3D Triangulations in CGAL

Monique Teillaud

www.cgal.org

January 2012

Overview

- Definitions
- Functionalities
- Geometry vs. Combinatorics
- Representation
- Software Design
- Algorithms
- Some recent and ongoing work

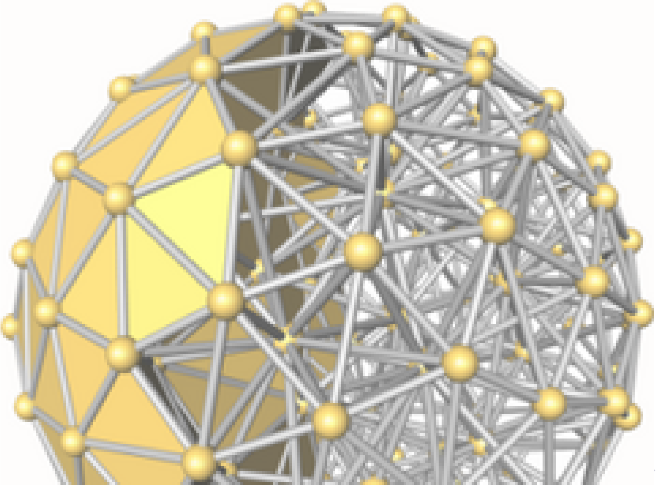
Part I

Definitions

Definition

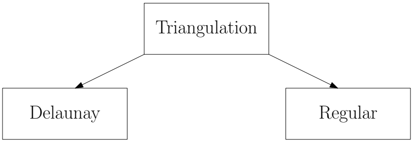
2D (3D) simplicial complex = set $\mathcal{K}$ of **0,1,2,3**-faces such that

- $\sigma \in \mathcal{K}, \tau \leq \sigma \Rightarrow \tau \in \mathcal{K}$
- $\sigma, \sigma' \in \mathcal{K} \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$



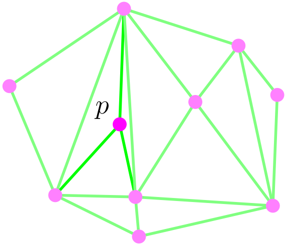
Various triangulations

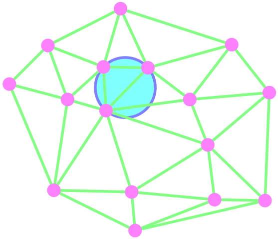
2D, **3D** Basic triangulations
2D, **3D** Delaunay triangulations
2D, **3D** Regular triangulations



Basic and Delaunay triangulations

(figures in 2D)





Basic triangulations : incremental construction

Delaunay triangulations: empty sphere property

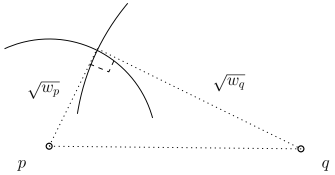
Regular triangulations

weighted point $p^{(w)} = (p, w_p), p \in \mathbb{R}^3, w_p \in \mathbb{R}$
$p^{(w)} = (p, w_p) \simeq$ sphere of center p and radius $\sqrt{w_p}$.
power product between $p^{(w)}$ and $z^{(w)}$

$$\Pi(p^{(w)}, z^{(w)}) = \|p - z\|^2 - w_p - w_z$$

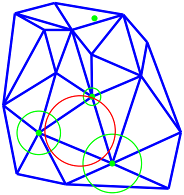
$p^{(w)}$ and $z^{(w)}$ **orthogonal** iff $\Pi(p^{(w)}, z^{(w)}) = 0$

(2D)



Regular triangulations

Power sphere of 4 weighted points in $\mathbb{R}^3$ =
unique common orthogonal weighted point.
$z^{(w)}$ is **regular** iff $\forall p^{(w)}, \Pi(p^{(w)}, z^{(w)}) \geq 0$



(2D)

Regular triangulations: generalization of Delaunay triangulations to weighted points. Dual of the **power diagram**.

The power sphere of all simplices is regular.

Part II

Functionalities of CGAL triangulations

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

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

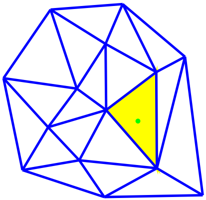
Around a vertex

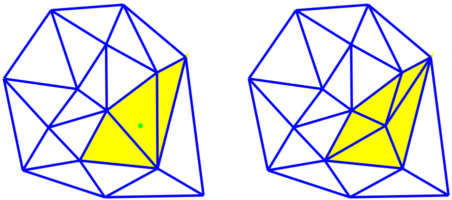
incident cells and facets, adjacent vertices

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

Point location, insertion, removal

basic triangulation:

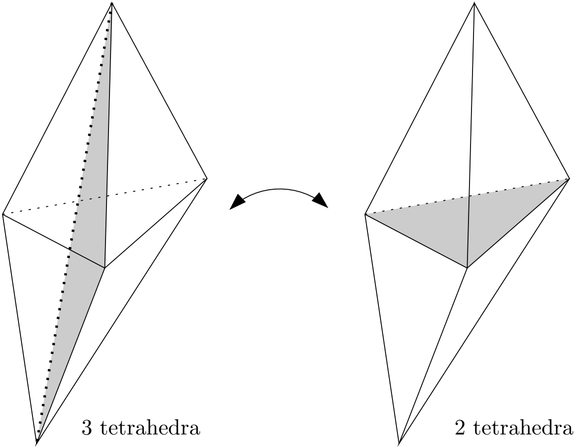




Delaunay triangulation :

