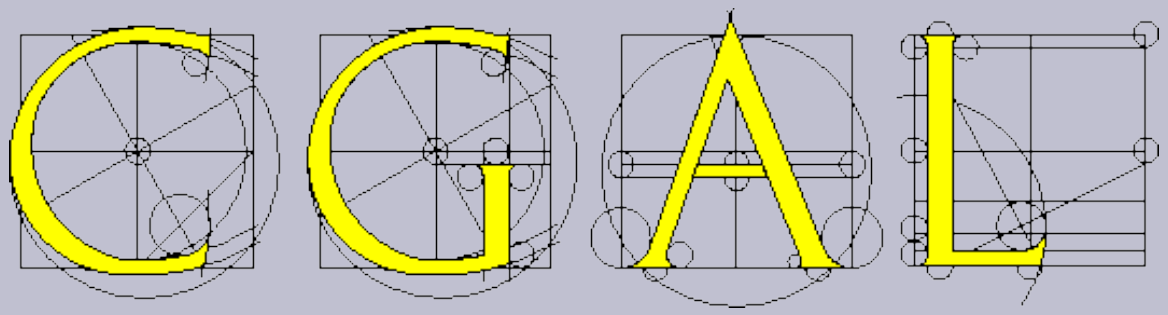
Triangulations and Mesh Generation in



Andreas Fabri

Pierre Alliez

Mariette Yvinec

Geometry Factory

Outline

- Definition of triangulations
- Representation
- Software design
- Algorithmic Issues

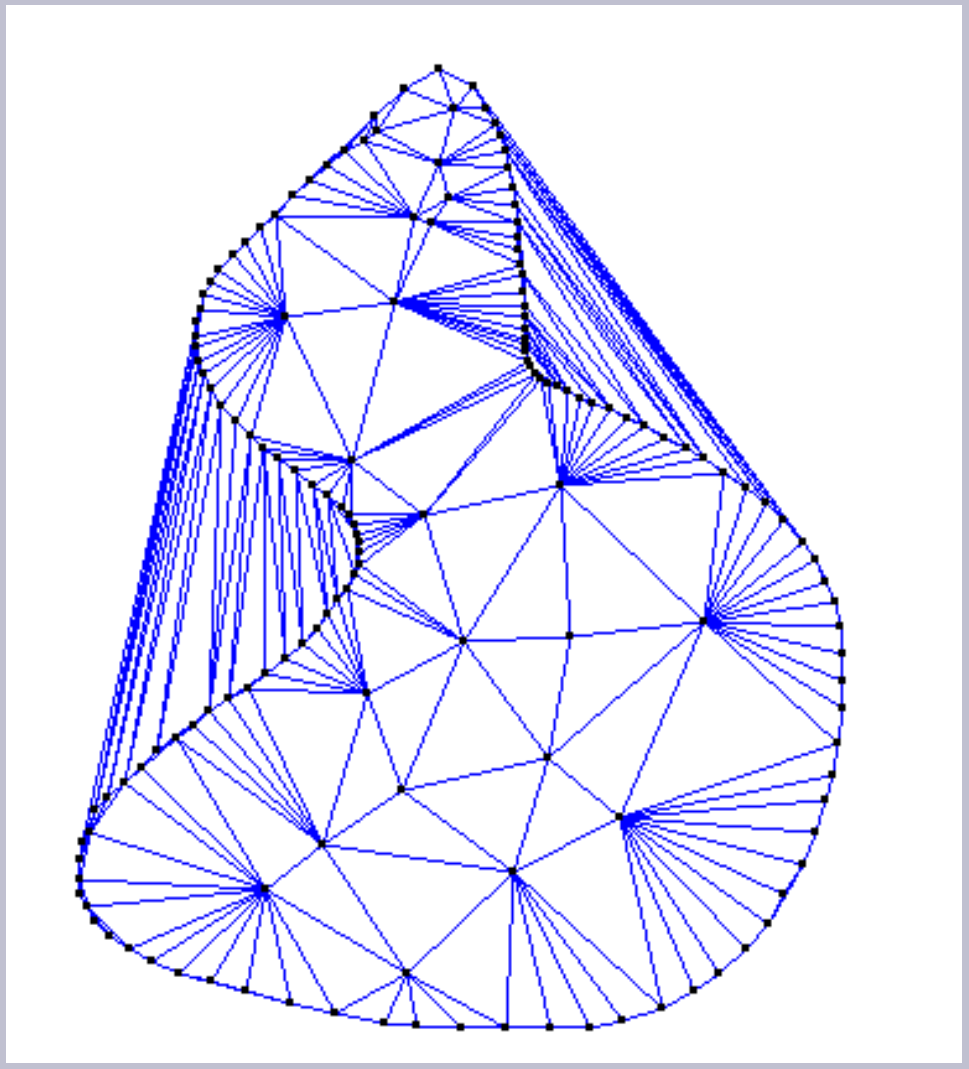
- Meshing

Definitions

Triangulations as a Set of Faces

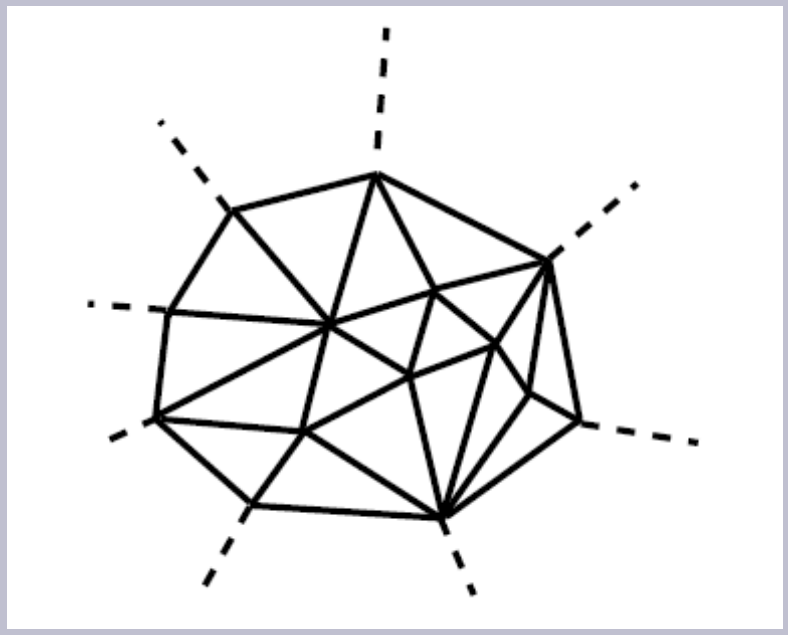
All triangulations in CGAL tile the convex hull of their vertices.

Triangulated polygonal regions can be obtained through constrained triangulations.



