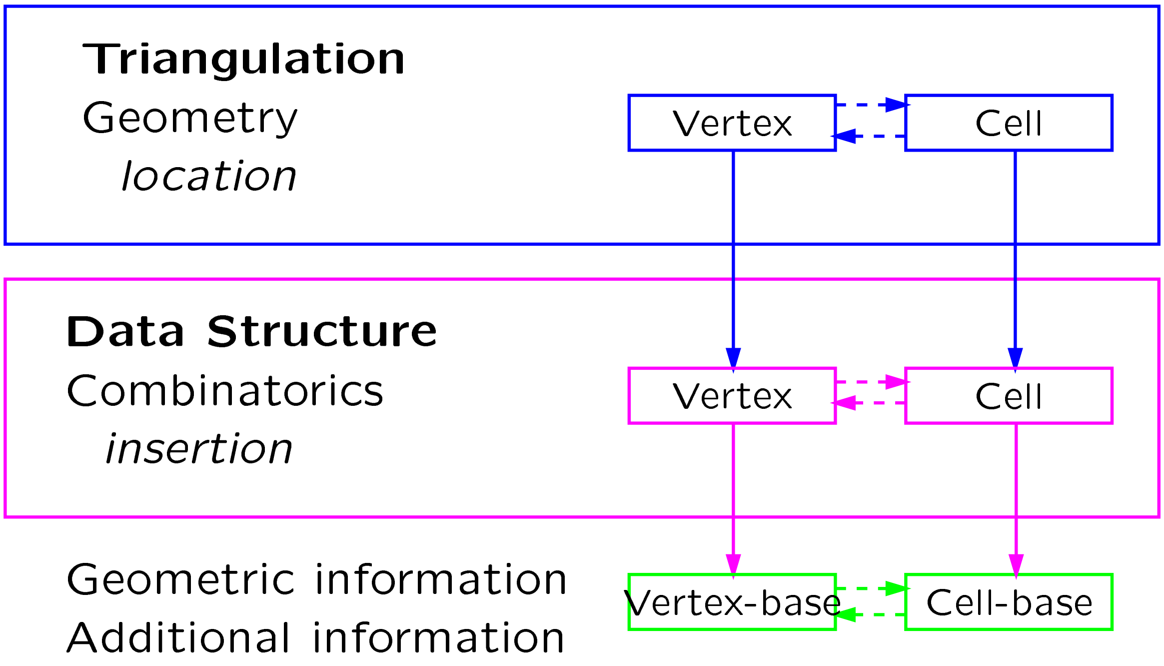
Problem

Triangulation with colored faces

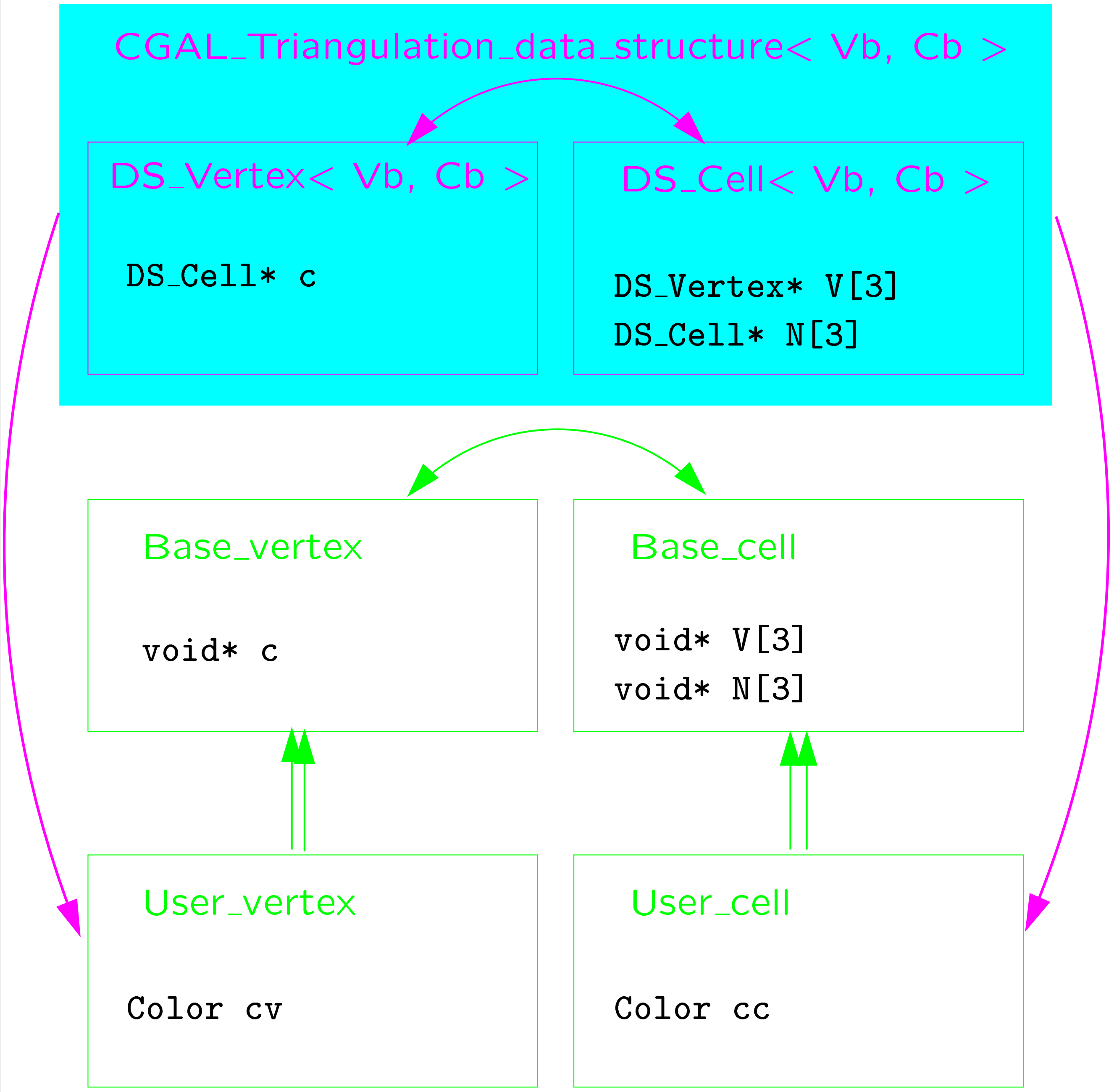


Design Overview

Three layers [Kettner]



Application **Colored cells**



Traits class

Definition

`CGAL_Triangulation_3< TDS, Traits >`

Geometric definitions

- Objects : points, segments, triangles, tetrahedra. . .
- Predicates :
  - orientation
  - in_circle, in_sphere
  - . . .

Traits class parameterized:

`CGAL_Triangulation_geom_traits< R >`

`R` representation class

Cartesian or homogeneous coordinates

Number type

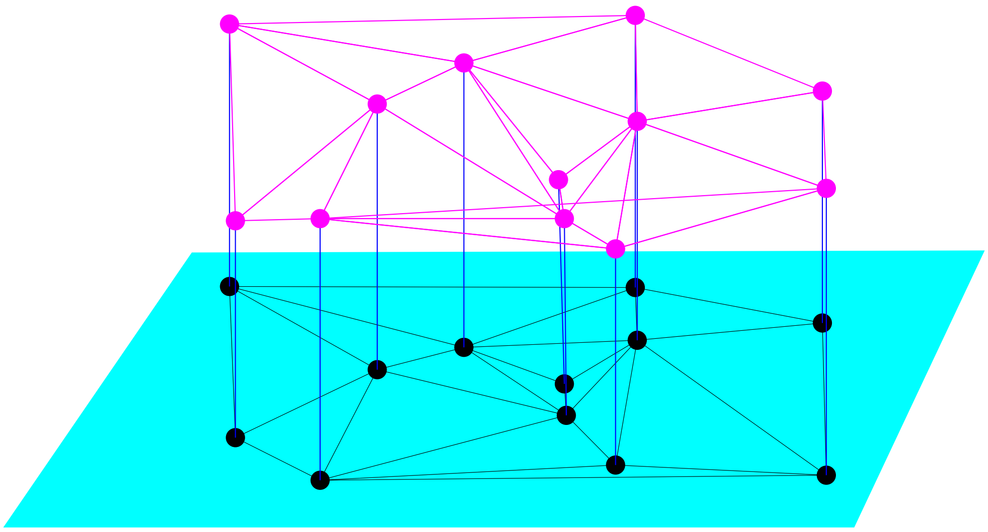
CODE

Writing a Traits class

Terrain in 3D

Data = 3D points (2D points with elevation)