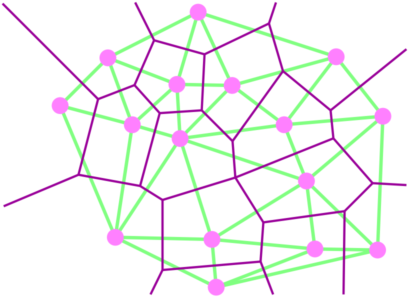
3D Flip

if convex position



Additional functionalities for Delaunay triangulations

Nearest neighbor queries
Voronoi diagram



Part III

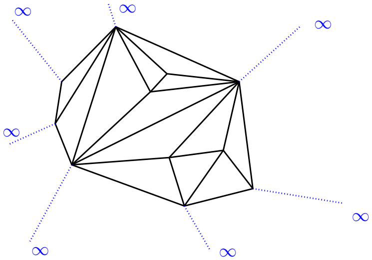
Geometry vs. Combinatorics

Infinite vertex

Triangulation of a set of points = partition of the **convex hull** into simplices.

Addition of an **infinite vertex**
$\longrightarrow$ "triangulation" of the outside of the convex hull.

- Any cell is a "tetrahedron".
- Any facet is incident to two cells.



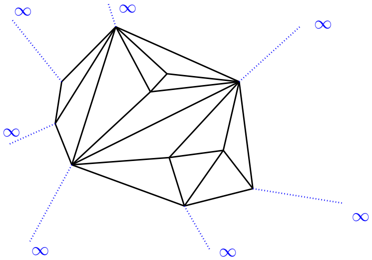
(2D)

Infinite vertex

Triangulation of a set of points = partition of the **convex hull** into simplices.

Addition of an **infinite vertex**
$\longrightarrow$ "triangulation" of the outside of the convex hull.

- Any cell is a "tetrahedron".
- Any facet is incident to two cells.



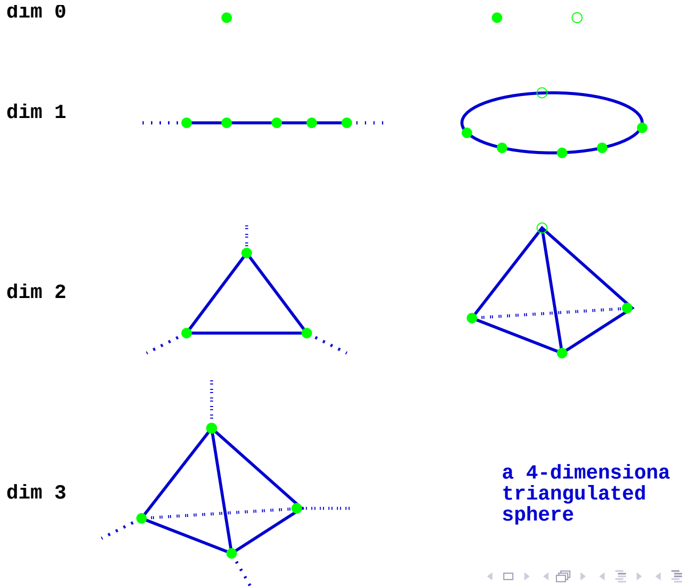
(2D)

Triangulation of $\mathbb{R}^d$

$\simeq$

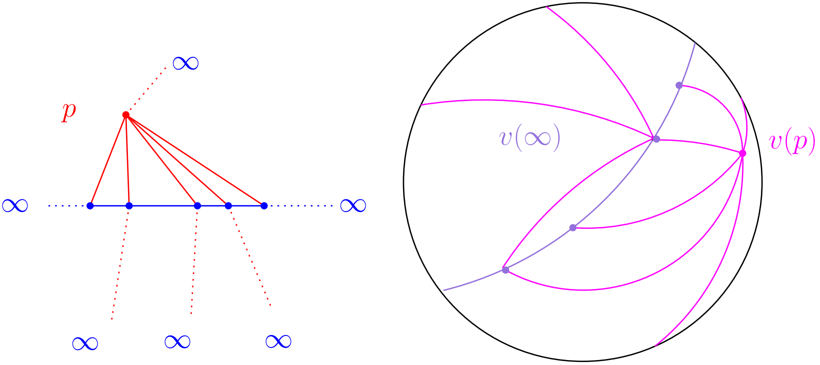
Triangulation of the topological **sphere** $\mathcal{S}^d$.

Dimensions



Dimensions

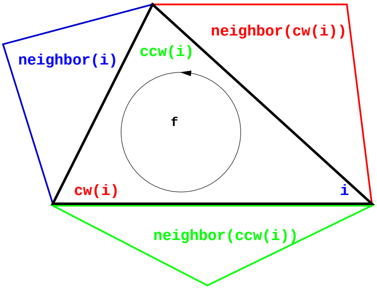
Adding a point outside the current affine hull:
From $d = 1$ to $d = 2$



