

## Mesh Generation



Andreas Fabri  
GeometryFactory



Pierre Alliez  
Inria



# Outline

- Triangle meshing with CGAL
- 2D Delaunay mesh generation
- 3D Delaunay mesh generation
  - Delaunay refinement and filtering
  - Multiple types of domains
  - Feature preserving
  - Meshing from images
  - Mesh optimization
  - API
- Isotropic tetrahedral remeshing

# Triangle Meshing with CGAL

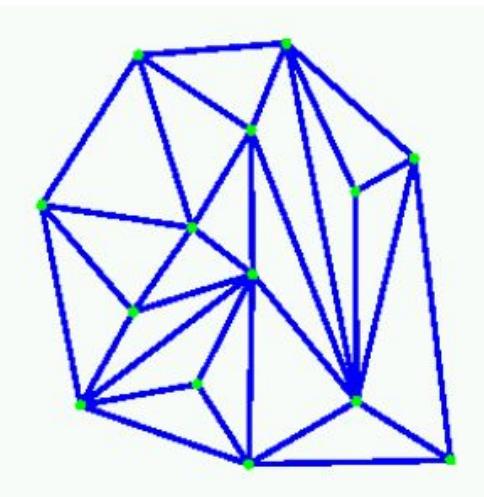
# CGAL Packages

- 2D Mesh Generation
- Surface Mesher (obsolete)
- 3D Mesh Generation
- Isotropic Remeshing

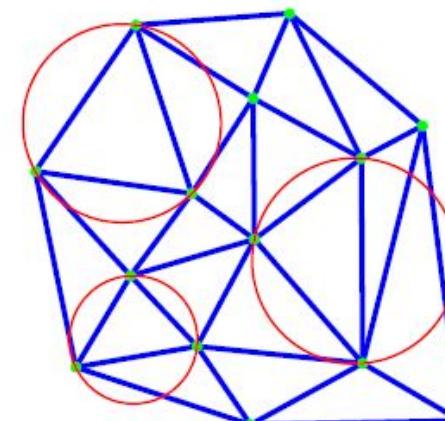
# 2D Mesh Generation

# Delaunay Triangulation

A triangulation is a Delaunay triangulation if the circumscribing circle of any facet of the triangulation contains no vertices in its interior