Goal = 2D Delaunay triangulation
of the projected points



Writing a Traits class

Solution :

- Use `CGAL_Delaunay_triangulation_2`
- Write an adapted traits class

CODE

Triangulation

Data Structure

Triangulation data structure

3D case

```
CGAL_Triangulation_3< TDS, Traits >
```

TDS purely combinatorial

Triangulation of $\mathbb{R}^3$ $\rightleftarrows$ Triangulation of a sphere S^3 in $\mathbb{R}^4$

Triangulation data structure parameterized

```
CGAL_Triangulation_data_structure< Vb, Cb >
```

Infinite vertex

Previous solution (2D package)

`is_infinite` test in the `TDS`

New solution (3D package)

- Special vertex of the triangulation

  `is_infinite` test

  no significant coordinates

  no geometric calculations

- Ordinary vertex of the TDS

  `TDS` does not know which vertex
  is associated to the infinite vertex

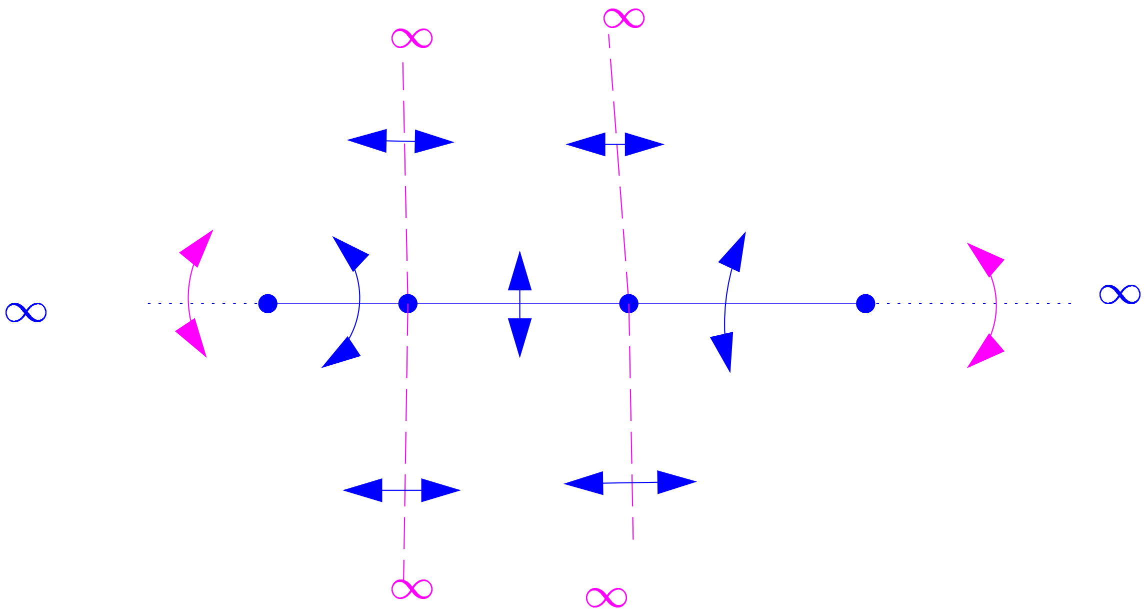
Degenerate dimensions

Representation

Previous solution

all points collinear : 1D triangulation in 2D

add the infinite vertex to all edges → 2d faces

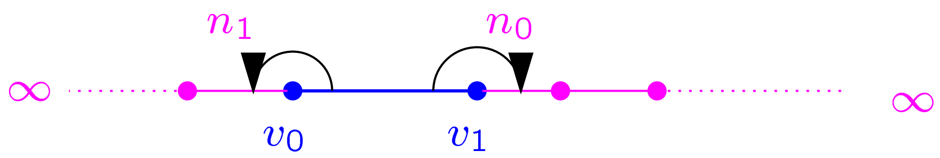



```
int i = c->index(n);
```
ambiguous

Representation

New solution

- 1D triangulation in 3D



vertices and neighbors 2 and 3 not used

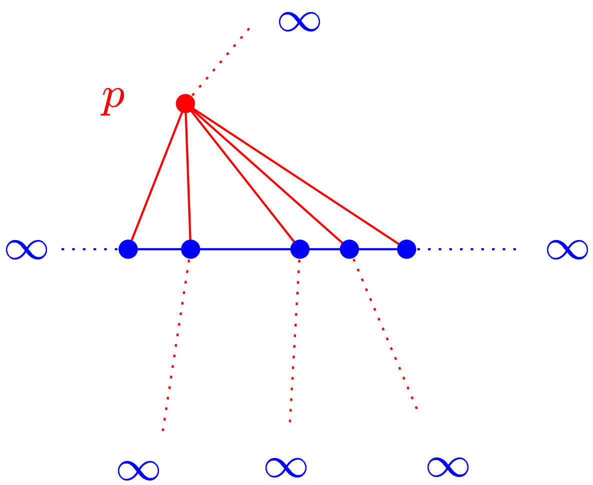
Cell* c
v_0, v_1, **NULL, NULL**
n_0, n_1, **NULL, NULL**

- 2D triangulation in 3D

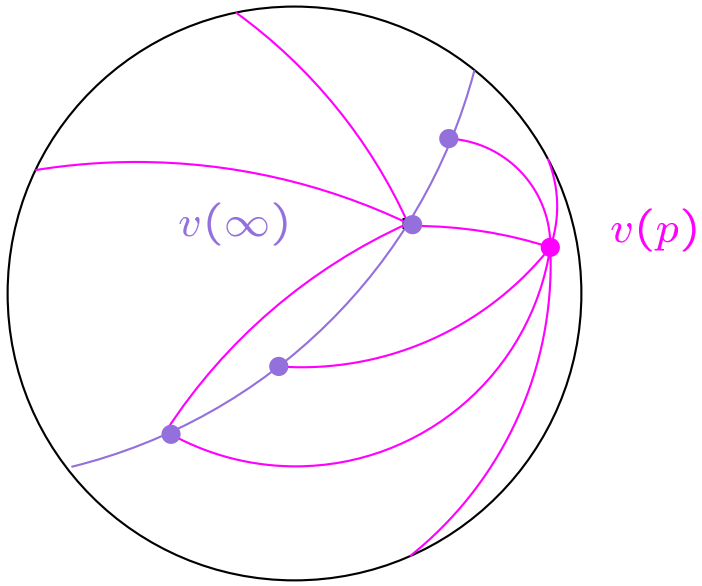
vertex and neighbor 3 not used

From $d = 1$ to $d = 2$

Triangulation



Triangulation data structure



Geometric triangulation

Triangulation data structure

$d \leq 3$

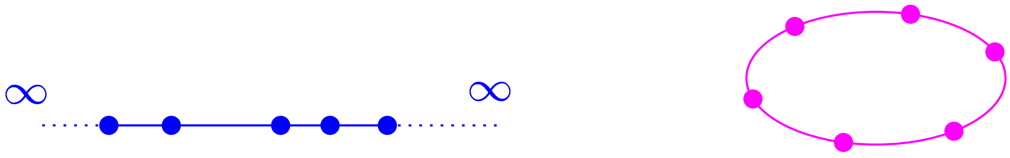
Triangulation of $\mathbb{R}^d$ $\rightleftarrows$ Triangulation of a sphere S^d in $\mathbb{R}^{d+1}$