Part IV

Representation

2D - Representation based on faces



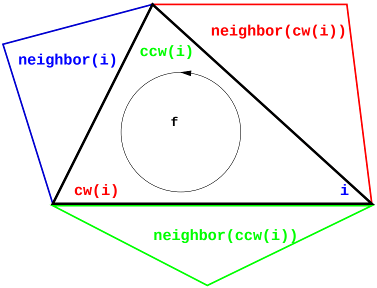
Vertex

Face_handle *v_face*

Face

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

2D - Representation based on faces



Vertex

Face_handle *v_face*

Face

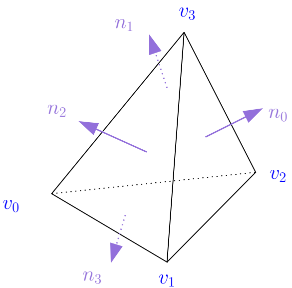
Vertex_handle *vertex*[3]
Face_handle *neighbor*[3]

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

From one face to another

$n = f \rightarrow neighbor(i)$
$j = n \rightarrow index(f)$

3D - Representation based on cells



Vertex

Cell_handle *v_cell*

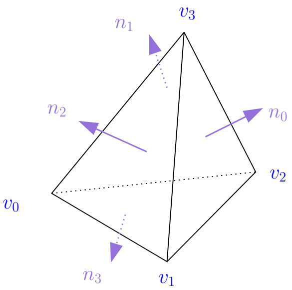
Cell

Vertex_handle *vertex*[4]
Cell_handle *neighbor*[4]

Faces are implicit: `std::pair<` c, i `>`
where c = one of the two incident cells.

Edges are implicit: `std::pair<` u, v `>`
where u, v = vertices.

3D - Representation based on cells



Vertex

Cell_handle *v_cell*

Cell

Vertex_handle *vertex*[4]
Cell_handle *neighbor*[4]

From one cell to another

```
n = c → neighbor(i)
j = n → index(c)
```

Part V

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 own traits class

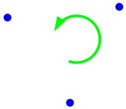
Traits class

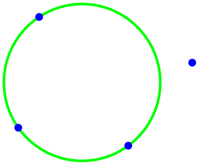
Generic algorithms

```
Delaunay_Triangulation_2<Traits, TDS>
```

Traits parameter provides:

- Point
- orientation test, in_circle test



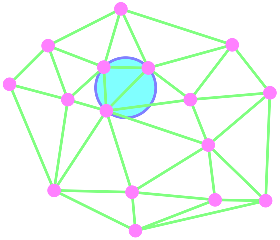


Traits class

2D Kernel used as traits class

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

- 2D points: coordinates $(\mathbf{x}, \mathbf{y})$
- orientation, in_circle



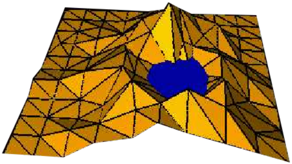
Traits class

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;
```

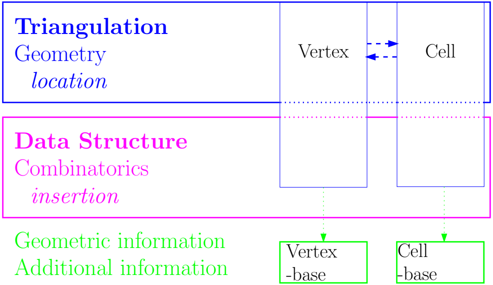
- 3D points: coordinates $(\mathbf{x}, \mathbf{y}, \mathbf{z})$
- orientation, in_circle:

  on **x** and **y** coordinates only



Layers

Triangulation_3< Traits, **TDS** >



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

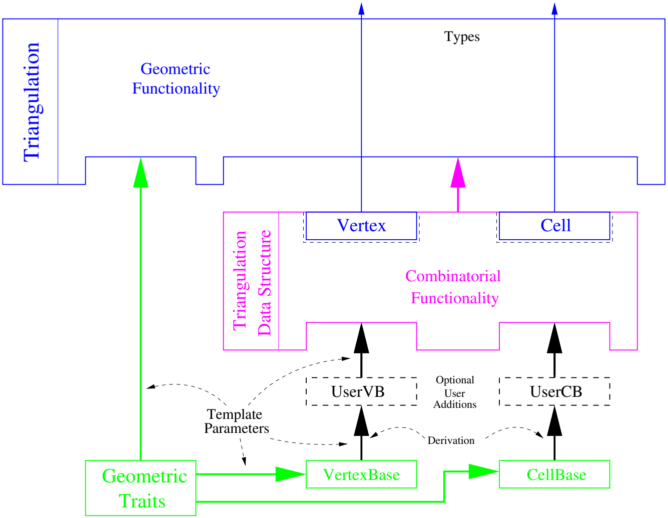
Layers

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.

Changing the Vertex_base and the Cell_base



Changing the Vertex_base and the Cell_base

First option: Triangulation_vertex_base_with_info_3

When the additional information does not depend on the TDS

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Triangulation_vertex_base_with_info_3.h>
#include <CGAL/IO/Color.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef CGAL::Triangulation_vertex_base_with_info_3
                                        <CGAL::Color,K> Vb;

typedef CGAL::Triangulation_data_structure_3<Vb>    Tds;
typedef CGAL::Delaunay_triangulation_3<K, Tds>      Delaunay;

typedef Delaunay::Point    Point;
```

Changing the Vertex_base and the Cell_base

First option: Triangulation_vertex_base_with_info_3

When the additional information does not depend on the TDS

```
int main()
{
  Delaunay T;
  T.insert(Point(0,0,0));  T.insert(Point(1,0,0));
  T.insert(Point(0,1,0));  T.insert(Point(0,0,1));
  T.insert(Point(2,2,2));  T.insert(Point(-1,0,1));

     // Set the color of finite vertices of degree 6 to red.
  Delaunay::Finite_vertices_iterator vit;
  for (vit = T.finite_vertices_begin();
                vit != T.finite_vertices_end(); ++vit)
     if (T.degree(vit) == 6)
          vit->info() = CGAL::RED;

  return 0;
}
```

Changing the Vertex_base and the Cell_base

Third option: write new models of the concepts

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

- Vertex and cell base classes:

  initially given a **dummy TDS** template parameter:

dummy TDS provides the types that can be used
by the vertex and cell base classes (such as handles).