Triangulations as a Set of Faces

- An imaginary vertex "at infinity" is added.
- Then
  - any face is a triangle.
  - any edge is incident to two exactly 2 faces.
  - the set of faces is equivalent to a 2D topological sphere

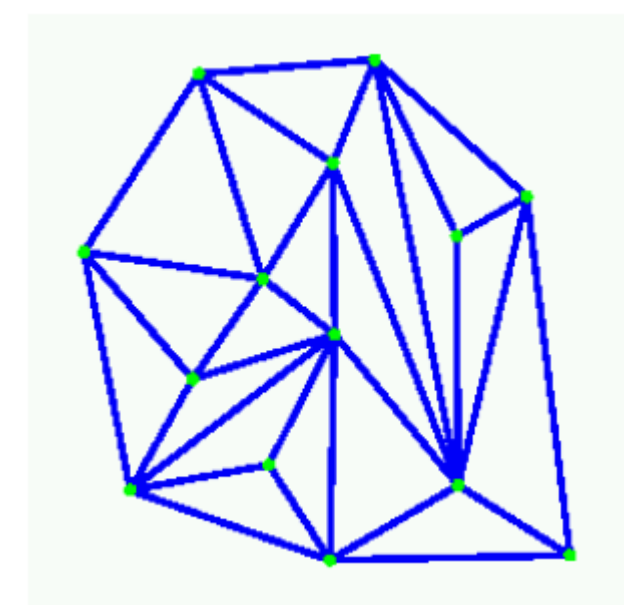


Triangulations in CGAL

- Basic
- Delaunay
- Constrained
- Constrained Delaunay
- Regular

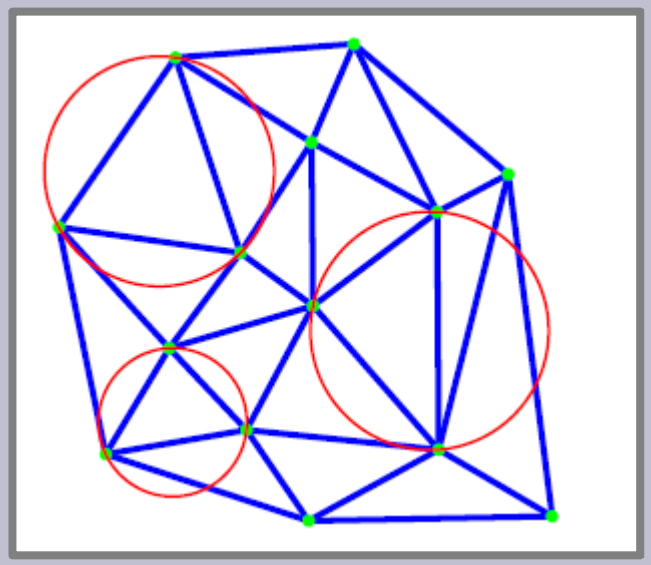
Basic Triangulation

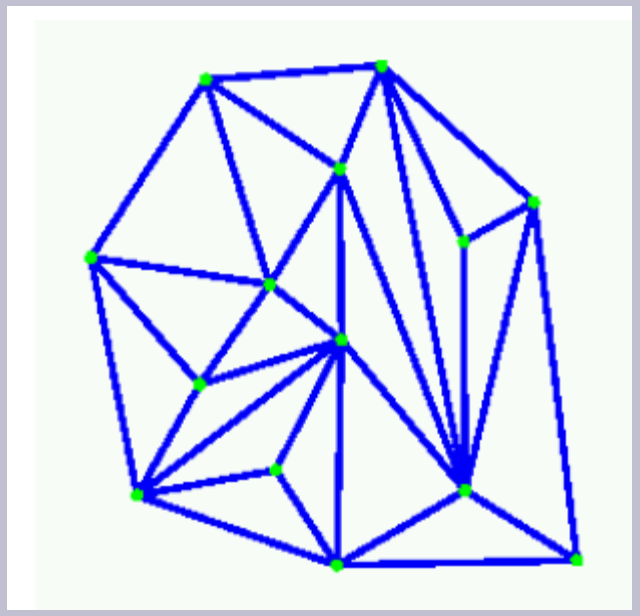
- Lazy incremental construction, no control over the shape of triangles



Delaunay Triangulation

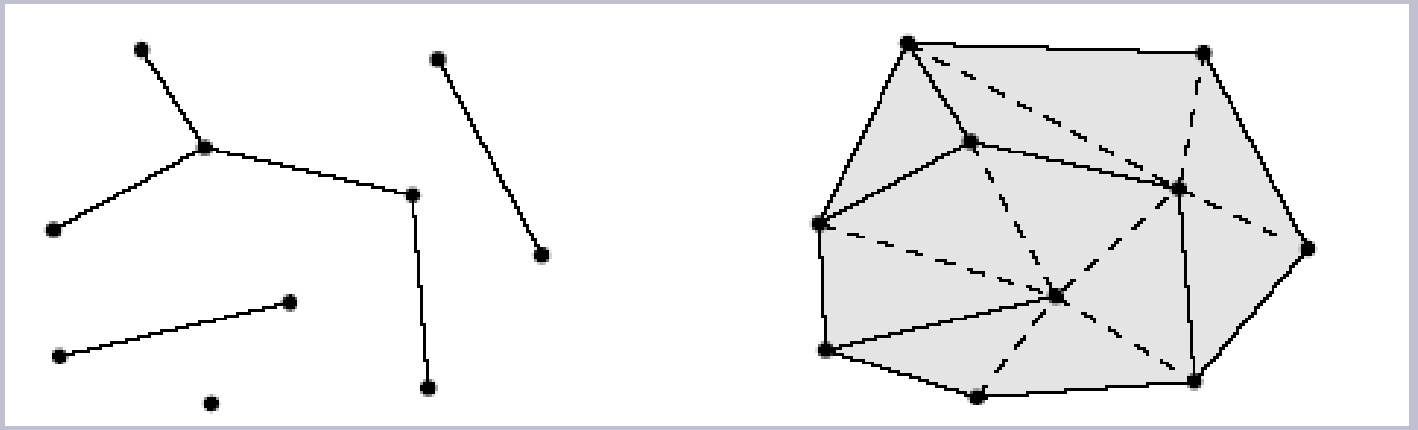
- Empty circle property





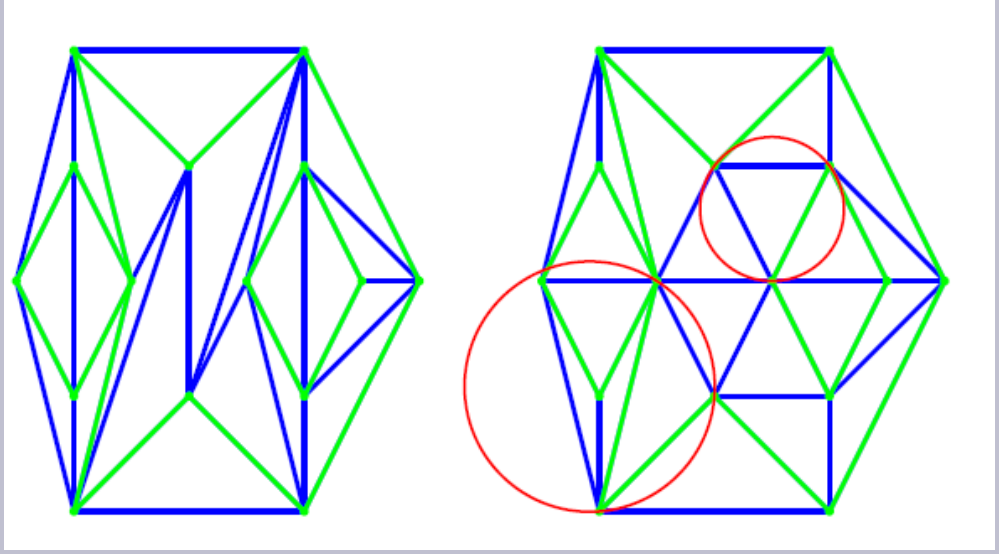
Constrained Triangulation

- Allows to enforce edges.