convention

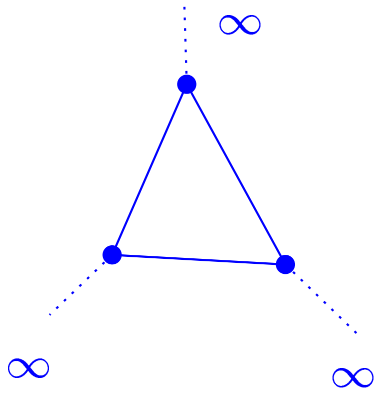
$d = -1$

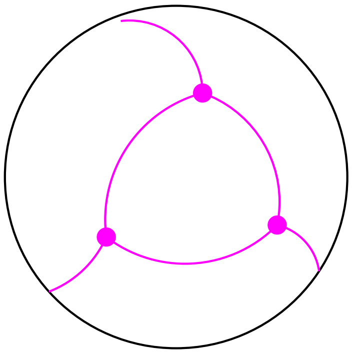
$d = 0$

$d = 1$

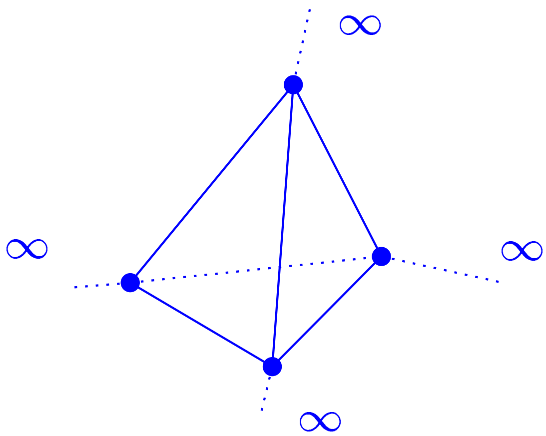


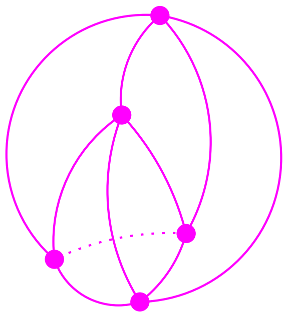
$d = 2$





$d = 3$





in $I\!R^4 \ldots$

Functionalities

Algorithms

- Checking
- Point location
- Traversal
- Insertion
- Removal (in 2D)
- Flip, join-split
- Iterators, circulators

Validity

Checking

Validity

Base level

Vertex : `cell()` $\neq$ `NULL`
Cell : `true`

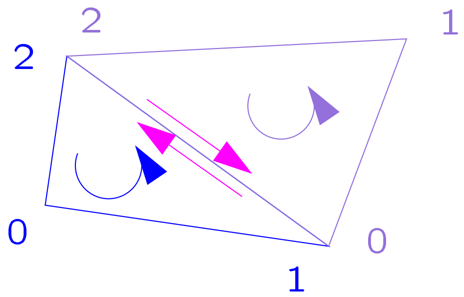
to be modified by the user

Validity

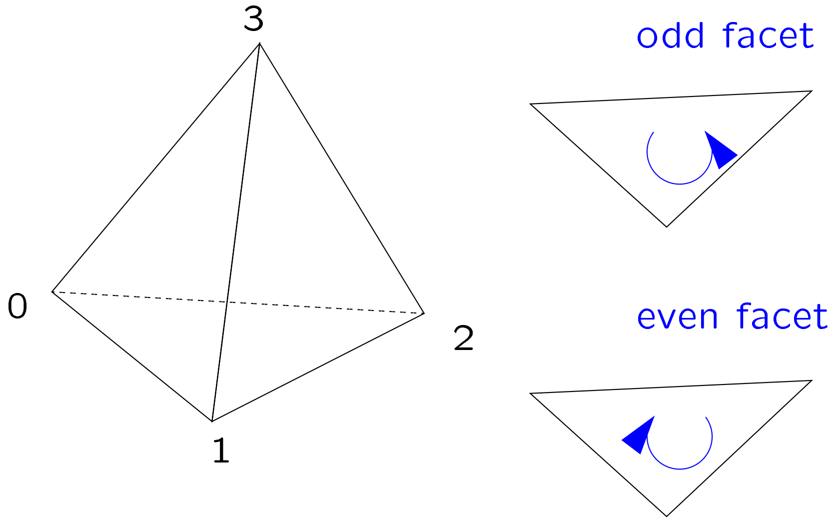
Data structure level

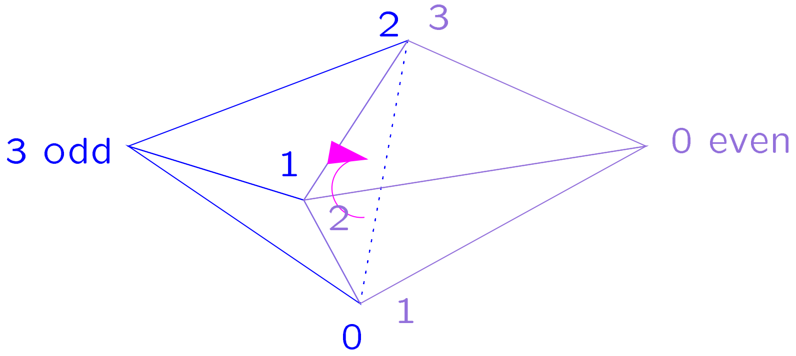
- Incidence and adjacency relations
- Relative combinatorial orientations of the cells
- Euler relation

2D $|\sigma_3| = 6$ permutations of $(0, 1, 2)$
Circular permutations are positive



3D $|\sigma_4| = 24$ permutations of $(0, 1, 2, 3)$
Circular permutations are negative





CODE

Validity

Geometric level

- Geometric orientations

  predicates of the geometric traits

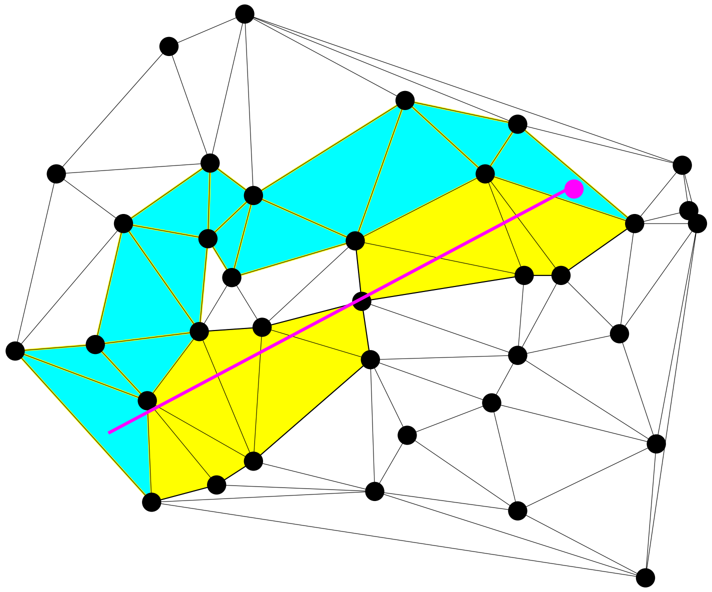
$\Longrightarrow$ *Necessary and sufficient conditions ?*

triangulation of S^3
triangulation of $\mathbb{R}^3$

The geometric triangulation should not
invalidate the combinatorial integrity

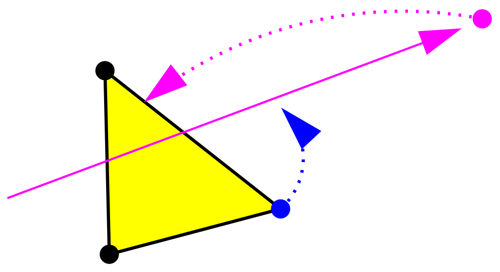
Point location

Basic point location



Basic point location

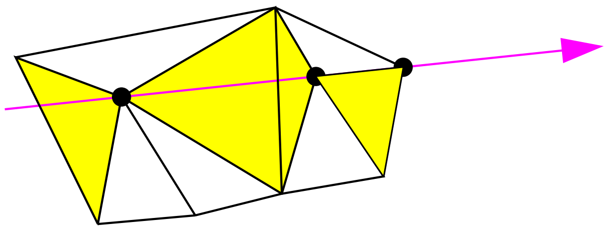
- traversing the triangulation along a line :