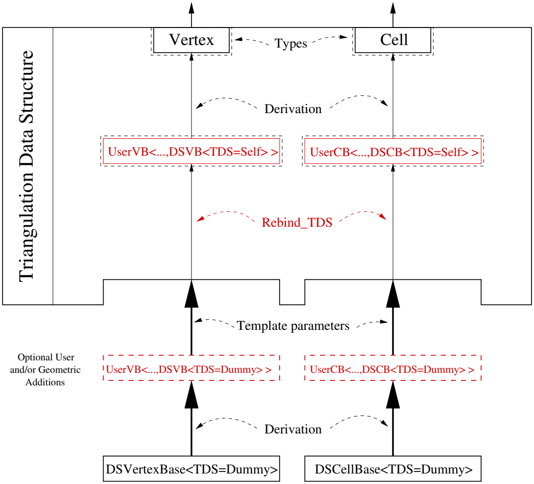
- inside the TDS itself,

  vertex and cell base classes are
  rebound to the real TDS type

→ the same vertex and cell base classes are now
parameterized with the real TDS instead of the dummy one.

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism



Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

```
template< class GT, class Vb= Triangulation_vertex_base<GT> >
class My_vertex
  : public Vb
{
public:
 typedef typename Vb::Point           Point;
 typedef typename Vb::Cell_handle     Cell_handle;

 template < class TDS2 >
 struct Rebind_TDS {
  typedef typename Vb::template Rebind_TDS<TDS2>::Other Vb2;
  typedef My_vertex<GT, Vb2>                            Other;
 };

 My_vertex() {}
 My_vertex(const Point&p)                : Vb(p) {}
 My_vertex(const Point&p, Cell_handle c) : Vb(p, c) {}
...
}
```

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

Example

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Triangulation_vertex_base_3.h>
```

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

Example

```
template < class GT, class Vb=CGAL::Triangulation_vertex_base_
class My_vertex_base   : public Vb
{ public:
 typedef typename Vb::Vertex_handle  Vertex_handle;
 typedef typename Vb::Cell_handle    Cell_handle;
 typedef typename Vb::Point          Point;

 template < class TDS2 > struct Rebind_TDS {
  typedef typename Vb::template Rebind_TDS<TDS2>::Other Vb2;
  typedef My_vertex_base<GT, Vb2>                       Other;
 };

 My_vertex_base() {}
 My_vertex_base(const Point& p) : Vb(p) {}
 My_vertex_base(const Point& p, Cell_handle c) : Vb(p, c) {}

 Vertex_handle   vh;
 Cell_handle     ch;
};
```

Changing the Vertex_base and the Cell_base

Second option: the “rebind” mechanism

Example

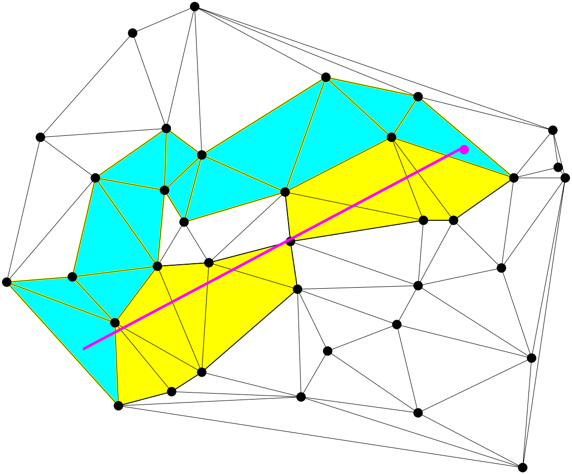
```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::
      Triangulation_data_structure_3< My_vertex_base<K> > Tds;
typedef CGAL::
      Delaunay_triangulation_3< K, Tds >              Delaunay;
typedef Delaunay::Vertex_handle    Vertex_handle;
typedef Delaunay::Point            Point;