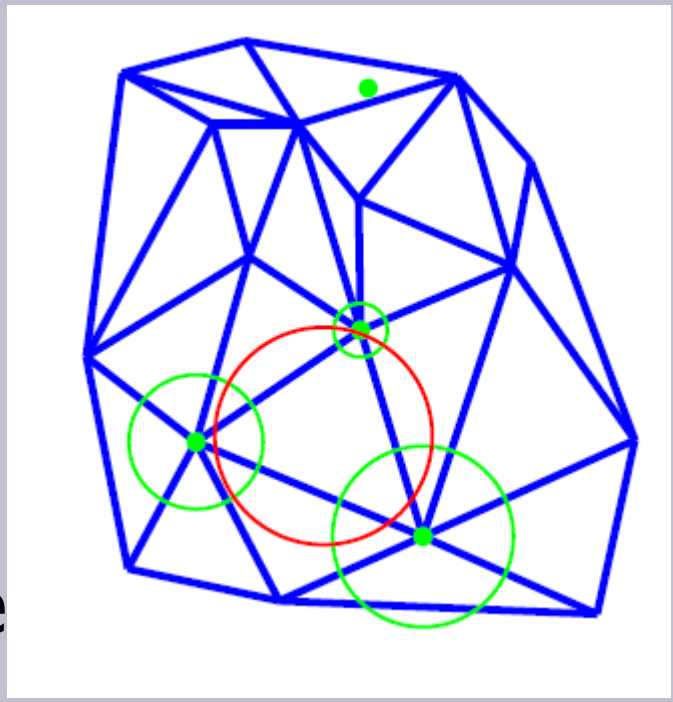
Constrained Delaunay Triangulation

- Constrained triangulation which is *as much Delaunay as possible*. Each triangle satisfies the constrained empty circle property : its circumscribing circle encloses no vertex visible from the interior of the triangle, where enforced edges as visibility obstacles.



Regular Triangulation

- Generalization of Delaunay triangulation.
- Defined for a set of *weighted points*. Each weighted point can be considered as a circle whose square radius is equal to the weight. The regular triangulation is the dual of the power diagram.

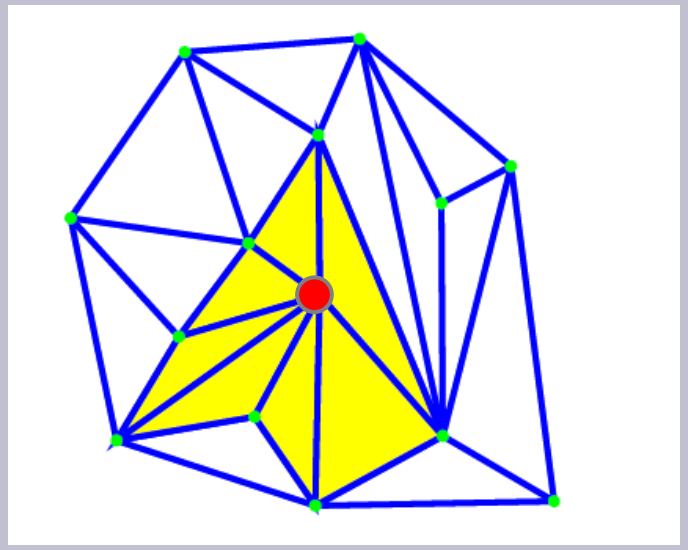


Traversal of the Triangulation

- Iterators
  - All/Finite faces iterator
  - All/Finite vertices iterator
  - All/Finite edges iterator

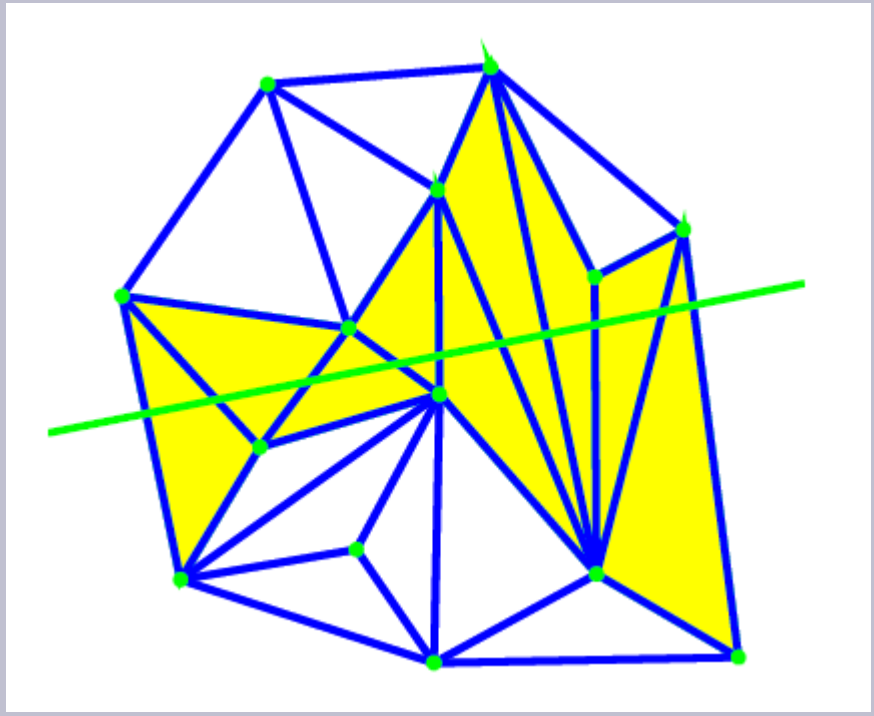
Traversal around a Vertex

- Circulators
  - Face circulator
    - faces incident to a vertex
  - Edge circulator
    - edges incident to a vertex
  - Vertex circulator
    - vertices incident to a vertex