2 orientation tests per triangle

3 per tetrahedron in 3D

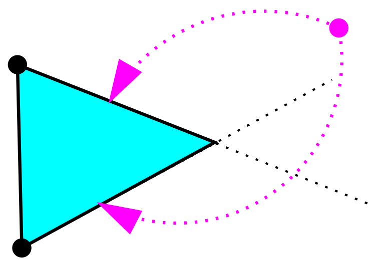
degenerate cases :



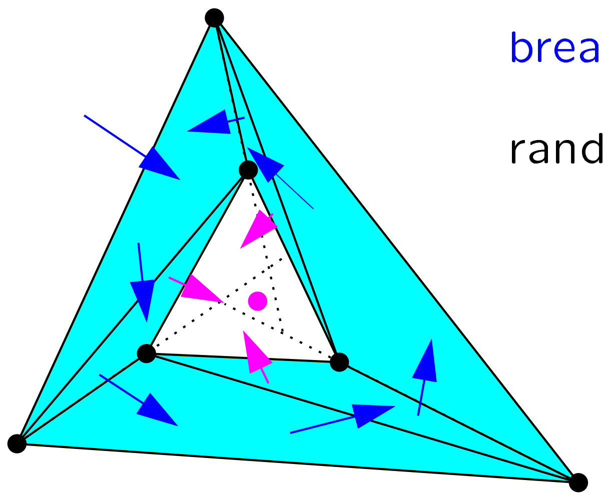
2D package

Basic point location

- by orientation :



< 1.5 tests per triangle
< 2 per tetrahedron in 3D



breaking cycles :

random choice of the neighbor

3D package

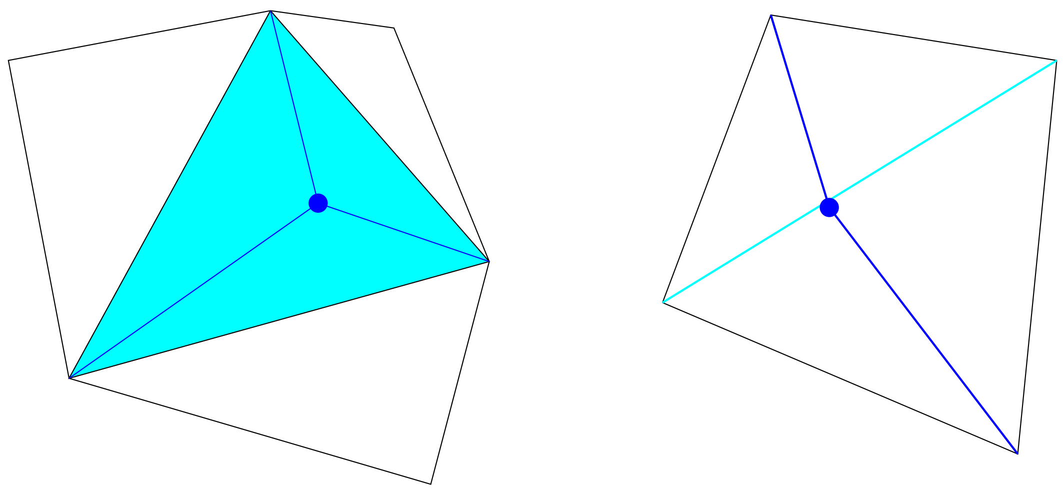
→ *Number of tetrahedra traversed by each method ?*

Insertion

Insertion

Decision : geometry $\longrightarrow$

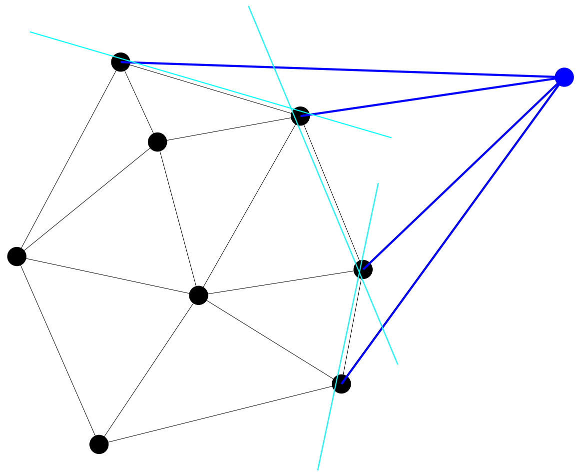
Face and `Locate_type` given by `locate()`



Combinatorial operations

Done by the Triangulation data structure

Insertion



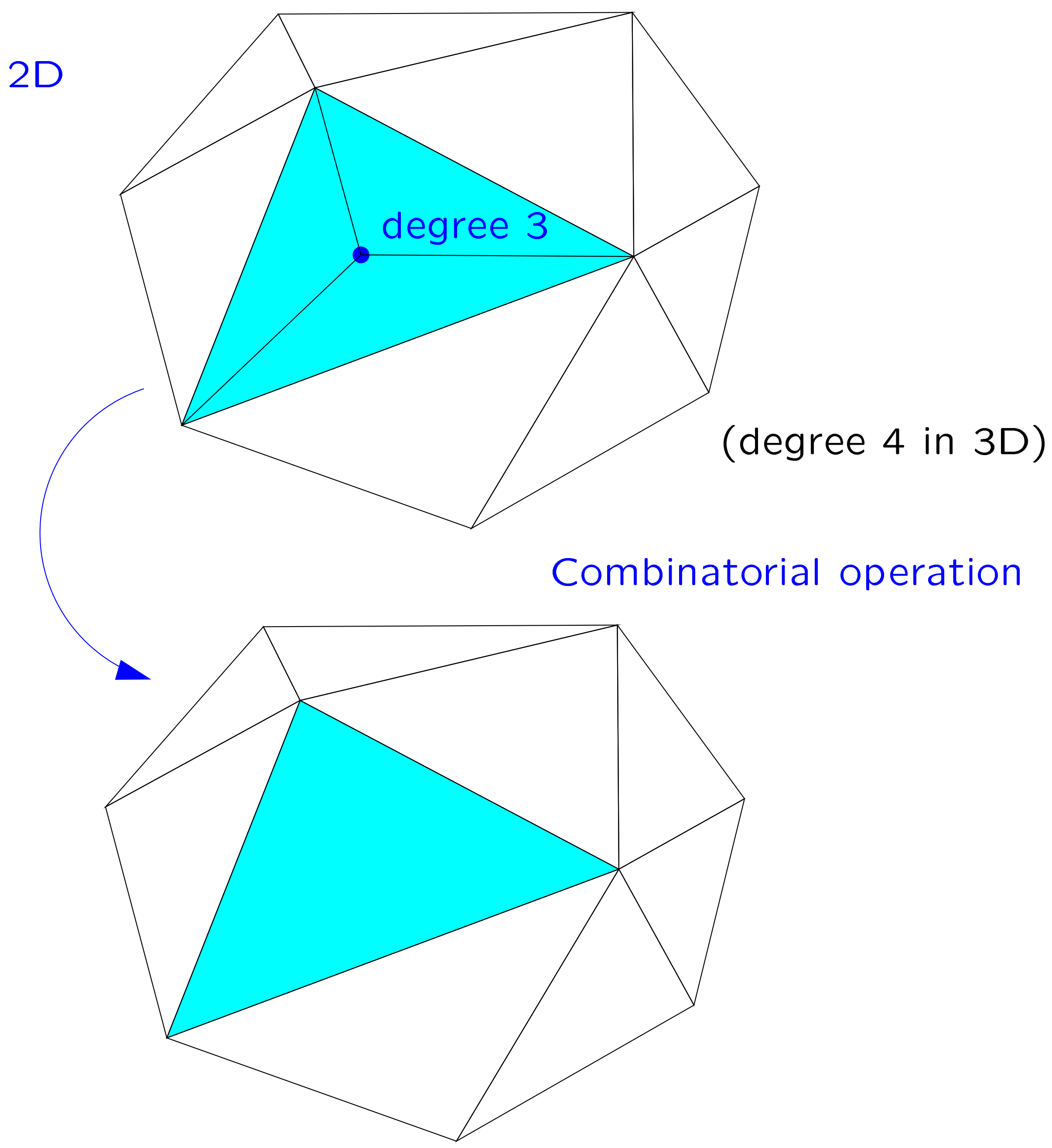
The Triangulation data structure
must know all modified faces

→ managed by the geometric triangulation ?