int main()
{ Delaunay T;
  Vertex_handle v0 = T.insert(Point(0,0,0));
  ... v1; v2; v3; v4; v5;
   // Now we can link the vertices as we like.
  v0->vh = v1;  v1->vh = v2;
  v2->vh = v3;  v3->vh = v4;
  v4->vh = v5;  v5->vh = v0;
  return 0;
}
```

Part VI

Algorithms

Point location

`Locate_type`



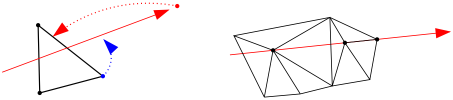
Point location

2D (/3D)

- Along a straight line

2 (/3) orientation tests
per triangle (/tetrahedron)

degenerate cases



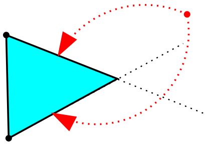
CGAL 2D triangulations

Point location

2D (/3D)

- By visibility

$<$ 1.5 (/2) tests
per triangle (/tetrahedron)



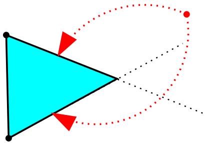
CGAL 3D triangulations

Point location

2D (/3D)

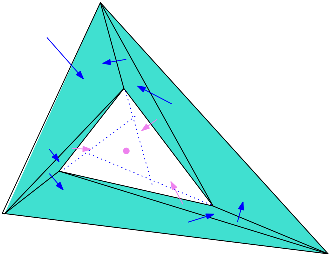
- By visibility

$<$ 1.5 (/2) tests
per triangle (/tetrahedron)



CGAL 3D triangulations

Breaking cycles: random choice of the neighbor



Point location - Example

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h
#include <CGAL/Triangulation_3.h>

#include <iostream>
#include <fstream>
#include <cassert>
#include <list>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef CGAL::Triangulation_3<K>      Triangulation;

typedef Triangulation::Cell_handle    Cell_handle;
typedef Triangulation::Vertex_handle  Vertex_handle;
typedef Triangulation::Locate_type    Locate_type;
typedef Triangulation::Point          Point;
```

Point location - Example

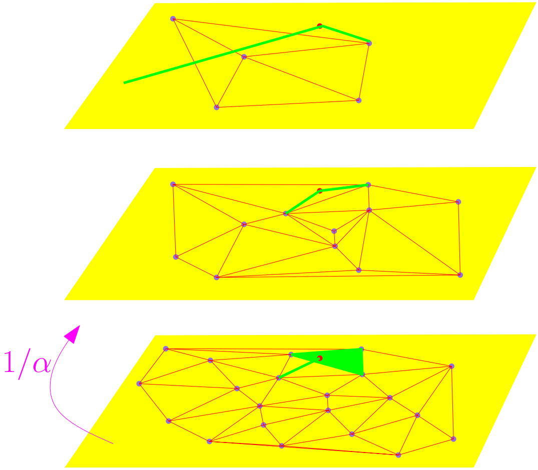
```
int main()
{
  std::list<Point> L;
  L.push_front(Point(0,0,0));
  L.push_front(Point(1,0,0));
  L.push_front(Point(0,1,0));
  Triangulation T(L.begin(), L.end());
  int n = T.number_of_vertices();

  std::vector<Point> V(3);
  V[0] = Point(0,0,1);
  V[1] = Point(1,1,1);
  V[2] = Point(2,2,2);
  n = n + T.insert(V.begin(), V.end());

  assert( n == 6 );
  assert( T.is_valid() );
```

Point location - Example

```
    Locate_type lt;   int li, lj;
    Point p(0,0,0);
    Cell_handle c = T.locate(p, lt, li, lj);
    assert( lt == Triangulation::VERTEX );
    assert( c->vertex(li)->point() == p );

    Vertex_handle v = c->vertex( (li+1)&3 );
    Cell_handle nc = c->neighbor(li);
    int nli;  assert( nc->has_vertex( v, nli ) );

    std::ofstream oFileT("output",std::ios::out);
    oFileT << T;
    Triangulation T1;
    std::ifstream iFileT("output",std::ios::in);
    iFileT >> T1;
    assert( T1.is_valid() );
    assert( T1.number_of_vertices() == T.number_of_vertices() );
    assert( T1.number_of_cells() == T.number_of_cells() );
    return 0;
}
```

The Delaunay Hierarchy

Point location data structure



Optimal randomized worst-case complexity

The Delaunay Hierarchy

Point location data structure

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h
#include <CGAL/Delaunay_triangulation_3.h>
#include <CGAL/Random.h>

#include <vector>
#include <cassert>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_3<K, CGAL::Fast_location>
typedef Delaunay::Point Point;
```

The Delaunay Hierarchy

Point location data structure

```
int main()
{ Delaunay T;
  std::vector<Point> P;
  for (int z=0 ; z<20 ; z++)
    for (int y=0 ; y<20 ; y++)
      for (int x=0 ; x<20 ; x++)
          P.push_back(Point(x,y,z));

  Delaunay T(P.begin(), P.end());
  assert( T.number_of_vertices() == 8000 );

  for (int i=0; i<10000; ++i)
    T.nearest_vertex
        ( Point(CGAL::default_random.get_double(0,20),
                CGAL::default_random.get_double(0,20),
                CGAL::default_random.get_double(0,20)) );

  return 0;
}
```

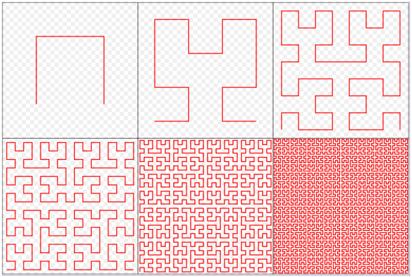
Spatial Sorting
Speeding-up construction

Incremental algorithms

$\longrightarrow$ Efficiency depends on the order of insertion

Sort points along a space-filling curve

Example: Hilbert curve



(2D picture from Wikipedia)

Spatial Sorting
Speeding-up construction

Incremental algorithms

$\longrightarrow$ Efficiency depends on the order of insertion

Sort points along a space-filling curve

close geometrically $\Longleftrightarrow$ close in the insertion order
with high probability

$\Longrightarrow$ point location

- previous cell in cache memory $\rightarrow$ faster start
- previous point close $\rightarrow$ shorter walk

$\Longrightarrow$ memory locality improved $\rightarrow$ speed-up in data structure

Spatial Sorting
Speeding-up construction

Incremental algorithms

$\longrightarrow$ Efficiency depends on the order of insertion

Sort points along a space-filling curve

CGAL spatial sort =

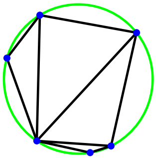
Hilbert sort + `std::random_shuffle`

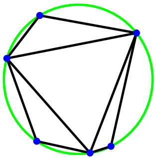
- points still close enough for speed-up
- some randomness for randomized algorithms