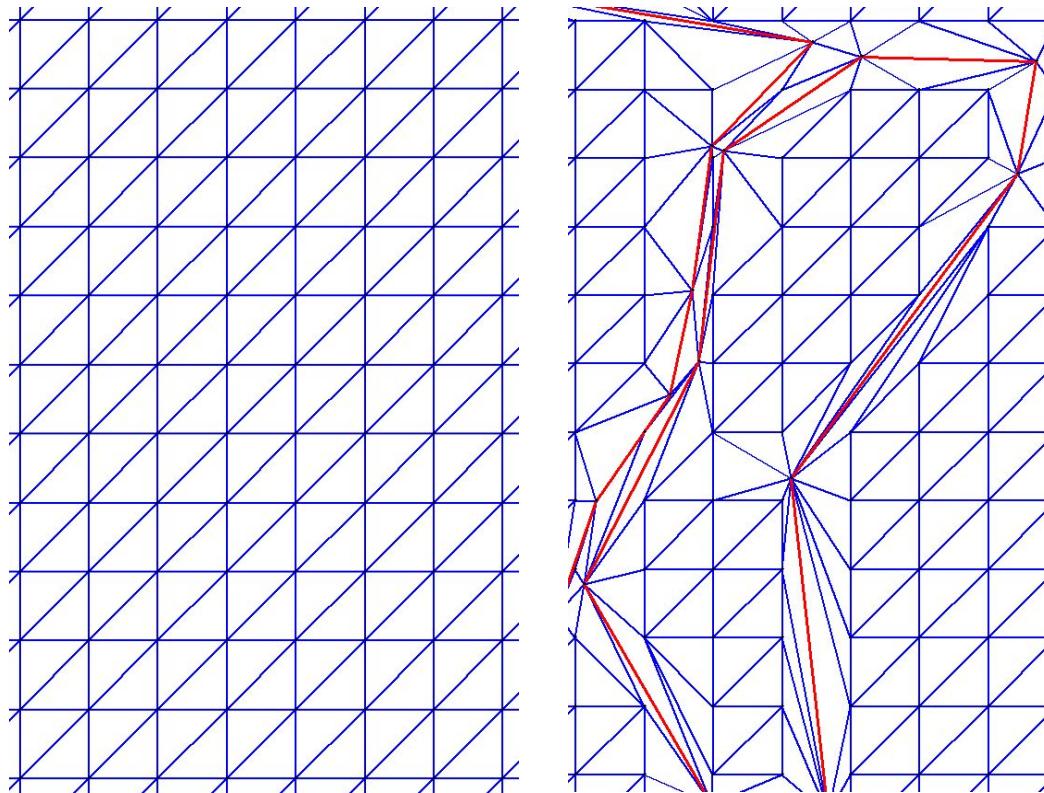


non-Delaunay

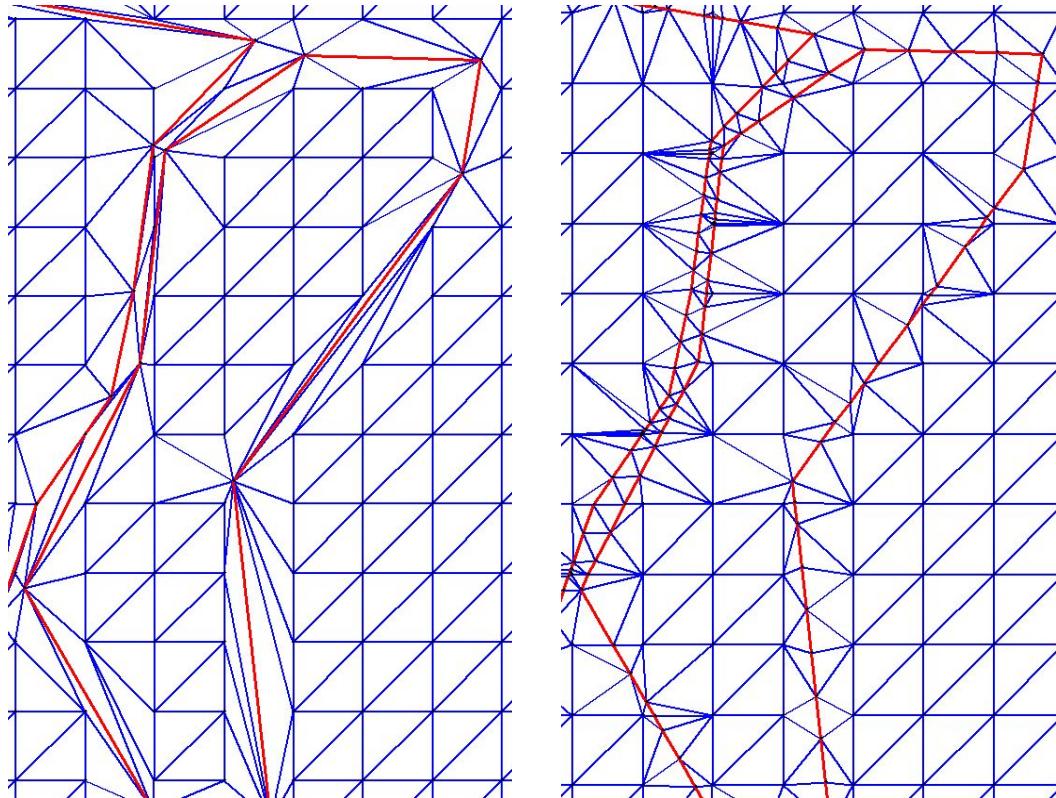


Delaunay

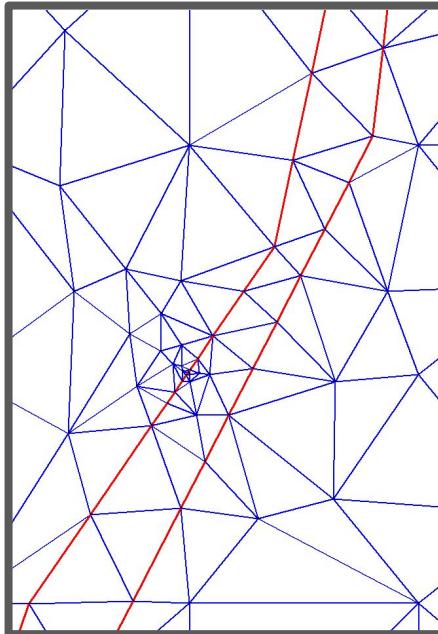
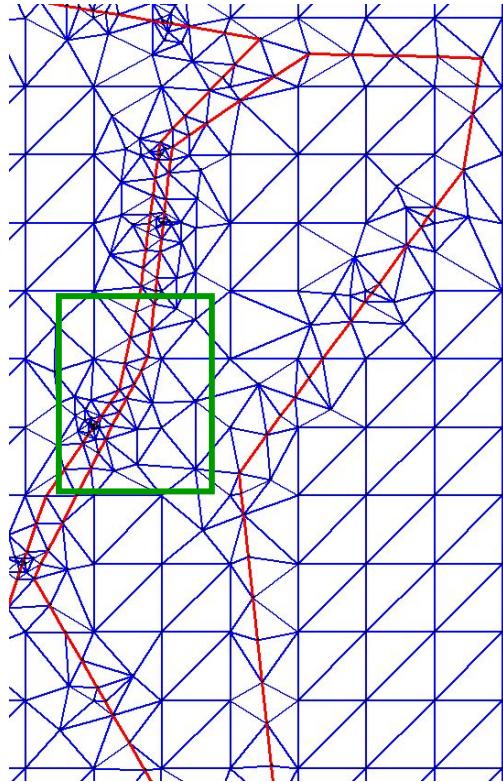
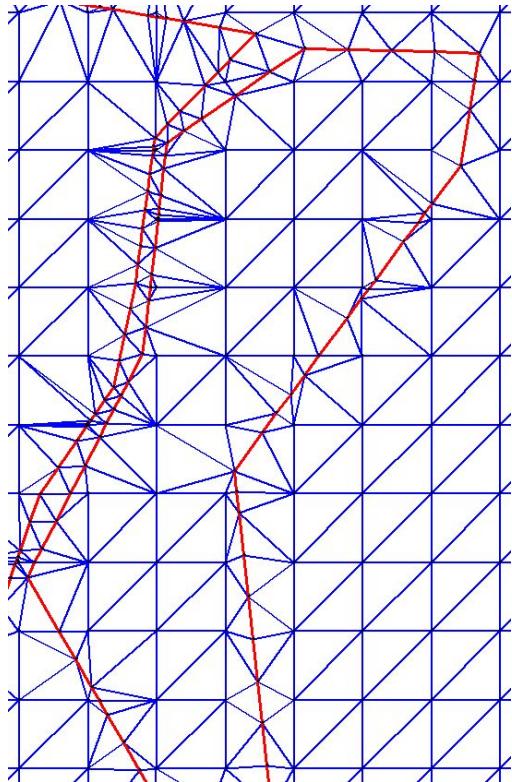
# Constrained Delaunay Triangulation



# Conforming Triangulation

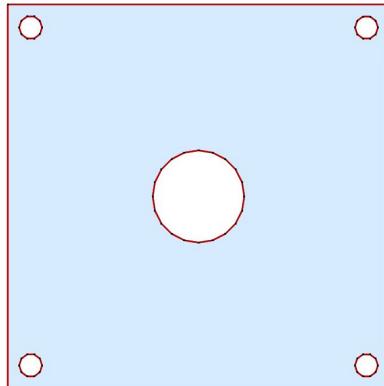
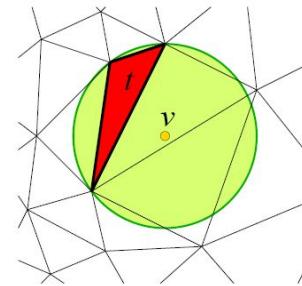