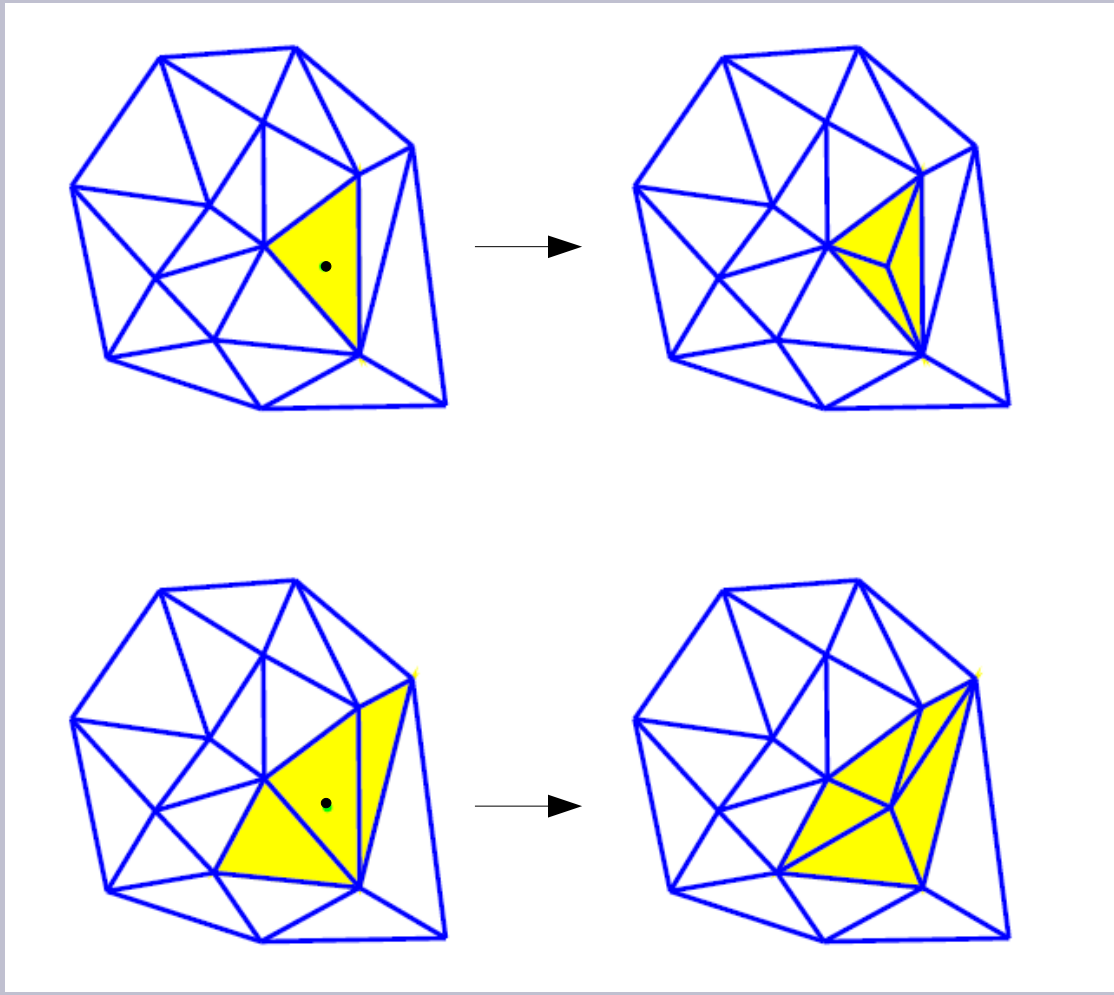


Traversal along a Line

- Line face circulator



Insertion



Representation

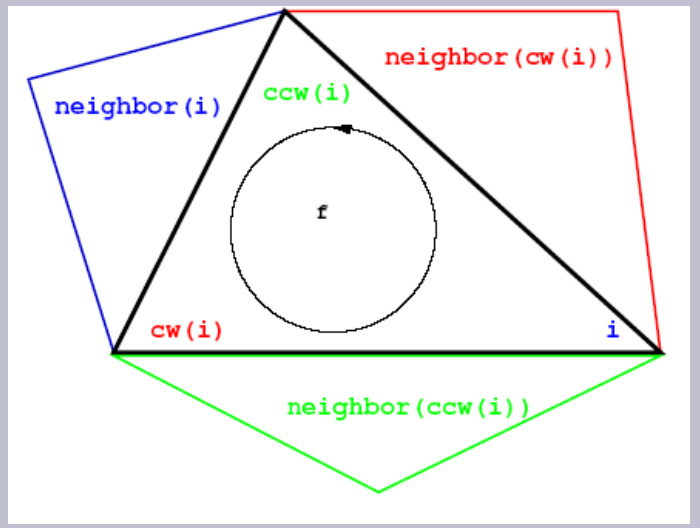
Representation

- The internal representation is based on *faces* and *vertices*.
- Edges are implicitly represented
- **`Vertex`**

  `Face_handle face()`
- **`Face`**

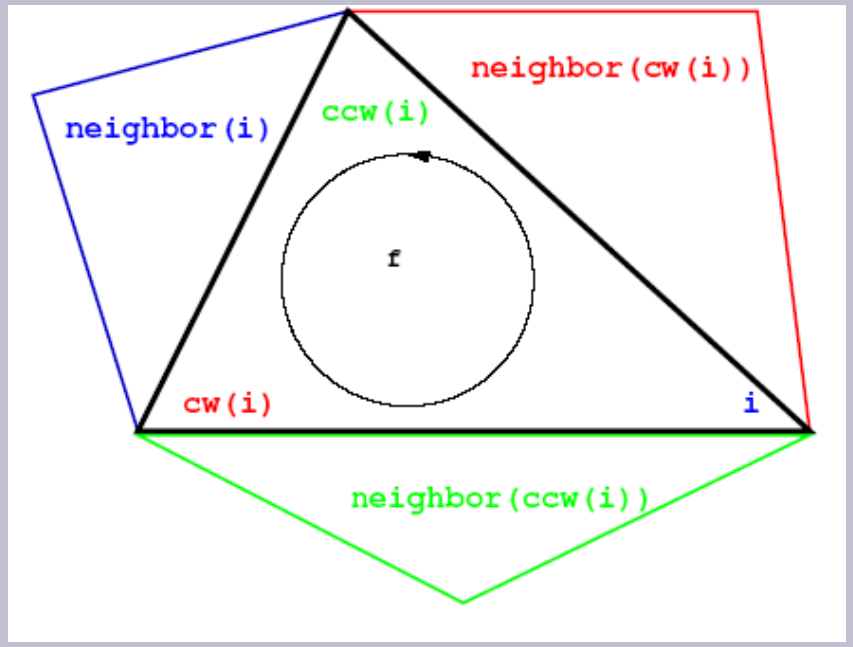
  `Vertex_handle vertex(int)`

  `Face_handle neighbor(int)`



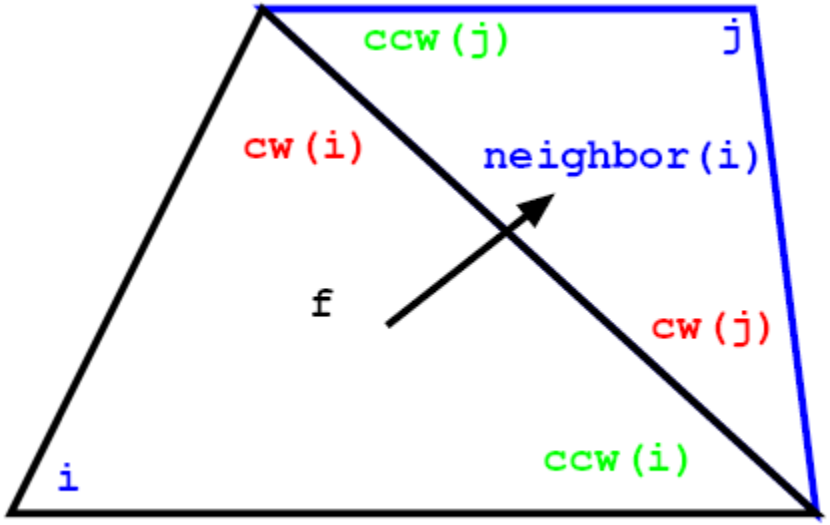
Representation

- functions cw(i) and ccw(i)