```
template < class InputIterator >
int
t.insert (InputIterator first, InputIterator last)
```

Cospherical points

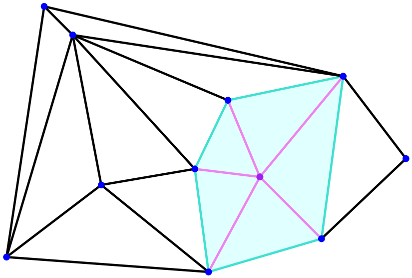
Any triangulation is a Delaunay triangulation





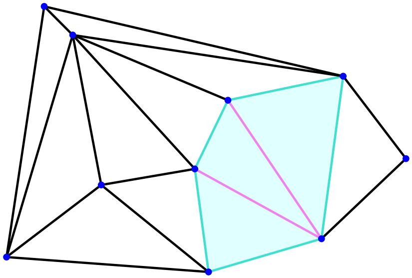
Vertex removal

1- remove the tetrahedra incident to $v \longrightarrow$ hole



Vertex removal

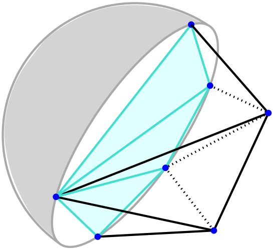
1- remove the tetrahedra incident to v $\longrightarrow$ hole
2- retriangulate the hole



Symbolic perturbation
Robustness to degenerate cases

Vertex removal **Cocircular points**

Several possible Delaunay triangulations of a facet of the hole

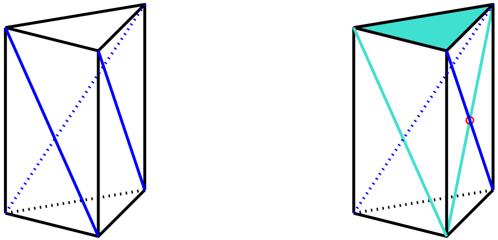


Triangulation of the hole must be
compatible with the rest of the triangulation

Remark on the general question:

H given polyhedron with triangulated facets.
Find a Delaunay triangulation of *H* keeping its facets ?

Not always possible:



Symbolic perturbation

Robustness to degenerate cases

Allowing flat tetrahedra?

k cocircular points on a facet

2D triangulation of the facet induced by tetrahedra in the hole

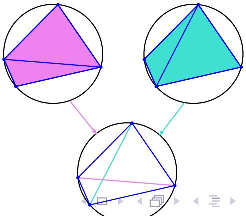
$\vdots$

sequence of $O(k^2)$ edge flips

$\vdots$

2D triangulation of the facet induced by tetrahedra outside the hole

edge flip $\longleftrightarrow$ flat tetrahedron



Symbolic perturbation

Robustness to degenerate cases

Allowing flat tetrahedra?

k cocircular points on a facet

2D triangulation of the facet induced by tetrahedra in the hole

$\vdots$

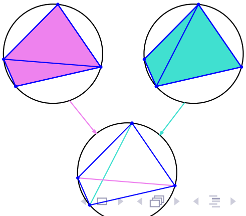
sequence of $O(k^2)$ edge flips

$\vdots$

2D triangulation of the facet induced by tetrahedra outside the hole

edge flip $\longleftrightarrow$ flat tetrahedron

Unacceptable



Symbolic perturbation
Robustness to degenerate cases

Solution

$\longrightarrow$ Perturbing the *in_sphere* predicate

orientation predicate not perturbed $\longrightarrow$ no flat tetrahedra

p_0, p_1, p_2, p_3, p_4 non coplanar

in_sphere $(p_0, p_1, p_2, p_3, p_4)$

$$\begin{cases} > 0 & \text{if } p_4 \text{ is outside} \\ = 0 & \text{if } p_4 \text{ is on the boundary of} \\ < 0 & \text{if } p_4 \text{ is inside} \end{cases}$$