# Delaunay Mesh Refinement

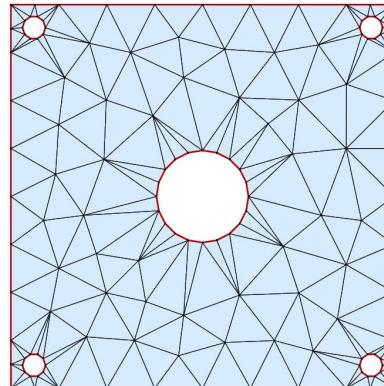


# Parameter: Shape

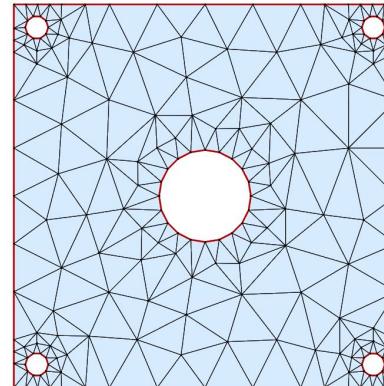
Lower bound on smallest angle



Input PSLG

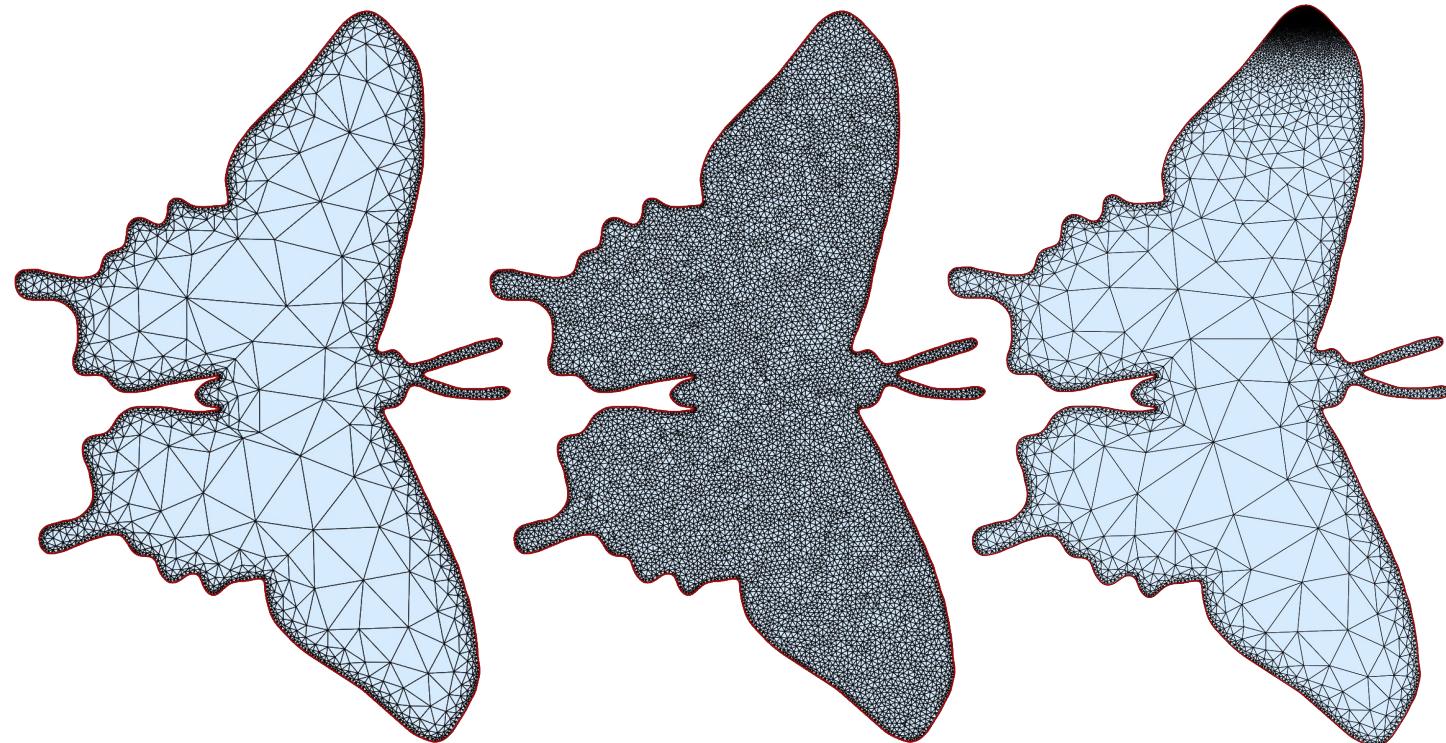


5 deg



20.7 deg

# Parameter: Size



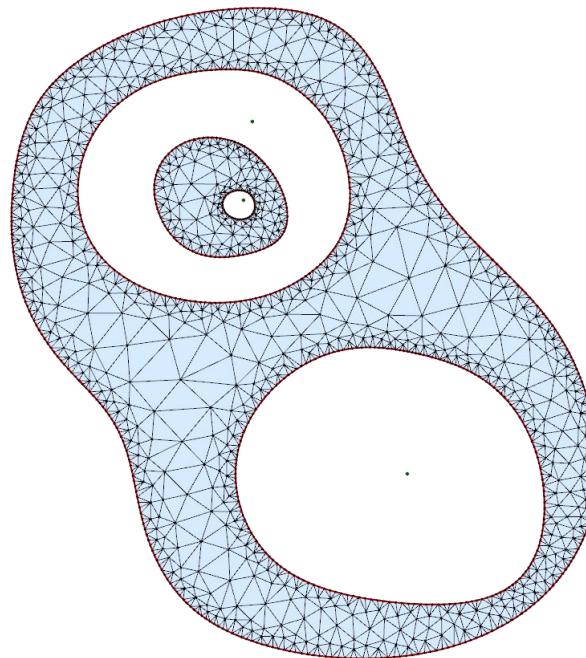
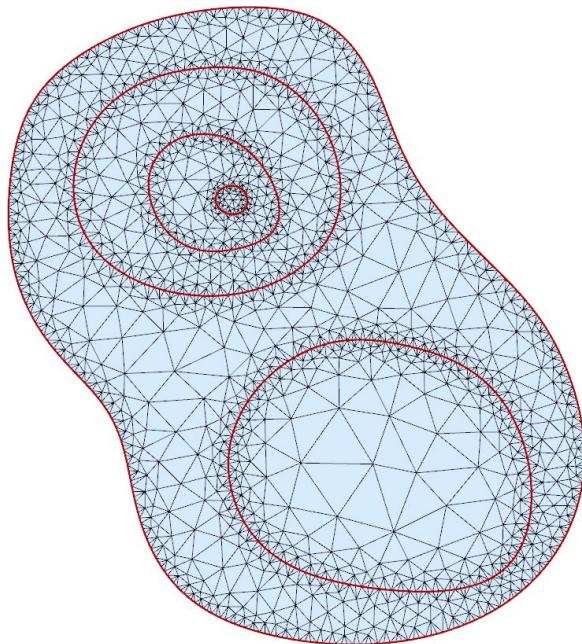
No constraints

Uniform Sizing

Sizing function

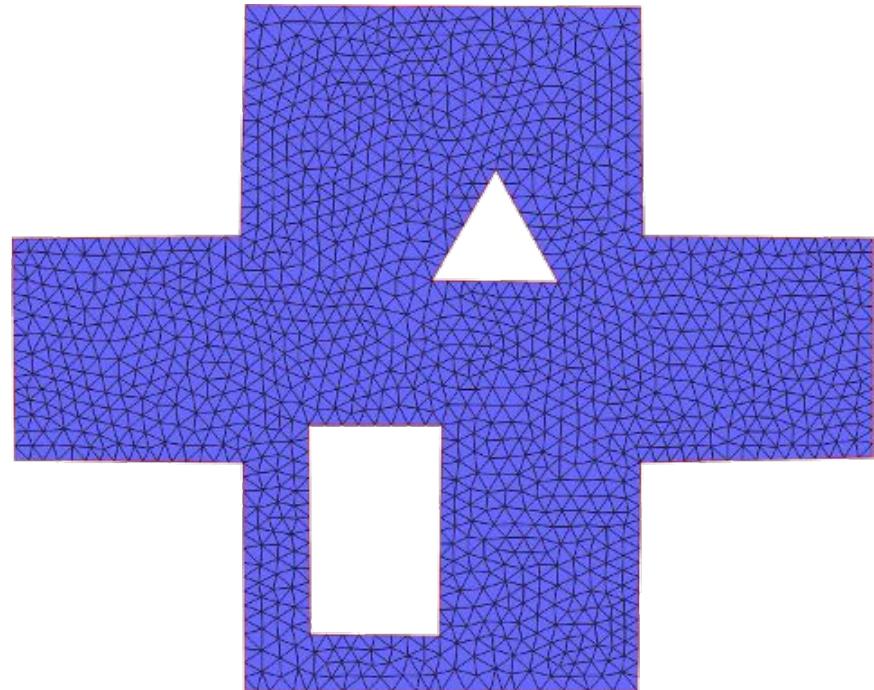
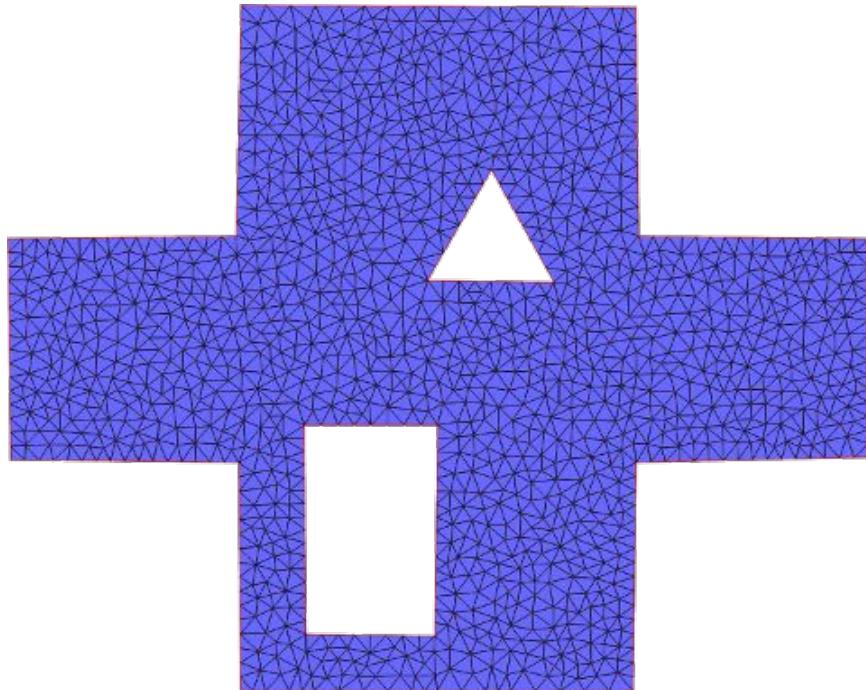
# Parameter: Seed points