Representation

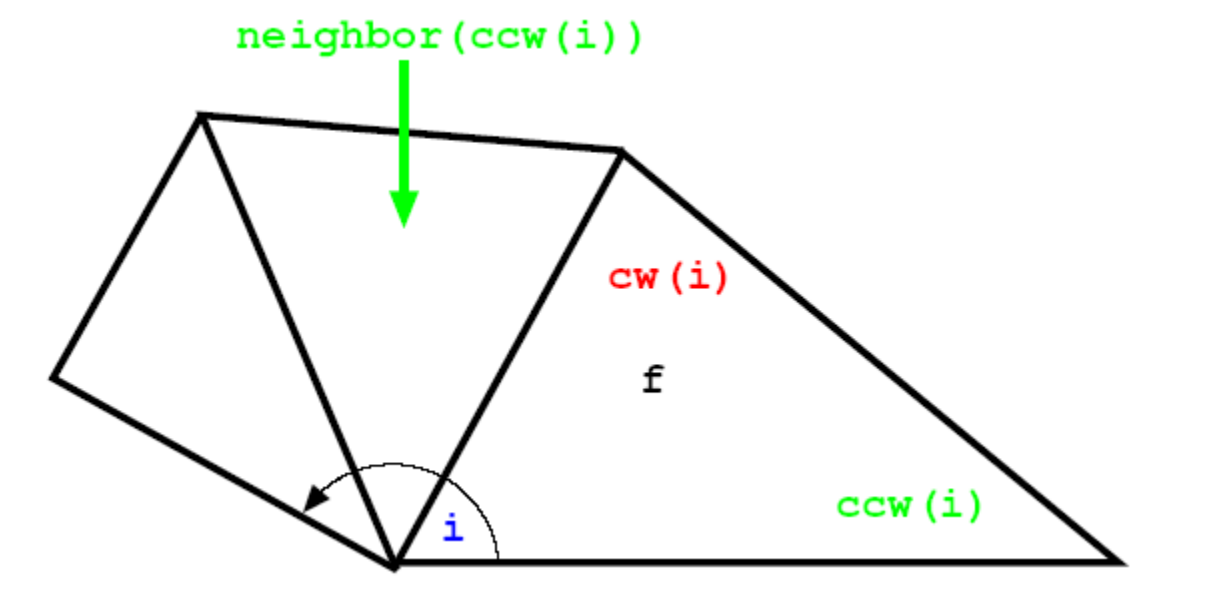
- From one Face to Another




```
n = f->neighbor(i)
j = n->index(f)
```

Representation

- Around a vertex



Software Design

Standard Template Library

```
template <Key, Less>
class set {
  insert(Key k)
  {
     if Less(k, treenode.key)
       insertLeft(k);
     else
       insertRight(k);
  }
};
```

Parameterization in CGAL

```
template < Geometry >
class Delaunay_triangulation_2 {


    void insert(Geometry::Point t) {

       if(Geometry::orientation(p,q,t)== ...) {...}

       if(Geometry::incircle(p,q,r,t)) {...}
    }
};
```

Triangulation Classes

Triangulation_2<Traits, TDS>

Triangulation_3<Traits, TDS>

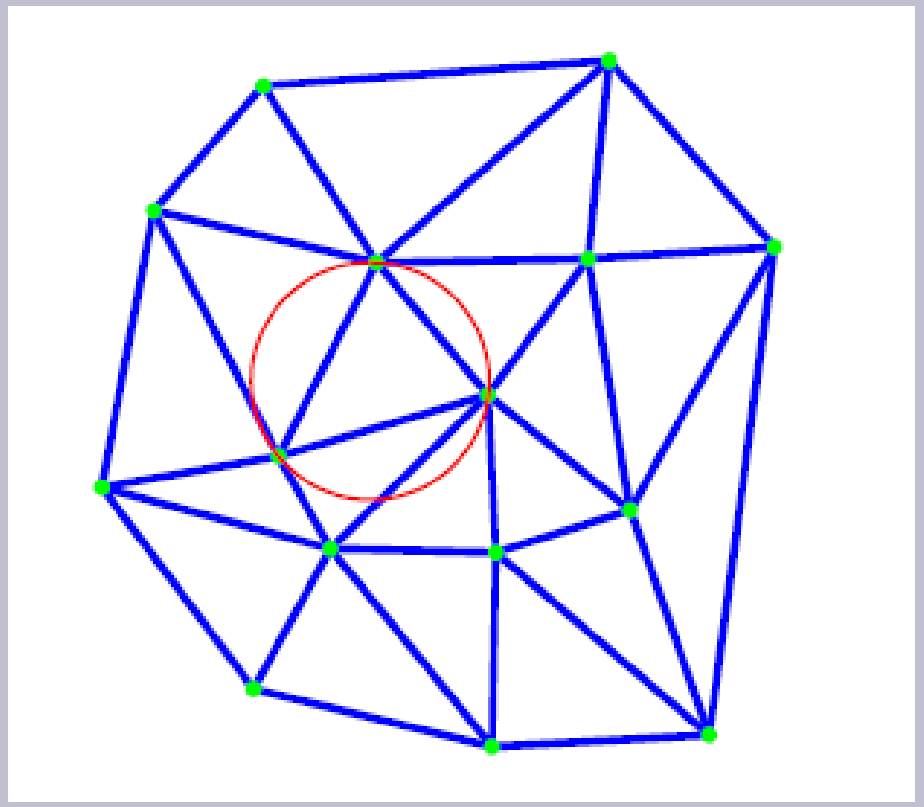
- Traits
  - Geometric traits
- TDS
  - Triangulation Data Structure
  - Provides storage, incidence, combinatorics

Geometric Traits

- Geometric traits classes require/provide:
  - Basic geometric objects
  - Predicates and constructions
- Requirements for traits are documented
  - basic library data structures and algorithms can be used with user-defined objects

DelaunayTriangulationTraits_2

- Types:
  - Point, Segment, Triangle
  - Line, Ray (for Voronoi)
- Predicates:
  - orientation test
  - in circle test
  - comparison of coordinates
- Constructions (for Voronoi):
  - circumcenter
  - bisector



DelaunayTriangulationTraits_2

- Some models for this concept
  - typedef Cartesian<double> Kernel;
  - typedef Filtered_kernel<Cartesian<double> > Kernel;
  - typedef Exact_predicates_inexact_constructions_kernel Kernel;
  - typedef Exact_predicates_exact_constructions_kernel Kernel;
- typedef Delaunay_triangulation_2<Kernel> Delaunay;

Geometric Traits Class for Terrains

Needs

- 3D points
- orientation in 2D projection
- in circle in 2D projection
- comparison on x and y coordinates

Triangulation_euclidean_traits_xy_3<Kernel>

Definition

```
typedef Filtered_kernel<Cartesian<double> > Kernel;
typedef Triangulation euclidean_traits_xy_3<Kernel> Traits;
typedef Delaunay_triangulation 2<Traits> Triangulation;
```

Triangulation Data Structure

template <class Vb, class Fb>
class Triangulation_data_structure_2

Provides types

- Vertex, Face
- Vertex_handle, Face_handle
- Face_iterator, Edge_iterator, Vertex_iterator
- Face_circulator, Edge_circulator, Vertex_circulator