the ball circumscribing p_0, p_1, p_2, p_3.

Symbolic perturbation

Robustness to degenerate cases

p_0, p_1, p_2, p_3, p_4 non coplanar

$$in\_sphere\,(p_0, p_1, p_2, p_3, p_4) = \frac{sign\ Det(p_0, p_1, p_2, p_3, p_4)}{orient(p_0, p_1, p_2, p_3)}$$

$$orient(p_0, p_1, p_2, p_3) = sign \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \end{vmatrix}$$

Symbolic perturbation
Robustness to degenerate cases

$$in\_sphere\,(p_0, p_1, p_2, p_3, p_4) = \frac{sign\ Det(p_0, p_1, p_2, p_3, p_4)}{orient(p_0, p_1, p_2, p_3)}$$

$$Det(p_0, p_1, p_2, p_3, p_4) =$$

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_0 & x_1 & x_2 & x_3 & x_4 \\ y_0 & y_1 & y_2 & y_3 & y_4 \\ z_0 & z_1 & z_2 & z_3 & z_4 \\ x_0^2+y_0^2+z_0^2 & x_1^2+y_1^2+z_1^2 & x_2^2+y_2^2+z_2^2 & x_3^2+y_3^2+z_3^2 & x_4^2+y_4^2+z_4^2 \end{vmatrix}$$

Symbolic perturbation

Robustness to degenerate cases

$$in\_sphere\,(p_0,p_1,p_2,p_3,p_4) = \frac{sign\; Det(p_0,p_1,p_2,p_3,p_4)}{orient(p_0,p_1,p_2,p_3)}$$

$$Det(p_0,p_1,p_2,p_3,p_4) =$$

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_0 & x_1 & x_2 & x_3 & x_4 \\ y_0 & y_1 & y_2 & y_3 & y_4 \\ z_0 & z_1 & z_2 & z_3 & z_4 \\ t_0 & t_1 & t_2 & t_3 & t_4 \end{vmatrix}$$

$\iff$ orientation in $\mathbb{R}^4$

$$\begin{array}{llll} \pi : & \mathbb{R}^3 & \rightarrow & \Pi \subset \mathbb{R}^4 \\ & p_i = (x_i, y_i, z_i) & \mapsto & \pi(p_i) = (x_i, y_i, z_i, t_i = x_i^2 + y_i^2 + z_i^2) \end{array}$$

Symbolic perturbation
Robustness to degenerate cases

Perturbation

points indexed

$$\begin{array}{rccc} \pi_\varepsilon : & \Pi \subset \mathbb{R}^4 & \rightarrow & \mathbb{R}^4 \\ & \pi(p_i) = (x_i, y_i, z_i, t_i) & \mapsto & (x_i, y_i, z_i, t_i + \varepsilon^{n-i}) \end{array}$$

$$Det_\varepsilon(p_i, p_j, p_k, p_l, p_m) =$$

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_l & x_m \\ y_i & y_j & y_k & y_l & y_m \\ z_i & z_j & z_k & z_l & z_m \\ t_i+\varepsilon^{n-i} & t_j+\varepsilon^{n-j} & t_k+\varepsilon^{n-k} & t_l+\varepsilon^{n-l} & t_m+\varepsilon^{n-m} \end{vmatrix}$$

Symbolic perturbation

Robustness to degenerate cases