Removal

Removal

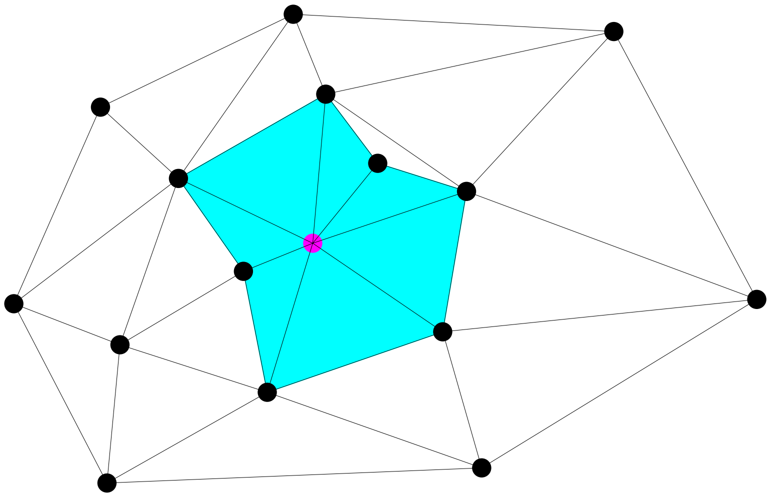
Data structure



Removal

Geometric triangulation

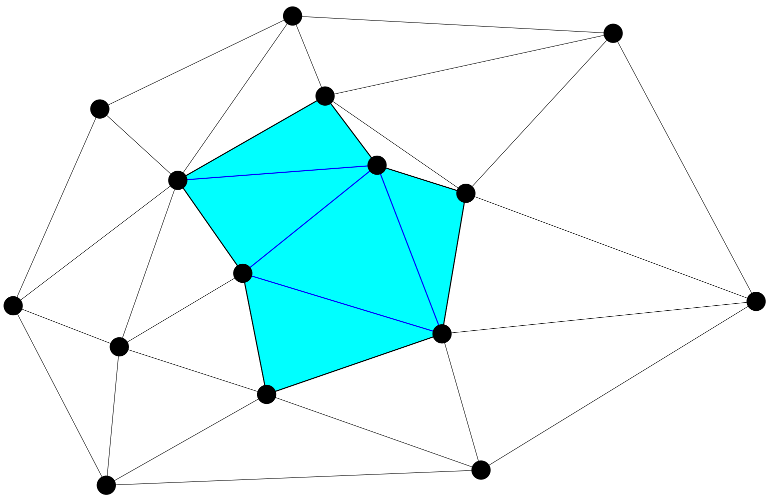
2D



Removal

Geometric triangulation

2D



Retriangulate the hole

Geometric decision

→ managed by the geometric triangulation ?

⟶ can all algorithms be rewritten
so that the geometric triangulation
does not modify the data structure ?

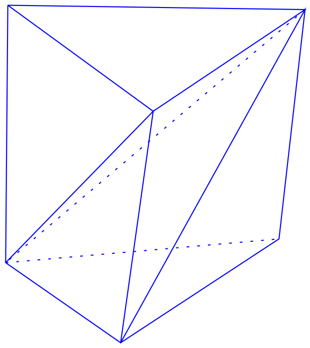
flips ?
interface between Triangulation and Tds ?

Removal

Geometric triangulation

3D

The hole is not always triangulable !