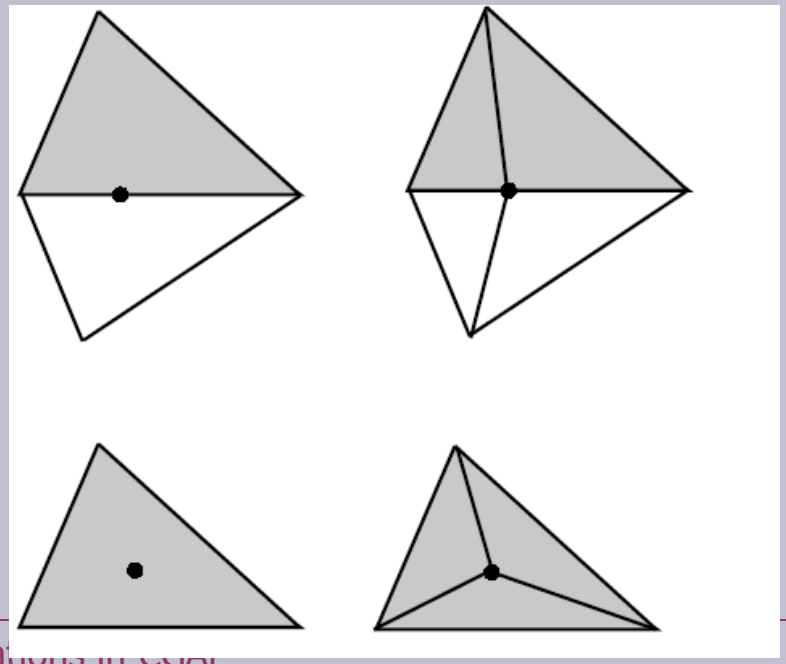
Does neither deal with coordinates nor with infinity

Combinatorial Operations

void insert_in_face (Vertex_handle v,Face_handle f)

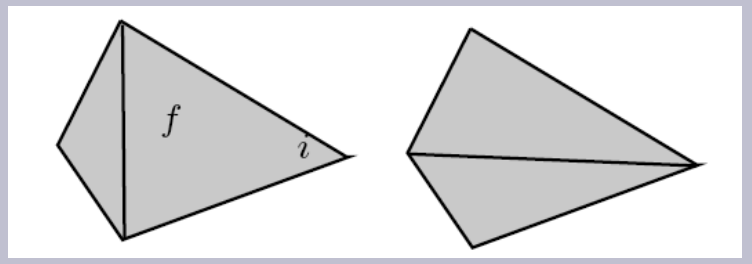
void insert_in_edge (Vertex_handle v, Face_handle f, int i)

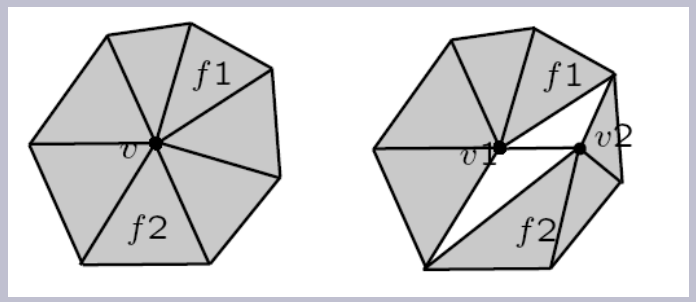
void remove_degre_3 (Vertex_handle v);



Combinatorial Operations

```
void flip (Face_handle f, int i);

void
split vertex (Vertex_handle,
              Face_handle f1,
              Face_handle f2)

void
join vertices (Vertex_handle v1,
               Vertex_handle v2)
```





The Triangulation Class

CGAL::Triangulation 2<Gt, Tds >

typedef Gt geometric_traits;
typedef Tds Triangulation_data_structure;
typedef Triangulation_2<Gt, Tds > Triangulation;

Types
Gt::Point_2
Gt::Segment_2
Gt::Triangle_2
Triangulation::Vertex inherits from Tds::Vertex
Triangulation::Face inherits from Tds::Face
Triangulation::Vertex_handle
Triangulation::Face_handle

typedef pair<Face handle, int> Edge ;
Triangulation::Face_iterator
Triangulation::Edge_iterator
Triangulation::Vertex_iterator
Triangulation::Line_face_circulator
Triangulation::Face_circulator
Triangulation::Edge_circulator
Triangulation::Vertex_circulator

Algorithmic Issues

Point Location

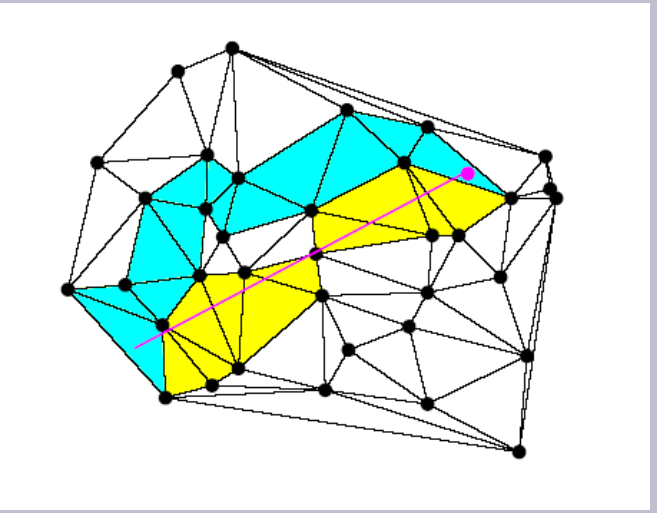
```
enum Locate_type { VERTEX=0,
                   EDGE,
                   FACE,
                   OUTSIDE_CONVEX_HULL,
                   OUTSIDE_AFFINE_HULL }
```

```
Face_handle locate(Point query,
                   Locate_type& lt,
                   int& li,
                   Face_handle h =Face_handle() );
```

Point Location

- All CGAL triangulations are built through *incremental* on-line insertion of points.
- The main algorithmic issue is therefore to deal with *point location*.