Include and exclude components



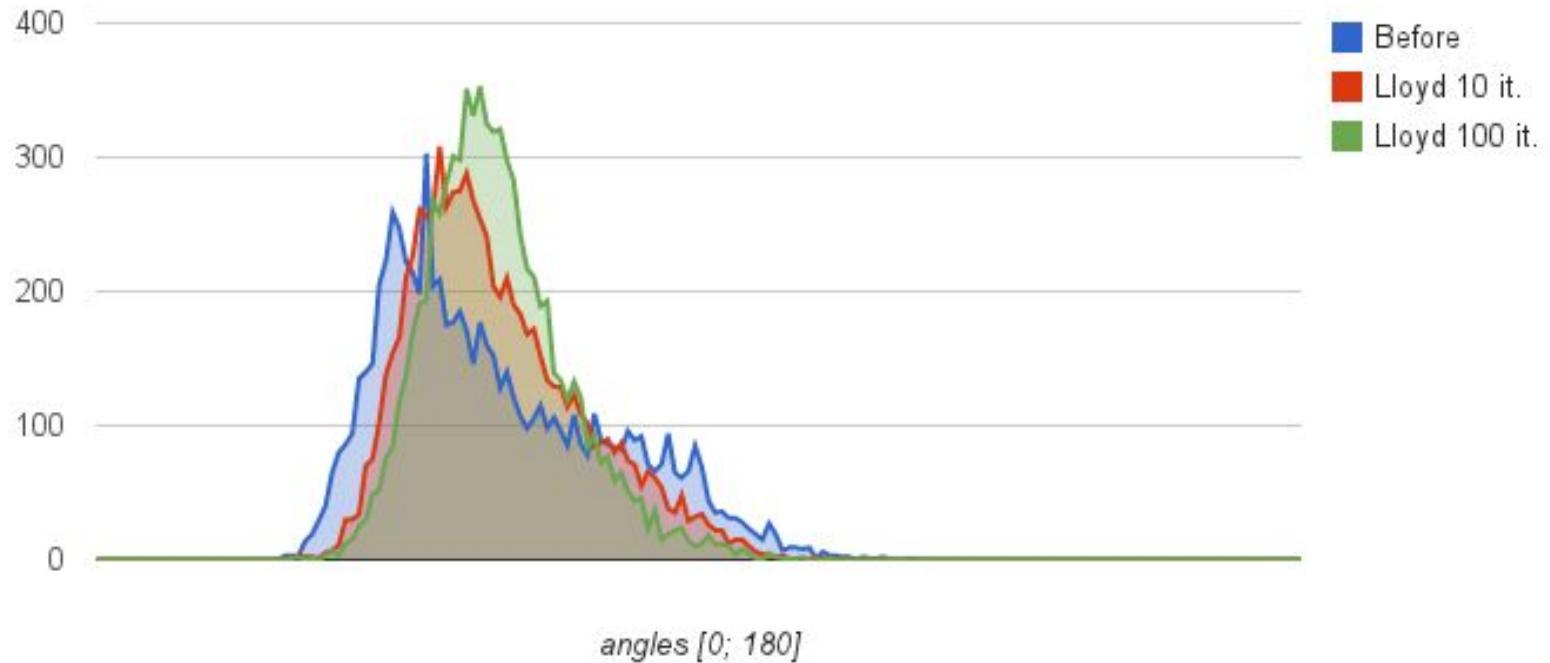
DEMO

# Smoothing: Lloyd Iterations



DEMO

# Smoothing: Lloyd Iterations



# Code Example

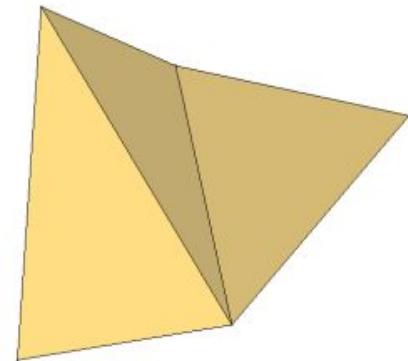
```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Constrained_Delaunay_triangulation_2.h>
#include <CGAL/Delaunay_mesher_2.h>
typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef CGAL::Constrained_Delaunay_triangulation_2<Kernel> CDT;
typedef CGAL::Delaunay_mesh_size_criteria_2<CDT> Mesh_criteria;
int main()
{
    CDT cdt;
    ... // insert points and constraints
    Mesh_criteria mesh_criteria(0.125, 0.5);
    CGAL::refine_Delaunay_mesh_2(cdt, mesh_criteria);
    return 0;
}
```