Perturbation

points indexed

$$\begin{array}{rccc} \pi_\varepsilon : & \Pi \subset \mathbb{R}^4 & \rightarrow & \mathbb{R}^4 \\ & \pi(p_i) = (x_i, y_i, z_i, t_i) & \mapsto & (x_i, y_i, z_i, t_i + \varepsilon^{n-i}) \end{array}$$

$$Det_\varepsilon(p_i, p_j, p_k, p_l, p_m) =$$

$$\begin{aligned} Det(p_i, p_j, p_k, p_l, p_m) & \\ & + orient(p_i, p_j, p_k, p_l)\ \varepsilon^{n-m} \\ & - orient(p_i, p_j, p_k, p_m)\ \varepsilon^{n-l} \\ & + orient(p_i, p_j, p_l, p_m)\ \varepsilon^{n-k} \\ & - orient(p_i, p_k, p_l, p_m)\ \varepsilon^{n-j} \\ & + orient(p_j, p_k, p_l, p_m)\ \varepsilon^{n-i} \end{aligned}$$

polynomial in ε

Perturbation

5 cospherical points:

point with highest index

$\rightarrow$

outside the sphere of the other 4 [non coplanar] points

4 coplanar points:

point with highest index

$\rightarrow$

outside the disk of the other 3 points in their plane

Perturbation

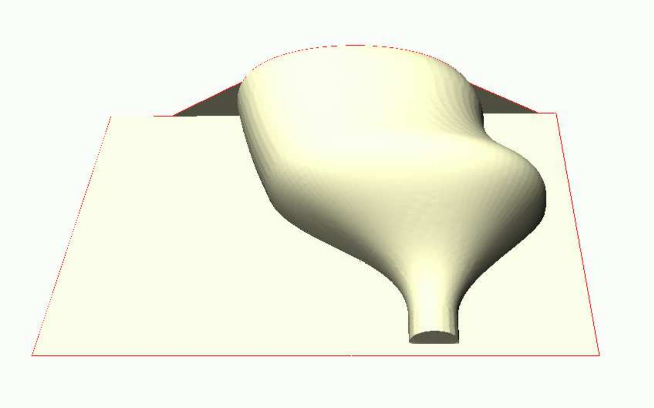
- Free choice for indexing the points

  lexicographic order

- Algorithm working even in degenerate situations
- No flat tetrahedra
- Perturbed predicate easy to code

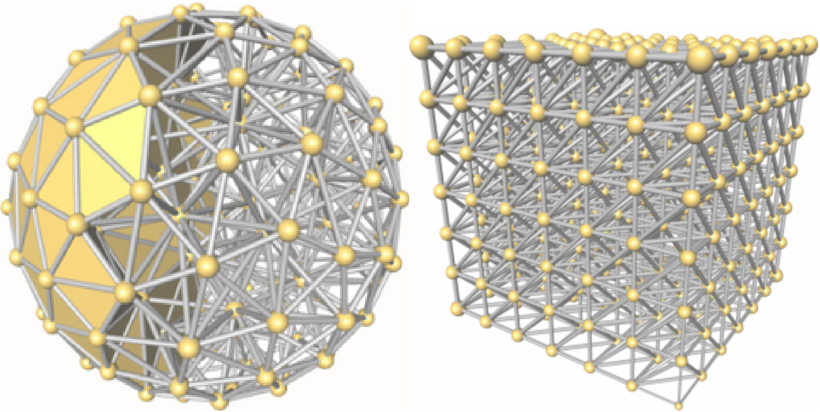
CGAL : only publicly available software
proposing a fully dynamic 3D Delaunay/regular triangulation.

Robustness



Dassault Systèmes

Robustness

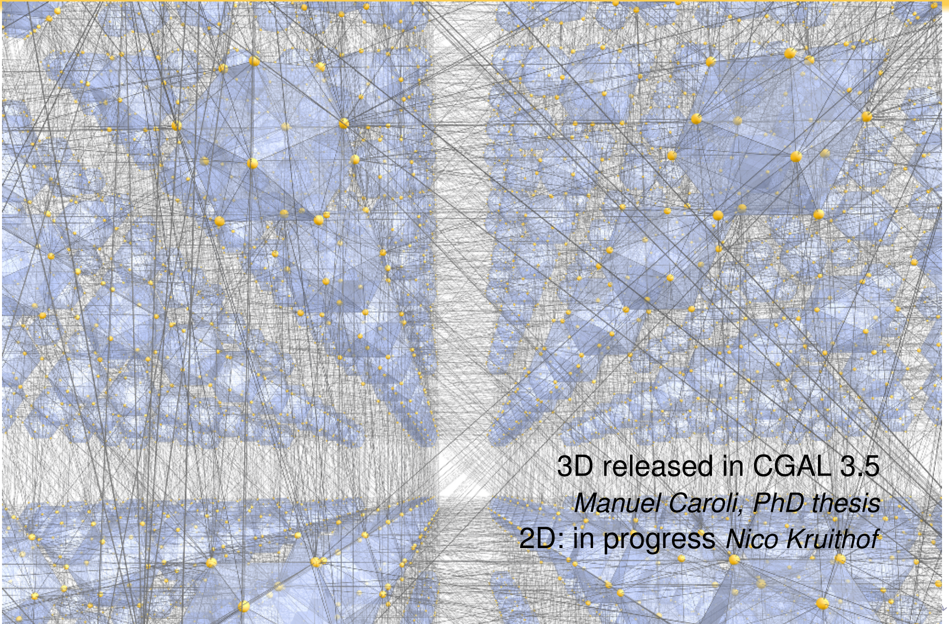


Pictures by Pierre Alliez

Part VII

Some recent and ongoing work

Periodic triangulations

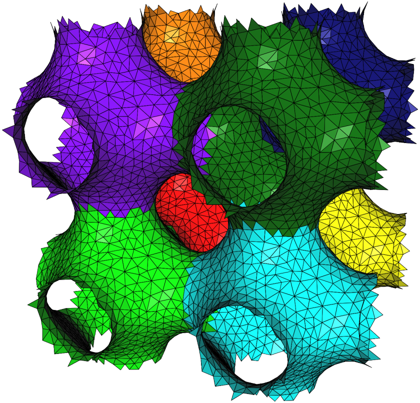


3D released in CGAL 3.5

Manuel Caroli, PhD thesis

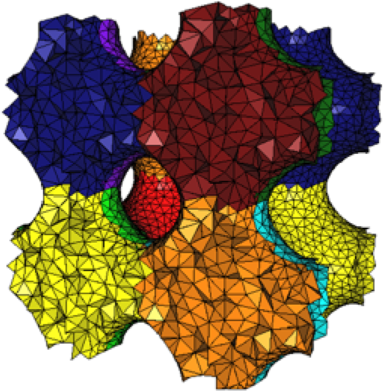
2D: in progress *Nico Kruithof*

Meshing of periodic surfaces



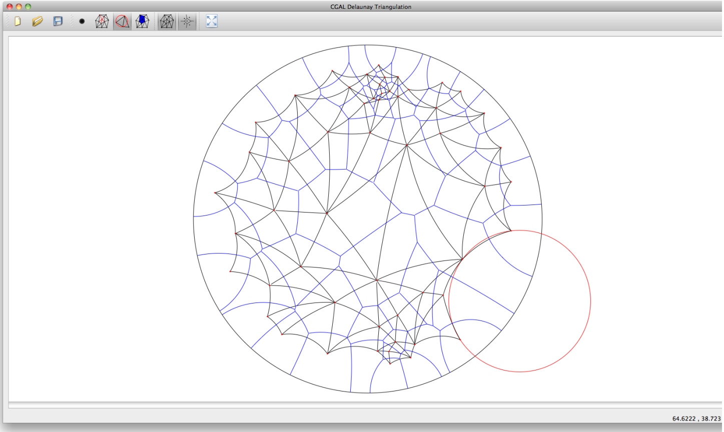
Vissarion Fisikopoulos, internship Uni. Athens

Meshing of periodic volumes



Mikhail Bogdanov, internship Moscow Institute of Physics and Technology

Hyperbolic triangulations



Mikhail Bogdanov, PhD student

Triangulation on the sphere

Olivier Rouiller, internship École Centrale Lille
Claudia Werner, internship Uni. Applied Sciences Stuttgart

To know more

CGAL

www.cgal.org

`http://www.inria.fr/sophia/members/Monique.Teillaud/`