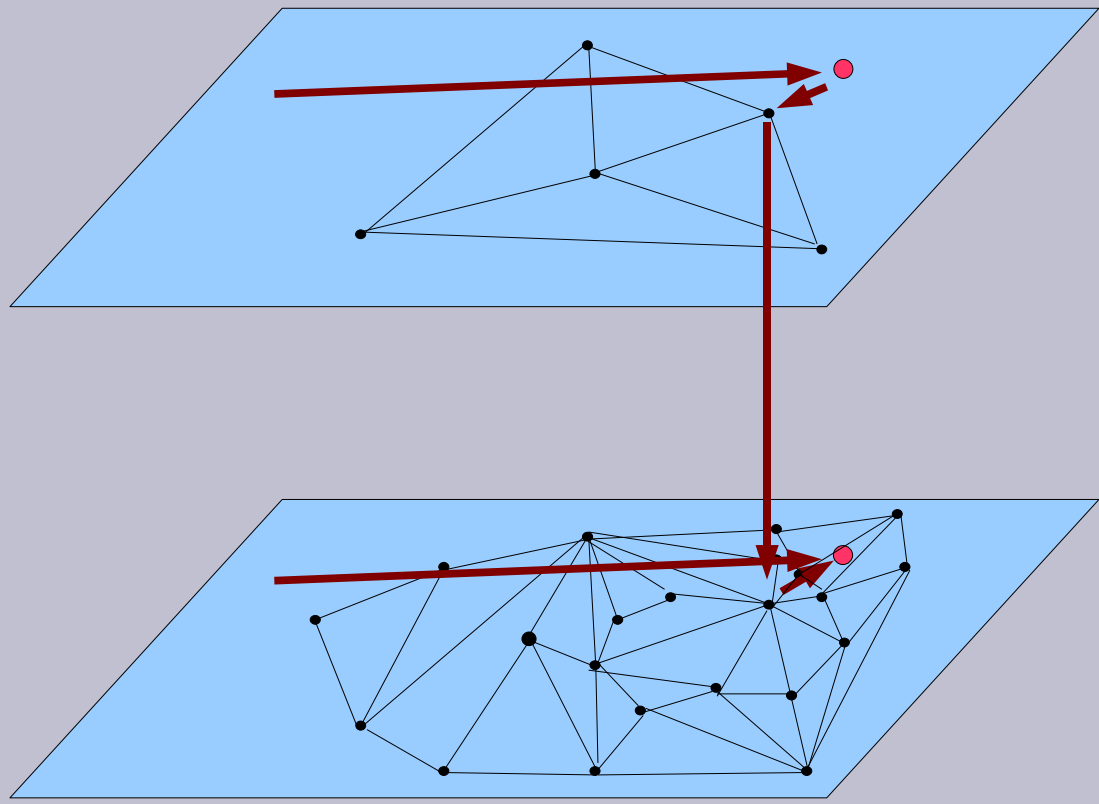
- CGAL offers different algorithms :
  - linewalk
  - Zigzag walk
  - the Delaunay hierarchy



Efficient Point Location

Hierarchical
search structure

Transparent
for the user



Insertion

Vertex_handle **insert** (Point p);

If you exactly know where you are:
Vertex_handle **insert** (Point p, Locate_type,
int i, Face_handle fh);

If you roughly know where you are:
Vertex_handle insert(Point p, Face_handle hint);

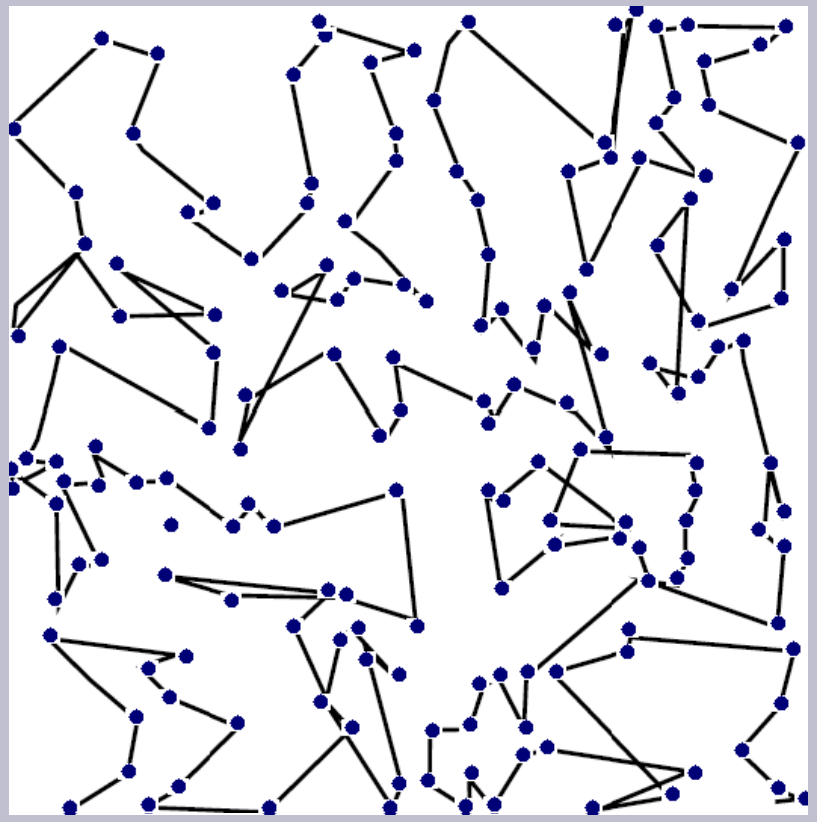
Insertion

Points are better in random order

```
std::vector<Point> points = ...;
std::random_shuffle(points.begin(),
                    points.end());
```

... or ordered along a Hilbert curve

```
CGAL::spatial_sort(points.begin(),
                   points.end());
```



Delaunay: Conflict Zone

The faces that are modified, in case one inserts a point

```
template <class OutputItFaces, class OutputItBoundaryEdges>
std::pair<OutputItFaces,OutputItBoundaryEdges>
get_conflicts_and_boundary ( Point p,
                             OutputItFaces fit,
                             OutputItBoundaryEdges eit,
                             Face_handle start  );
```

Examples for fit: std::backinserter(myFaceList)
or CGAL::Emptyset_iterator

Adding Information

Because TDS::Vertex and TDS::Face are concepts and

```
template <class VB, class FB >
class Triangulation_datastructure_2;
```

```
template <class Info, class VB>
class Triangulation_vertex_base_with_info_2;
```

Develop your own models of vertex/face base concept

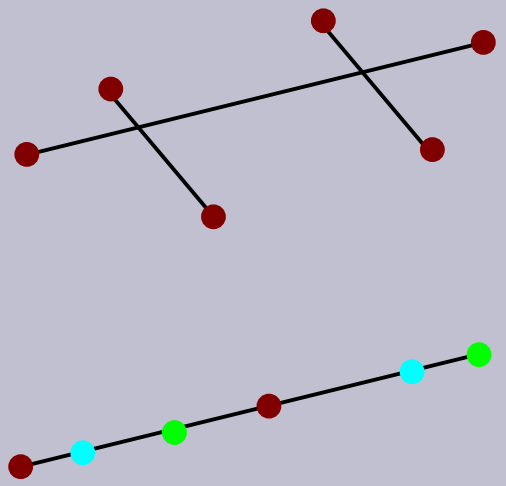
Voronoi Diagram

```
Delaunay_triangulation_2 dt;
Face_handle fh = ...
Point_2 p = dt.dual(fh);

Edge e = ...
Segment_2 segment;
Ray_2 ray;

CGAL::Object voronoi_edge = dt.dual(e);
if(CGAL::assign(segment, voronoi_edge))   {
} else {
  CGAL::assign(ray, voronoi_edge);
}
```

Constrained Triangulations

```
class Constrained_triangulation_2 {
bool      is_constrained( Edge )
void      ct.insert_constraint ( Point a, Point b )
void      ct.insert_constraint ( Vertex_handle va, Vertex_handle vb )
}
```

```
class Constrained_triangulation_2_plus {
  typedef ... Subconstraint_iterator;
  typedef ... Context;

}
```