# 3D Mesh Generation

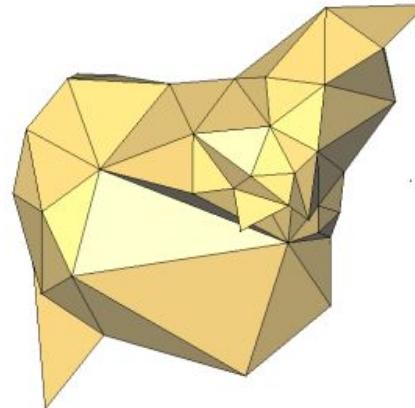
# 3D Mesh Generation

## Delaunay Refinement and Filtering

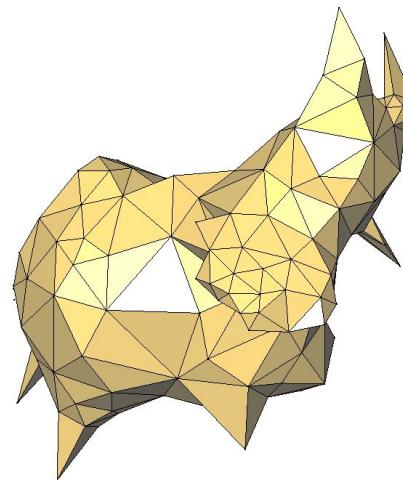
# 3D Delaunay Mesh Generation



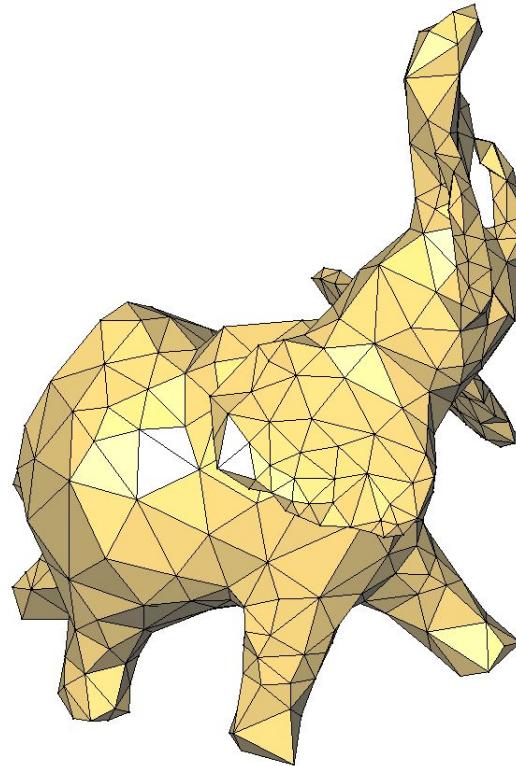
# 3D Delaunay Mesh Generation



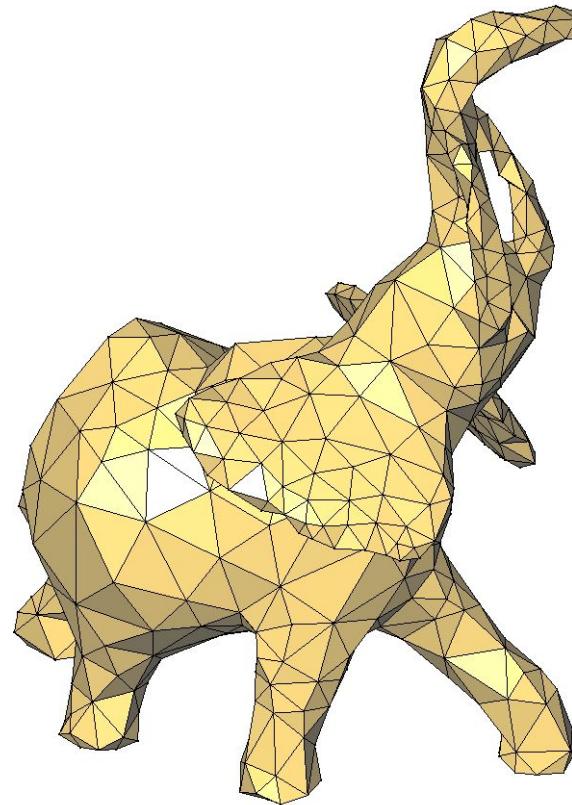
# 3D Delaunay Mesh Generation



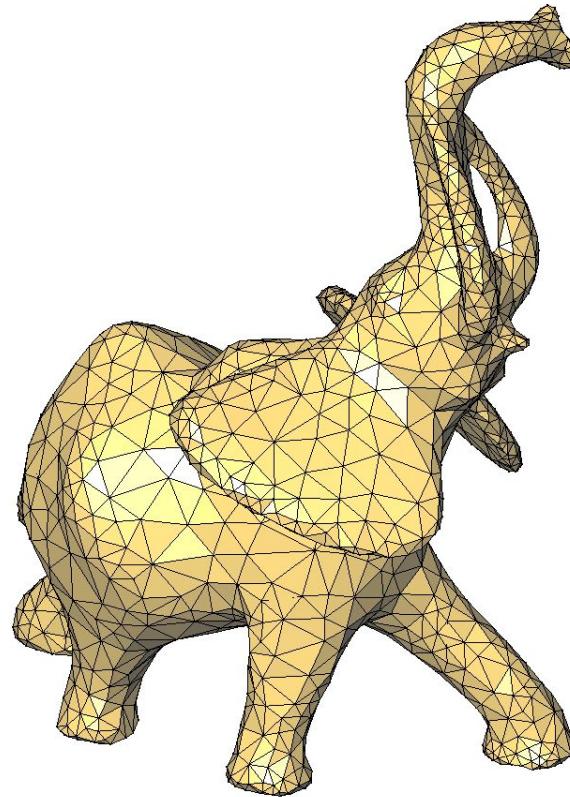
# 3D Delaunay Mesh Generation



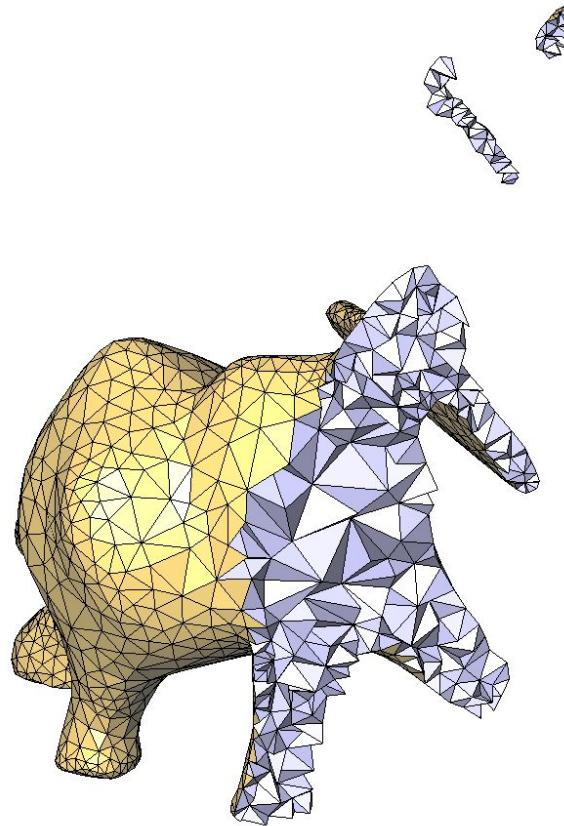
# 3D Delaunay Mesh Generation



# 3D Delaunay Mesh Generation

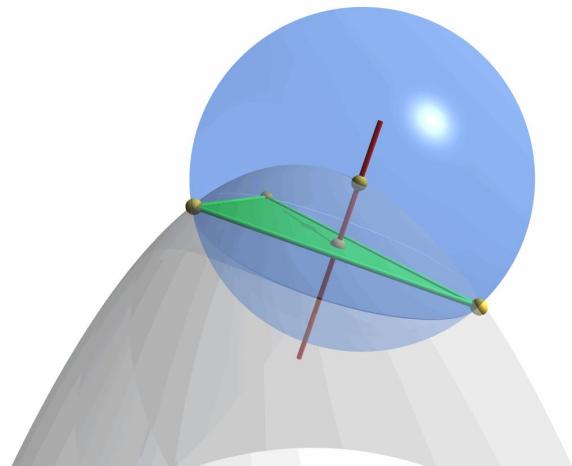


# 3D Delaunay Mesh Generation

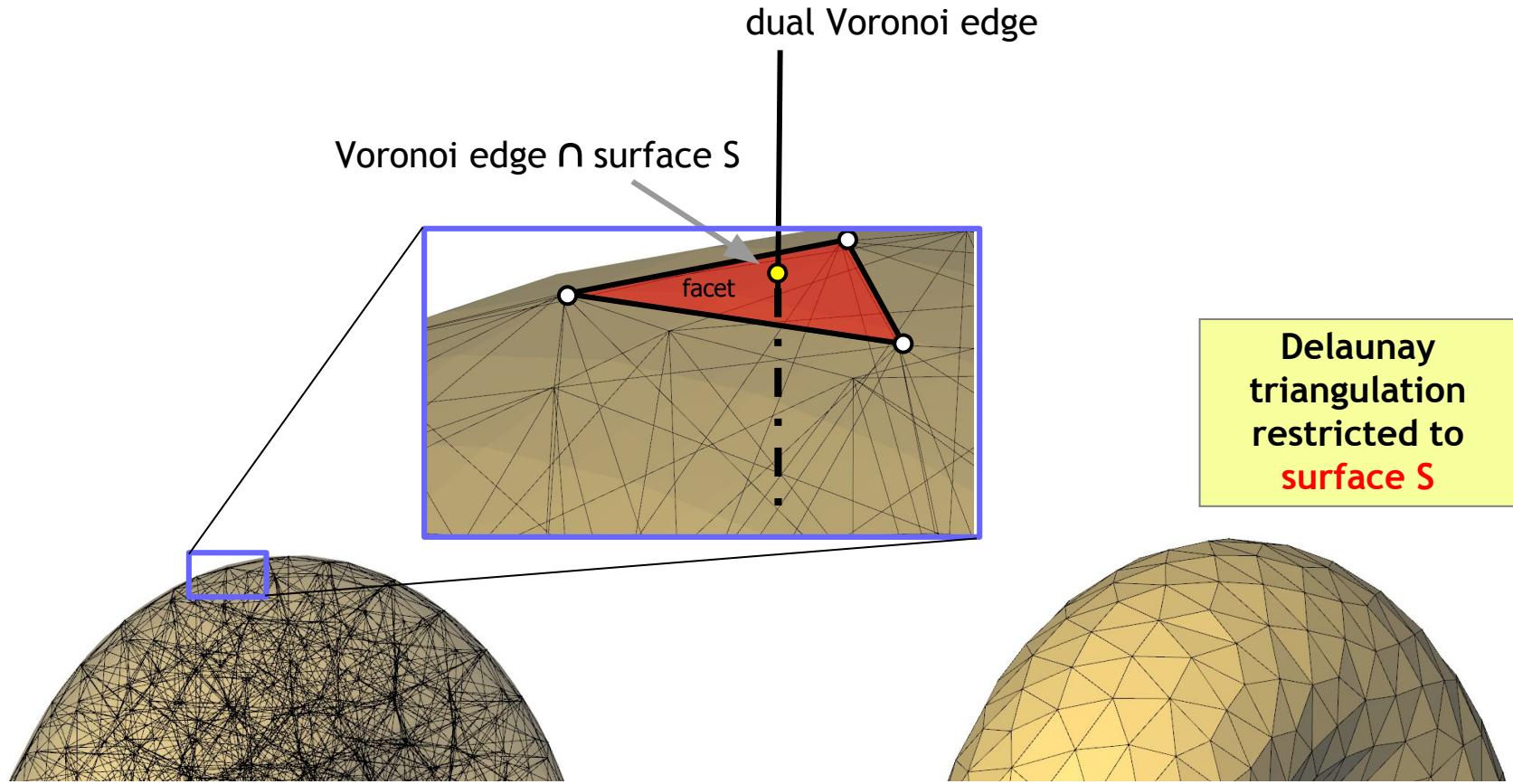


# The Algorithm

```
repeat
{
    pick worst facet f
    insert  $dual(f) \cap S$  in Delaunay triangulation
    update Delaunay triangulation restricted to S
}
until all facets are good
```