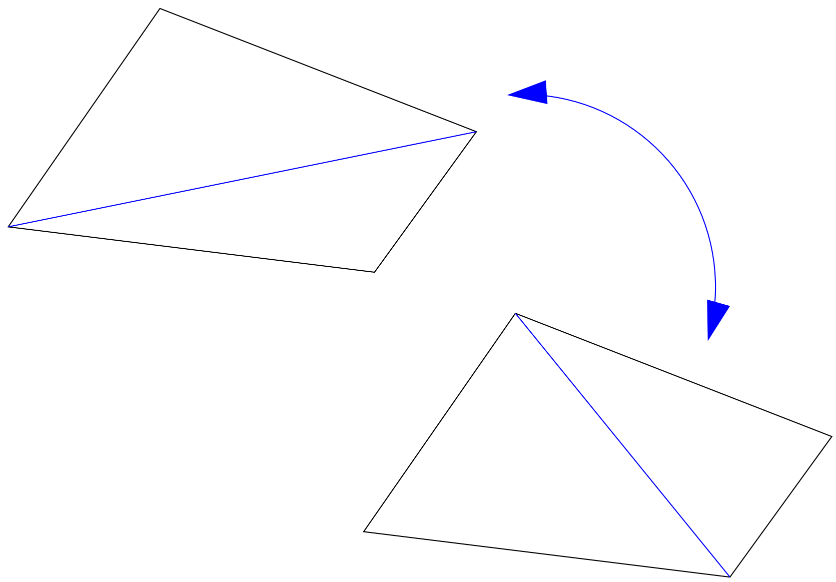


$\longrightarrow$ *Updating the triangulation ?*

Other combinatorial operations

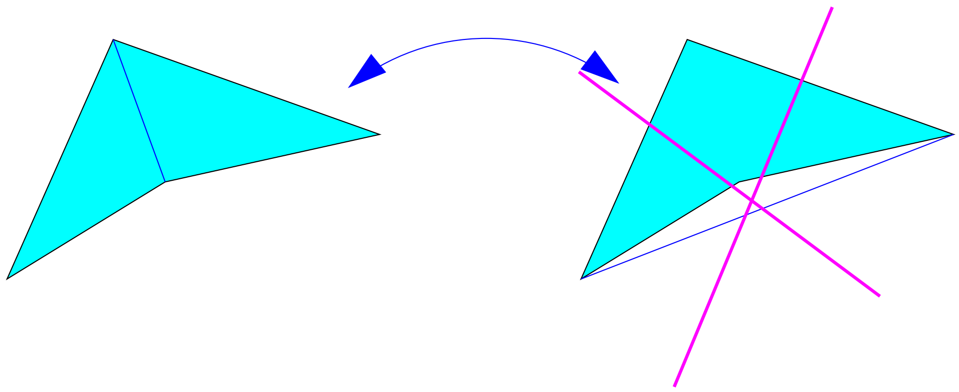
Flip

2D



Operation: combinatorial

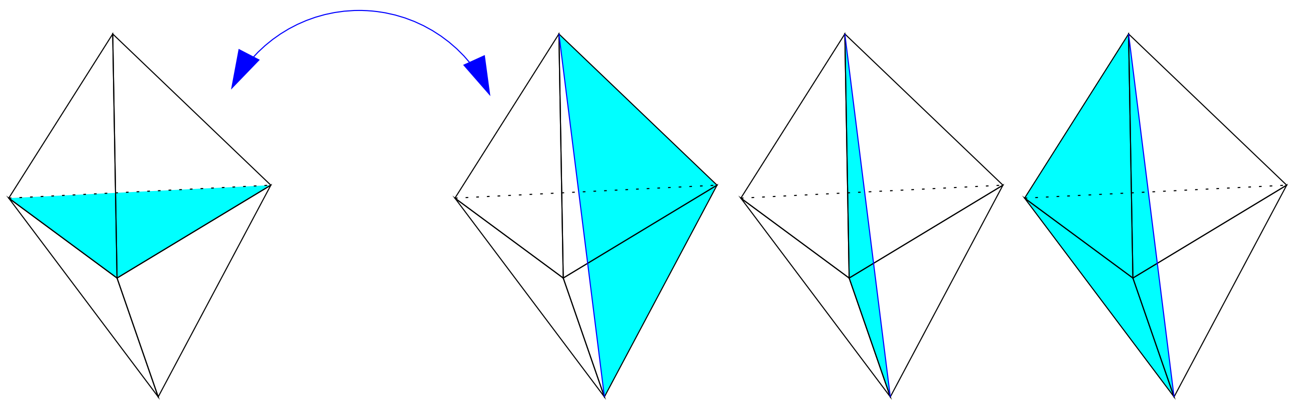
Decision: geometric



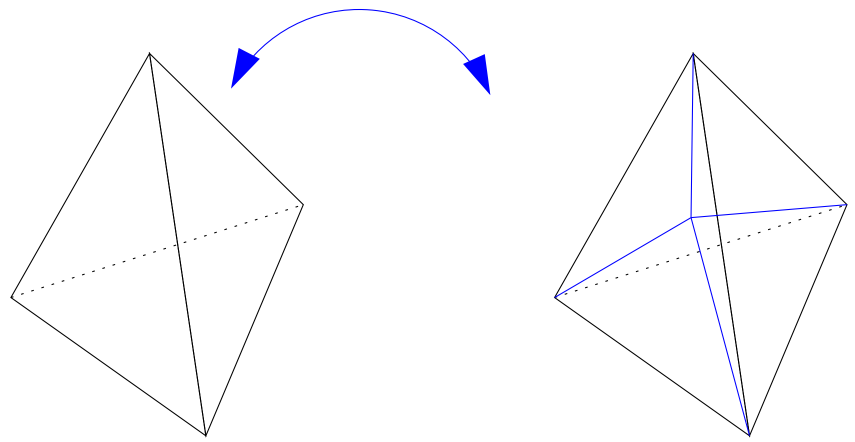
Flip

3D

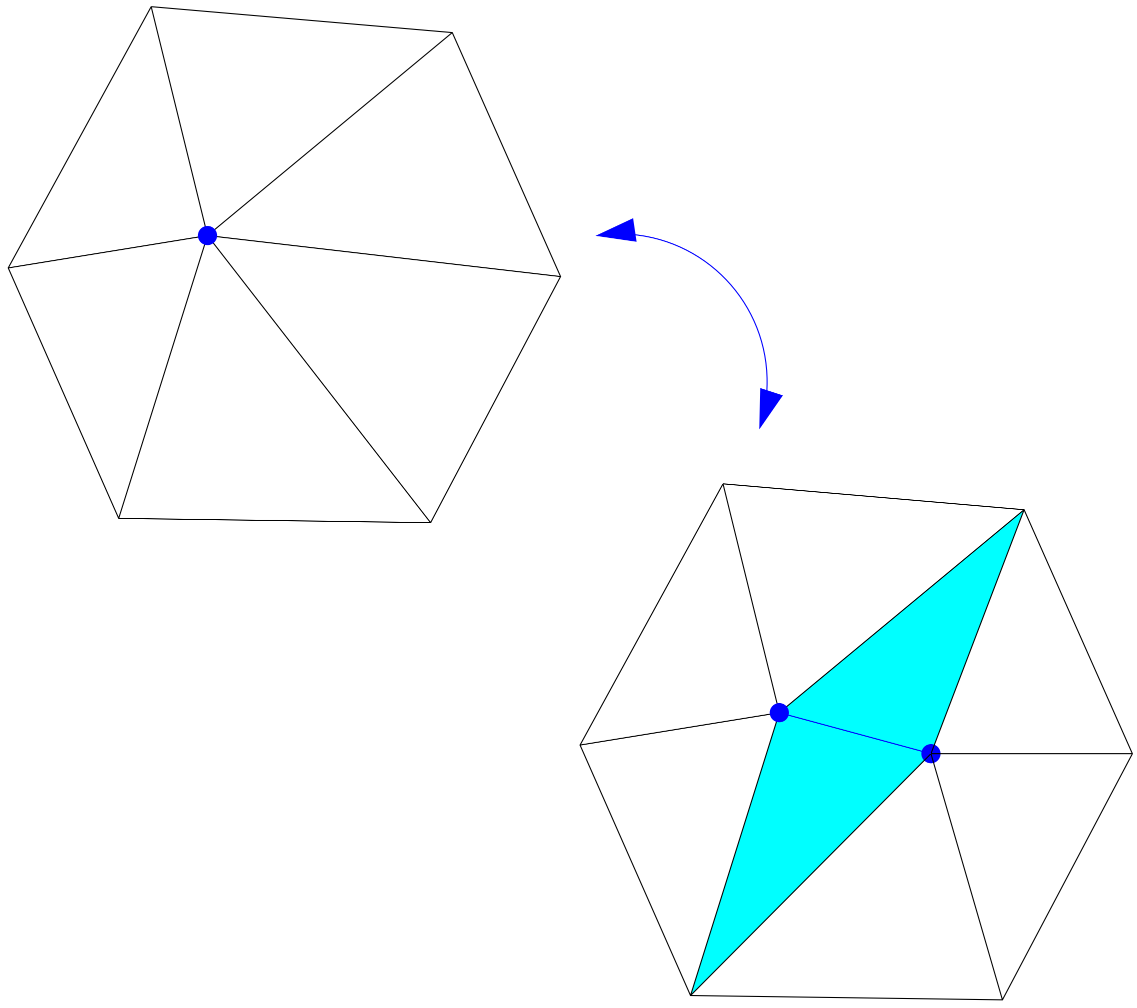
2↔3



1↔4



Join - Split



Operation: combinatorial

Decision: geometric

Iterators

Circulators

2D case

Iterators for

all finite vertices, edges, faces

Circulators for

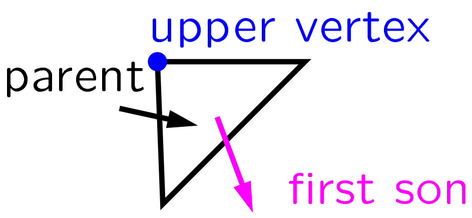
vertices, edges, faces incident to a vertex