3D Triangulations

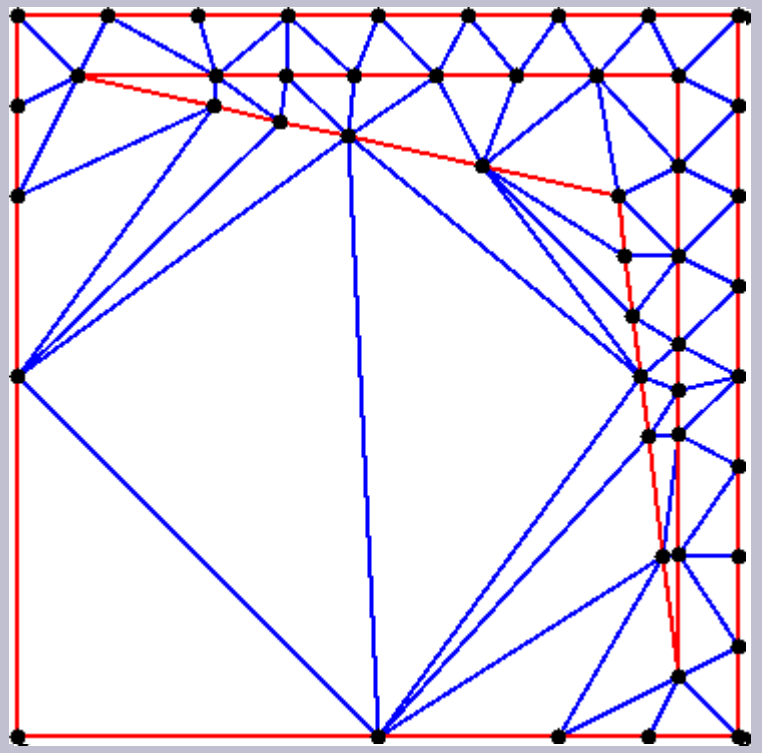
- Conceptually identically to 2D triangulation
- Cells instead of faces
- No handling of constraints yet

Mesh Generation

Conforming Triangulation

Add points on constraints such that they satisfy

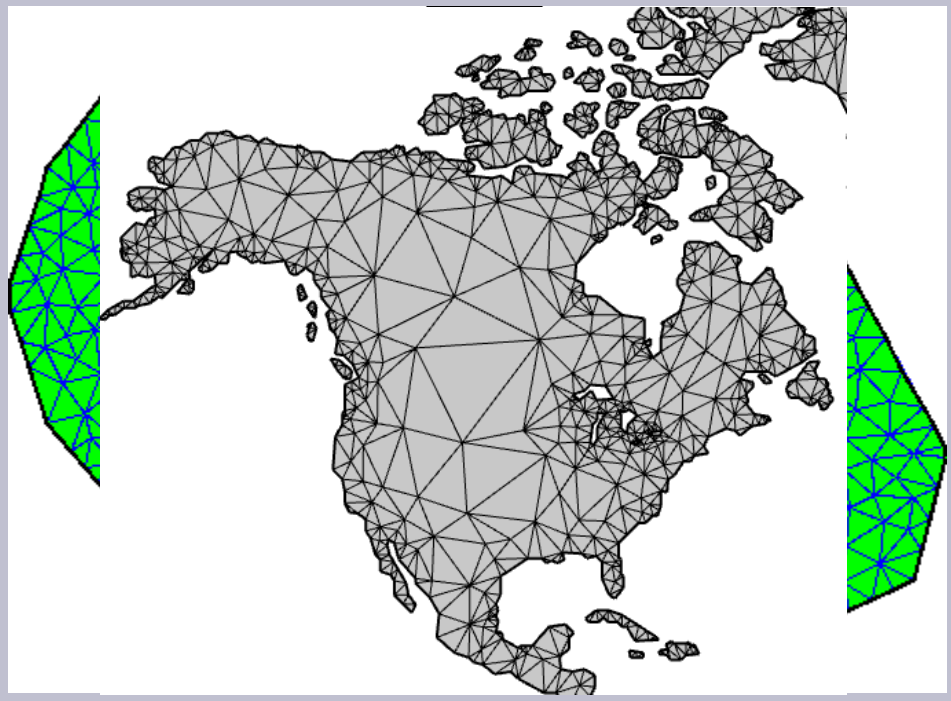
- Delaunay criterium
- Gabriel criterium



Delaunay Mesh

Add points on constraints and in faces such that they satisfy

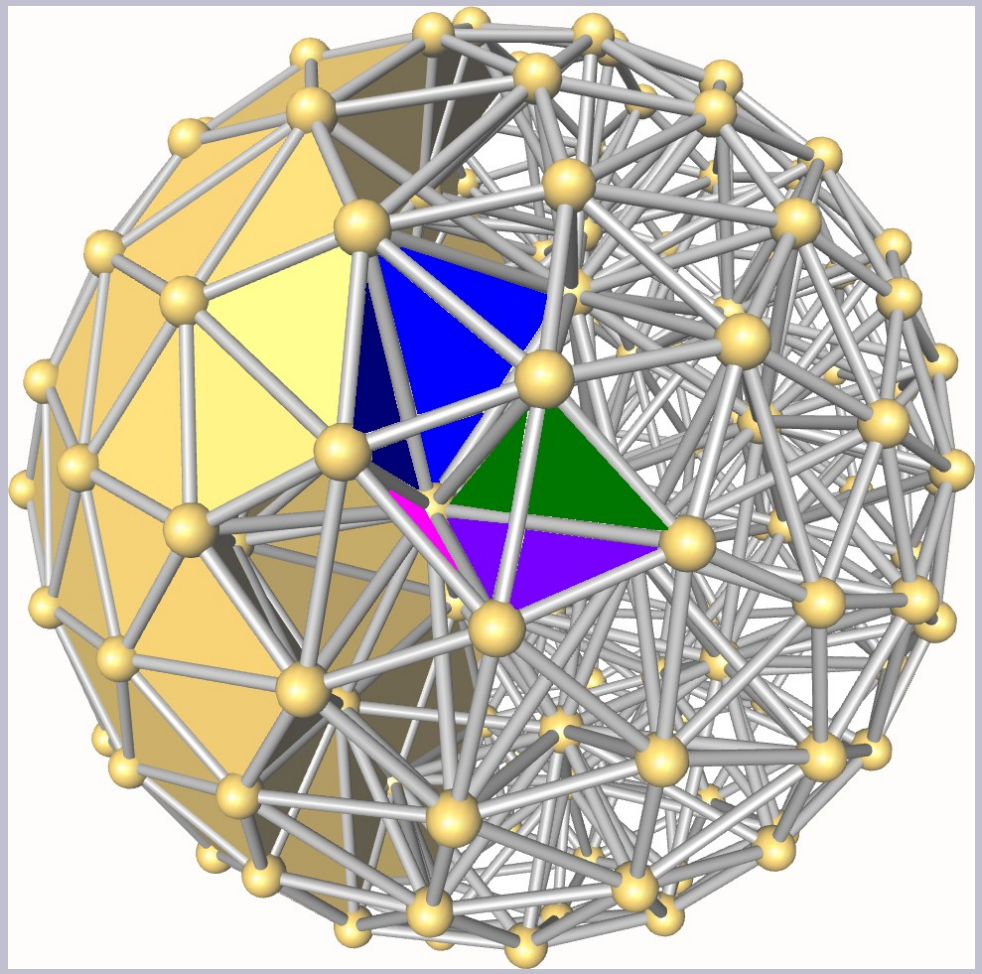
- size criterium
- shape criterium
- ...



2D Complex in 3D Triangulations

- Subset of facets
- Surface is non self-intersecting
- May be manifold
- API for exploring the structure

- Output of the CGAL surface mesher



3D Mesh Generation

- Surface mesh
  - of implicit surface
  - from image
  - for polyhedral mesh*
- Volume mesh*