# Delaunay Filtering

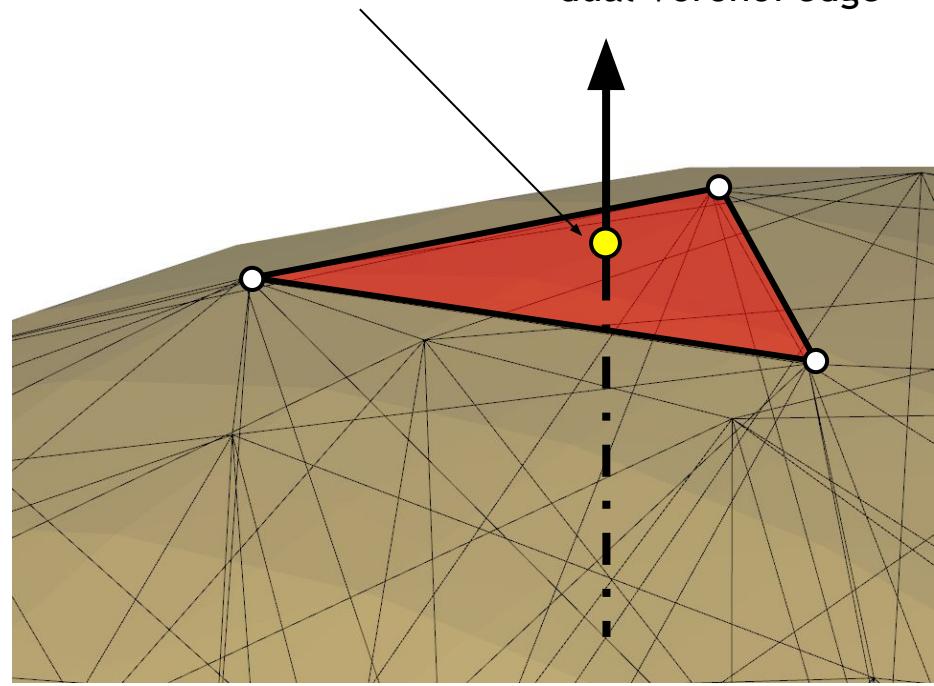


# Delaunay Refinement

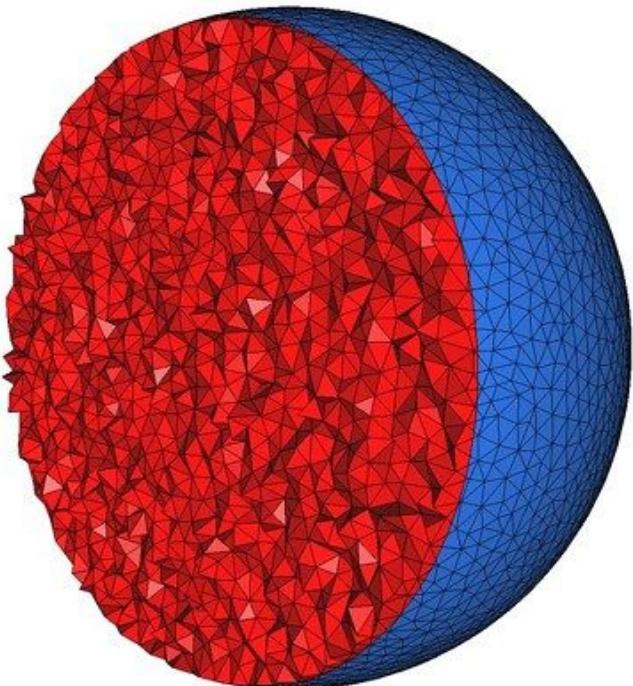
**Bad facet** = large or  
badly shaped or  
large approximation error

Refinement point

dual Voronoi edge



# Meshing an Implicit Function



```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Labeled_mesh_domain_3<K> Mesh_domain;
typedef CGAL::Mesh_triangulation_3<Mesh_domain>::type Tr;
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;
typedef CGAL::Mesh_criteria_3<Tr> Mesh_criteria;

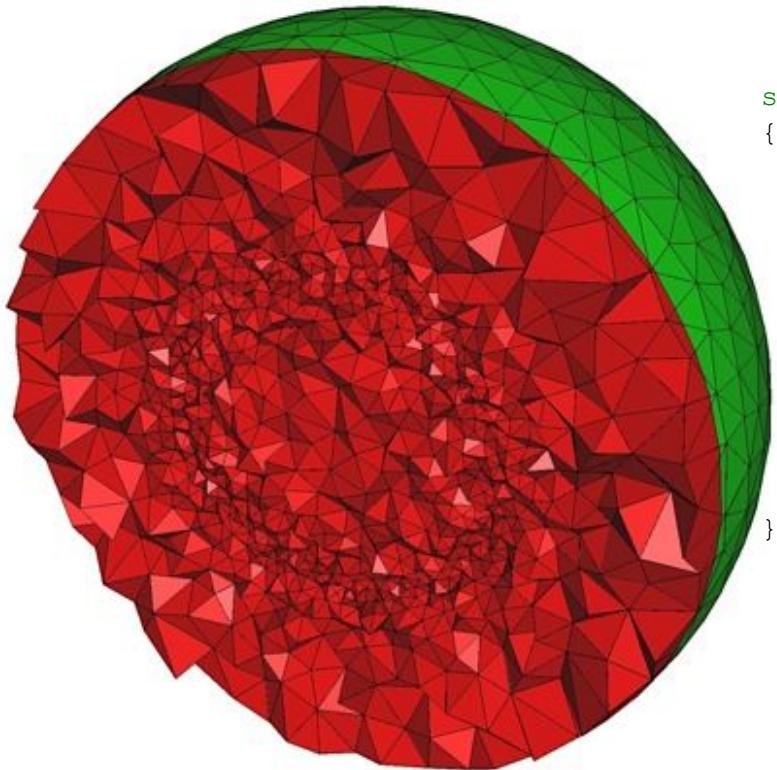
using CGAL::parameters;

int main()
{
    // Domain
    Mesh_domain domain =
        Mesh_domain::create_implicit_mesh_domain(
            function = [] (const K::Point_3& p)
            { return CGAL::squared_distance(
                p, K::Point_3(CGAL::ORIGIN))-1; },
            bounding_object = K::Sphere_3(CGAL::ORIGIN, 2.));

    // Set mesh criteria
    Mesh_criteria criteria(facet_angle=30, facet_size=0.1,
                           facet_distance=0.025,
                           cell_size=0.1);

    // Mesh generation
    C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
}
```

# Specifying a Sizing Field

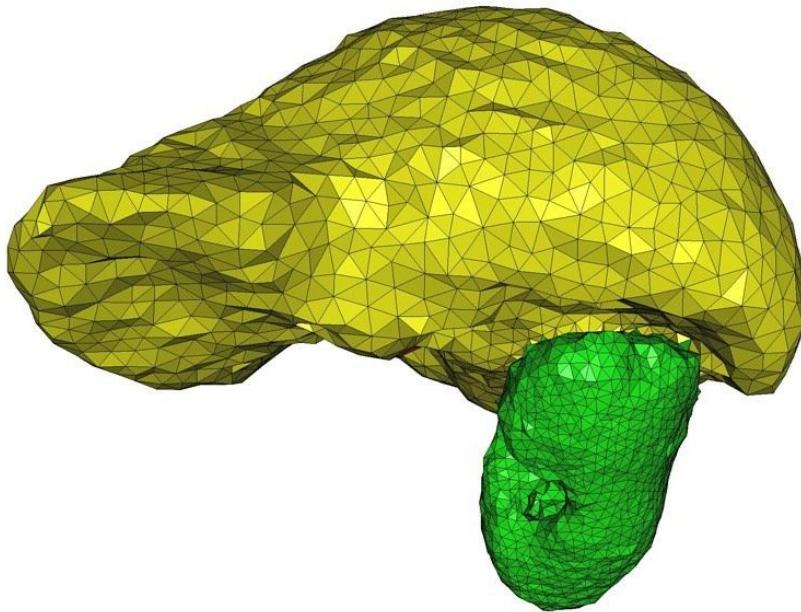


```
struct Spherical_sizing_field
{
    typedef Mesh_domain::Index Index;

    double operator() (const Point_3& p, const int, const Index&)
    {
        double sq_d_to_origin =
            CGAL::squared_distance (p, Point(CGAL::ORIGIN));

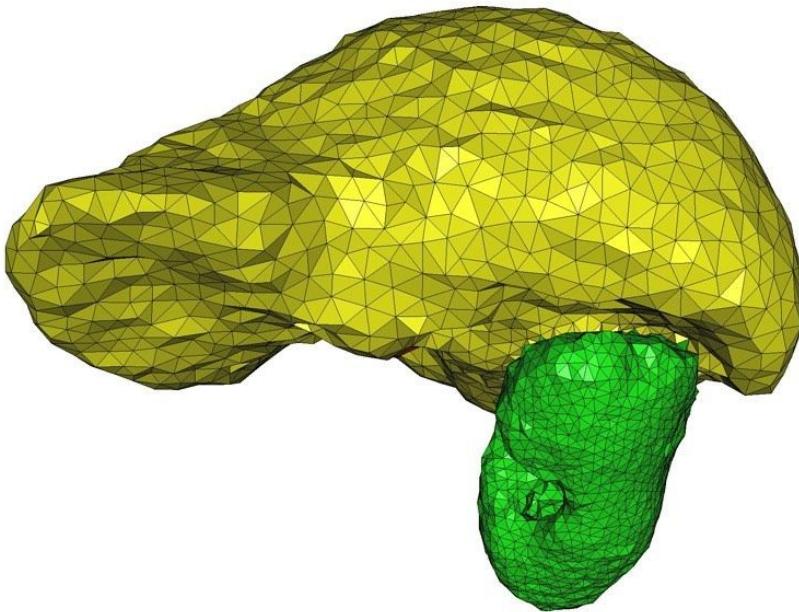
        return CGAL::abs( sqrt(sq_d_to_origin)-0.5 ) / 5. + 0.025;
    }
};
```