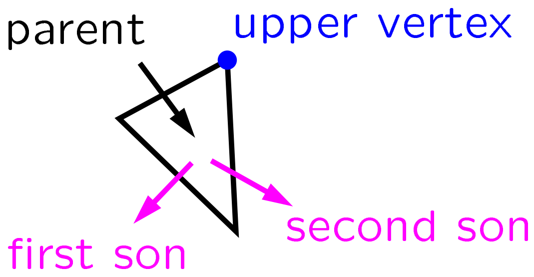
faces traversed by a line

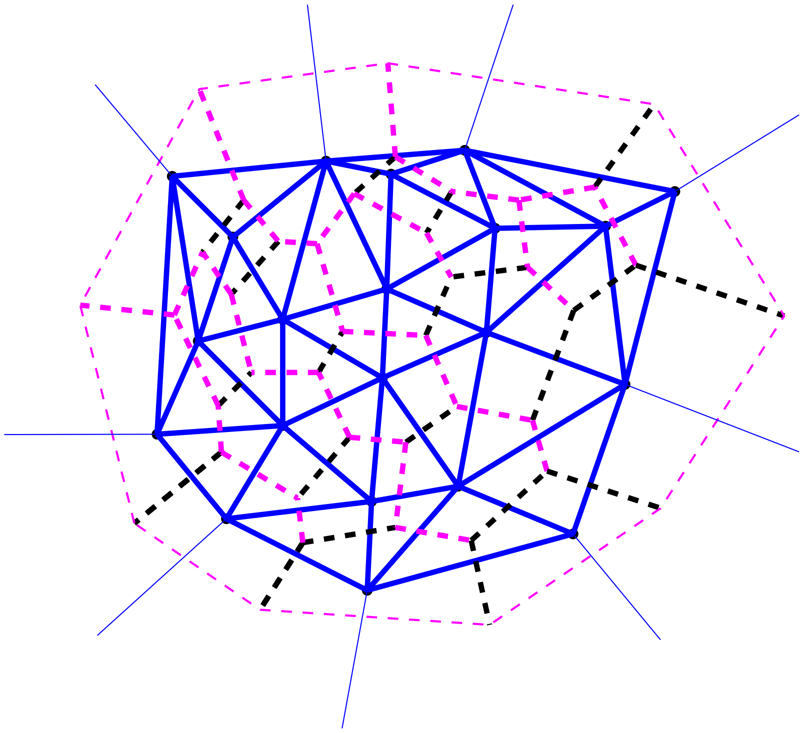
2D case

Iterator on all faces

[deBerg et al 96,Gold et al 77]







Also used by `CGAL_Triangulation_data_structure_2`
$\longrightarrow$ needs the geometric traits

3D case

Iterators for

Vertices, edges, facets, cells

Circulators for

Facets and cells incident to an edge

Set for

Edges, facets, cells incident to a vertex

→ iterator on a set

3D case

Iterator on all cells

Geometric iterator works in theory. . .

Triangulation data structure
should be independent
from the geometry

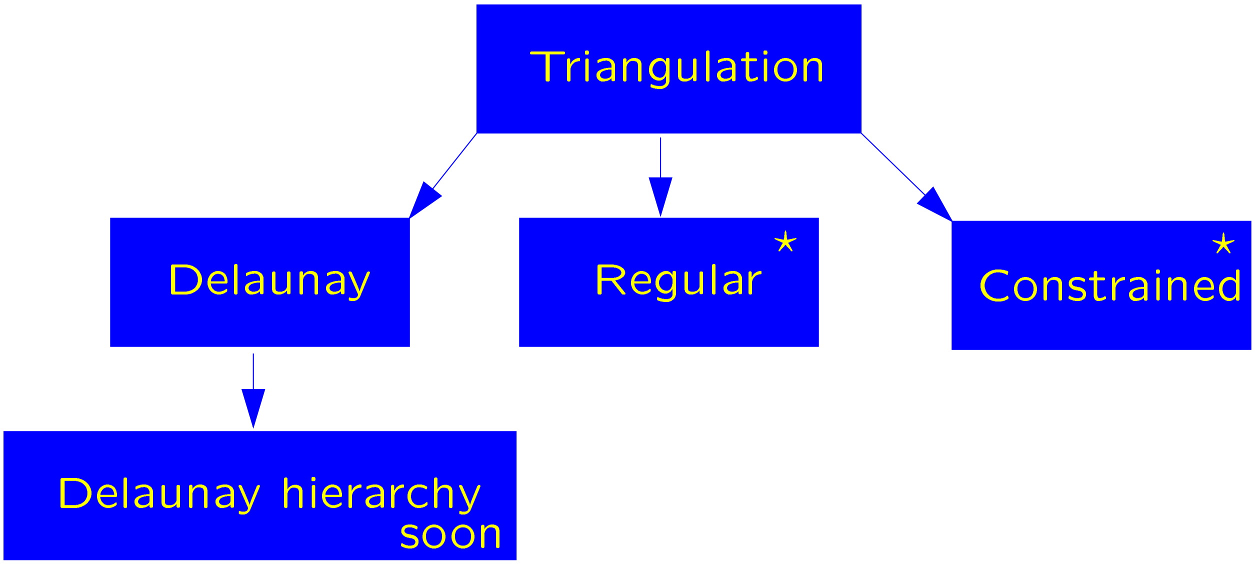
→ list of cells

no external data structure

```
Cell of Triangulation_data_structure_3
private:
Cell* _previous_cell;
Cell* _next_cell;
```