# Specifying a Sizing Field



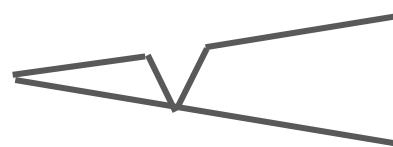
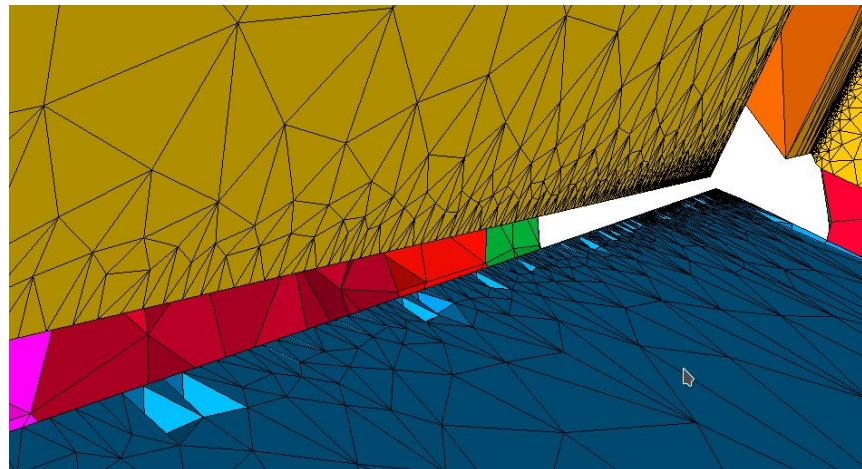
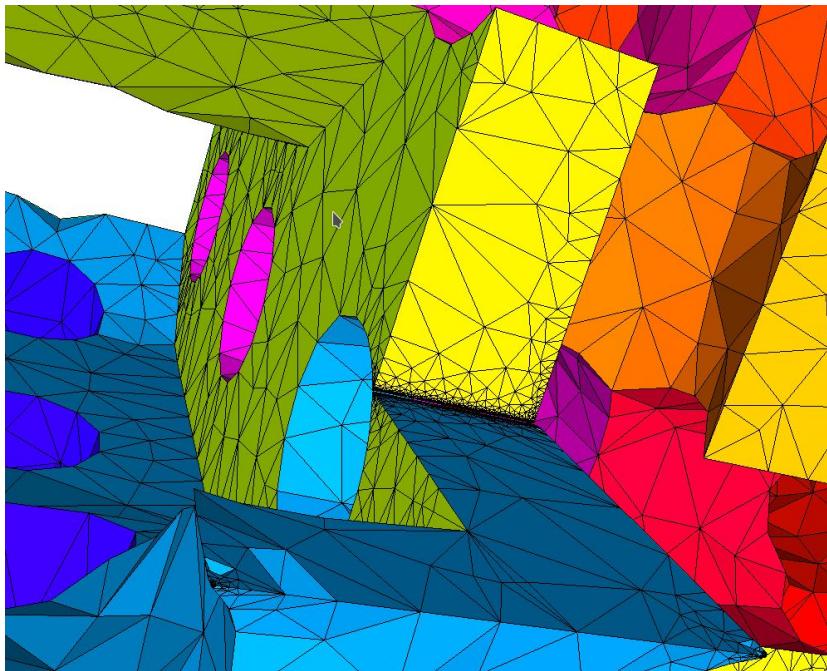
```
typedef CGAL::Labeled_mesh_domain_3<K> Mesh_domain;  
  
typedef CGAL::Mesh_constant_domain_field_3 <  
    Mesh_domain::R, Mesh_domain::Index> Sizing_field;  
  
// Domain  
Mesh_domain domain =  
    Mesh_domain::create_labeled_image_mesh_domain(image);  
  
// Sizing field:  
// set global size to 8, kidney size (label 127) to 3  
double kidney_size = 3.;  
int volume_dimension = 3;  
Sizing_field size(8);  
  
size.set_size(kidney_size, volume_dimension,  
              domain.index_from_subdomain_index(127));  
  
// Mesh criteria  
Mesh_criteria criteria(facet_angle=30, facet_size=6,  
                      facet_distance=2,  
                      cell_radius_edge_ratio=3,  
                      cell_size=size);  
  
// Meshing  
C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
```

# Specifying a Distance Field

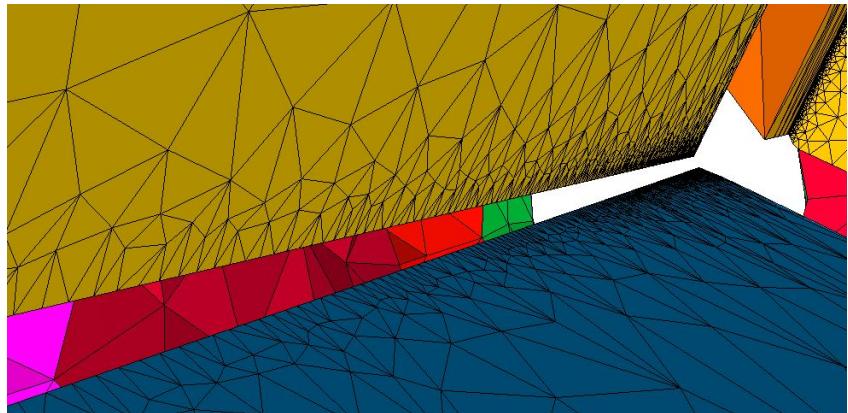
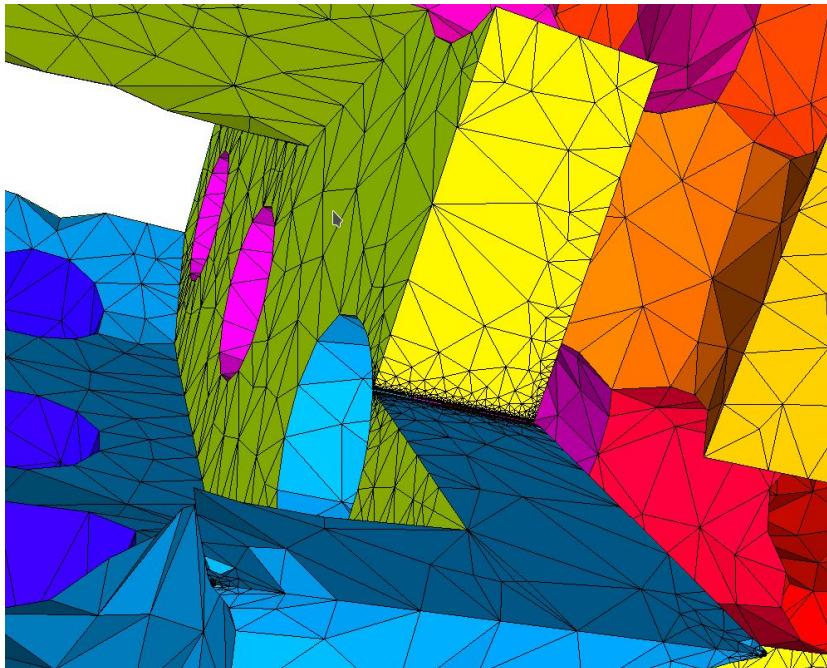


```
typedef CGAL::Labeled_mesh_domain_3<K> Mesh_domain;  
  
typedef CGAL::Mesh_constant_domain_field_3 <  
    Mesh_domain::R, Mesh_domain::Index> Distance_field;  
  
// Domain  
Mesh_domain domain =  
    Mesh_domain::create_labeled_image_mesh_domain(image);  
  
// Sizing field:  
// set global size to 8, kidney size (label 127) to 3  
double kidney_size = 3.;  
int volume_dimension = 3;  
Distance_field variable_distance;  
  
// Mesh criteria  
Mesh_criteria criteria(facet_angle=30,  
                      facet_size=6,  
                      facet_distance=variable_distance,  
                      cell_radius_edge_ratio=3,  
                      cell_size=6);  
  
// Meshing  
C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
```