Derived classes

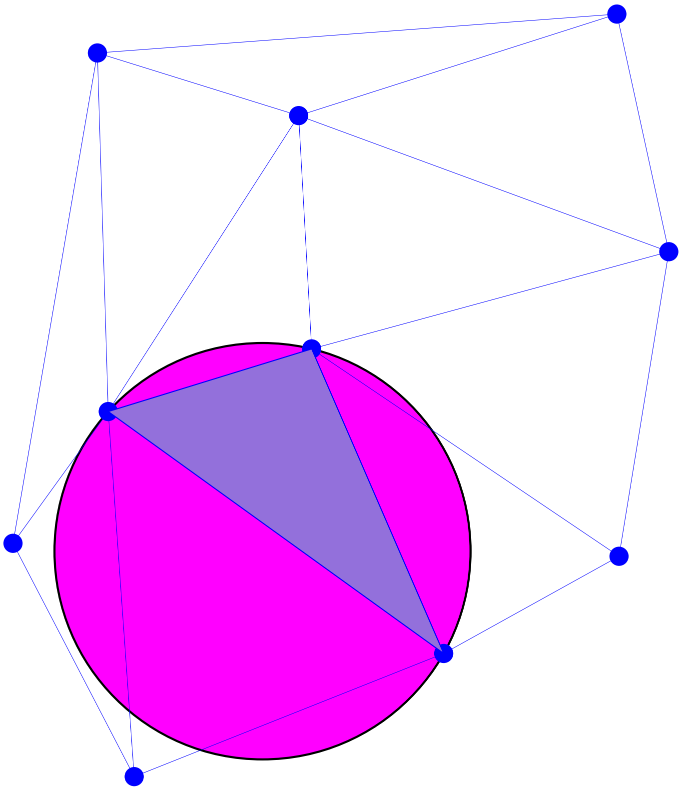
Hierarchy of Triangulations



⋆ only in 2D

Delaunay triangulation

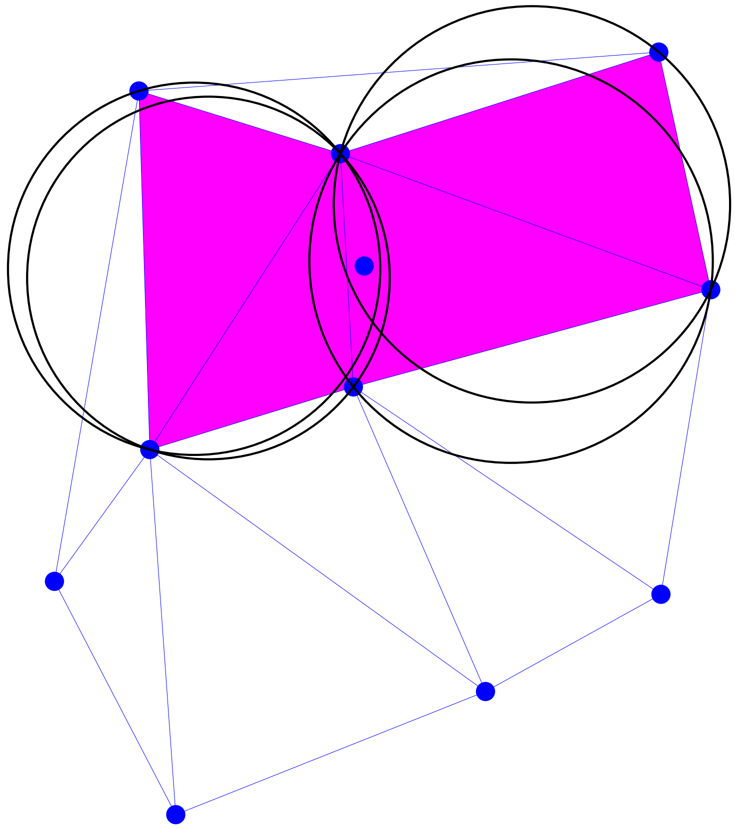
S set of sites in $\mathbb{R}^d$



Delaunay triangulation

Insertion

Location step

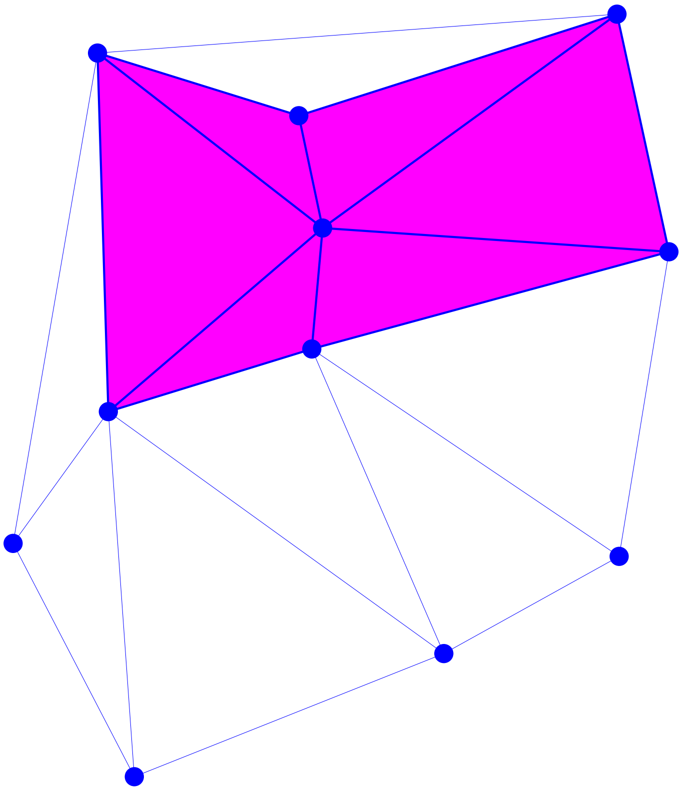


Many algorithms

Delaunay triangulation

Insertion

Update step



Delaunay triangulation

`CGAL_Delaunay_triangulation_3 < Gt, Tds >`

- Inherits from
  `CGAL_Triangulation_3`
- `insert` overloaded
- `is_valid()` checks the empty sphere property using the geometric traits predicates

Delaunay hierarchy

[Devillers]

Randomized incremental construction

[Boissonnat-Teillaud 86,93]
[Guibas-Knuth-Sharir 90]
[Mulmuley 91]
...

New structure

Simple

Small storage

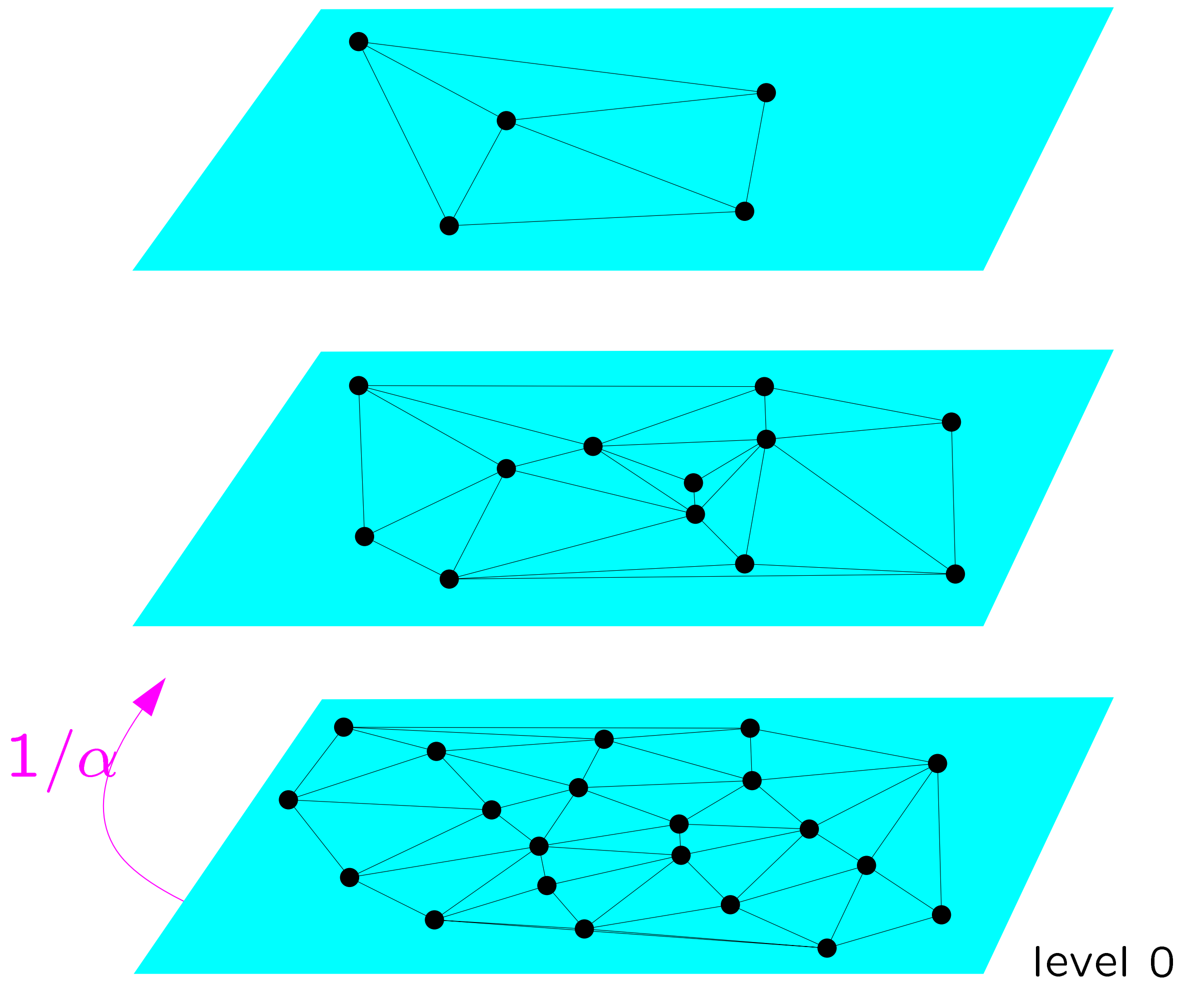
Dynamic

Efficient in practice

worst-case provable

Delaunay hierarchy

Location structure



Delaunay hierarchy

q query

v_i nearest vertex at level i

At level i

walk from v_{i+1} to v_i

- Turn around v_{i+1}
- Walk along $v_{i+1}q$
- Look at vertices "close to q "

Delaunay hierarchy

Worst case

$O(n \log_\alpha n)$	time
$\left(1 + \frac{1}{\alpha - 1}\right) n$	memory

Random points

$\frac{32 + 6.2\sqrt{\alpha}}{\log_2 \alpha} \log_2 n \qquad \alpha \simeq 40$

Delaunay hierarchy

`CGAL_Delaunay_hierarchy_2`

- Inherits from `CGAL_Delaunay_triangulation_2`
- `locate`, `insert`, `remove` overloaded

`CGAL_Delaunay_hierarchy_vertex_base_2`

- Inherits from `Triangulation_vertex_base_2`

CODE

Future

New functionalities

3D package

- (Benchmarks, optimization,. . .)
- Removal of a vertex in a Delaunay triangulation
  [Devillers]
- . . .
- Regular triangulations
- Constrained triangulations
- Mesh generation
- . . .

Future design ?

Reorganization of all the classes

All layers share the same vertices and cells

Advantages

coherence between types in Vertex and Traits class

no `void*` pointers

no casts

CODE

Demo

- by François Rebufat
  (also the test-suite for 3D triangulations)

  3D-visualization : MESA (OpenGL)

- Future : CGAL on the web
  `http://cgal.inria.fr/`