# Manifold Criterion

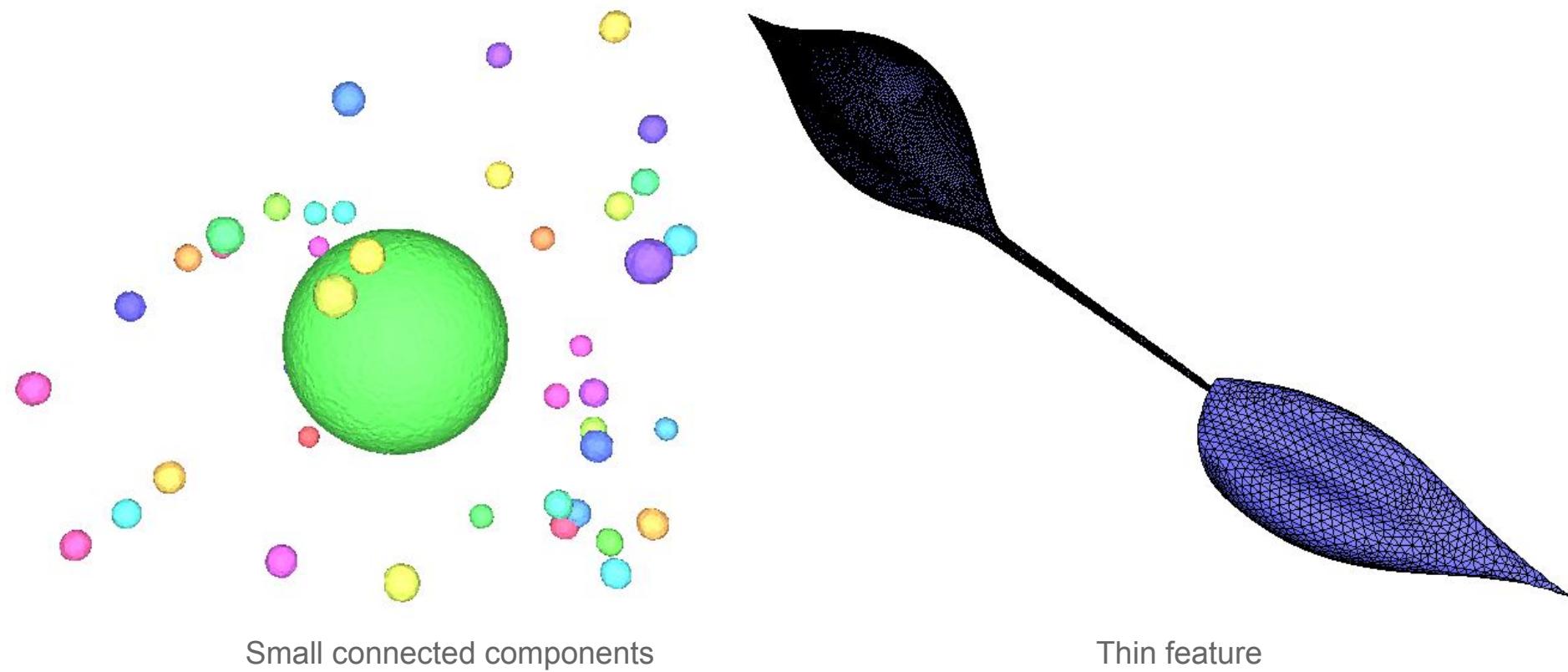


# Manifold Criterion



- Input must be 2-manifold
- Sharp Features must be protected

# About Initial Points

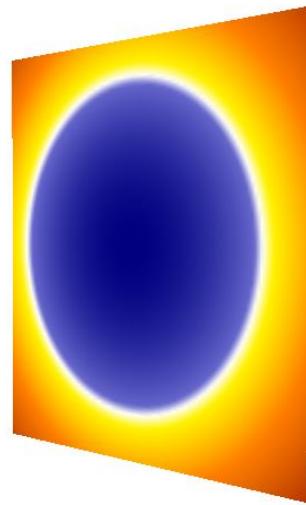
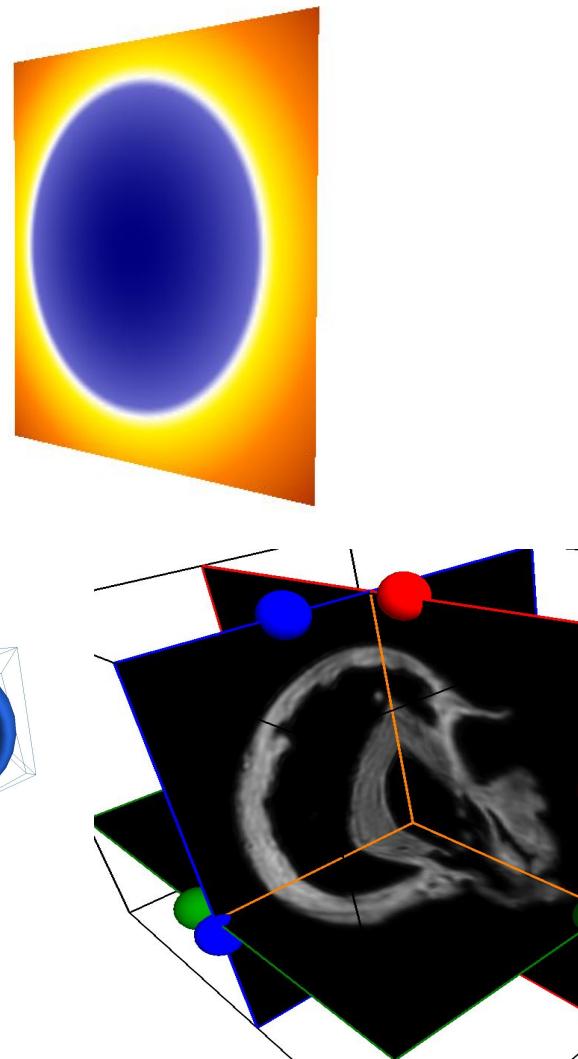
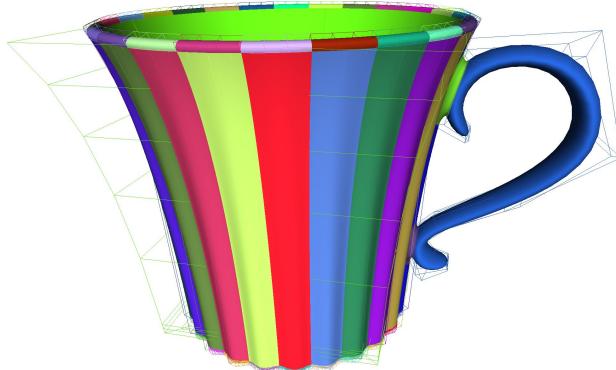
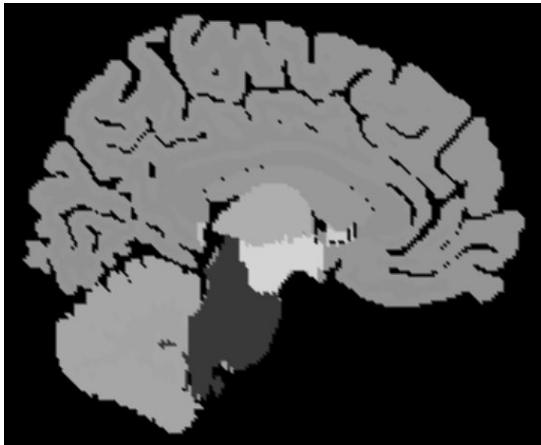


# 3D Mesh Generation

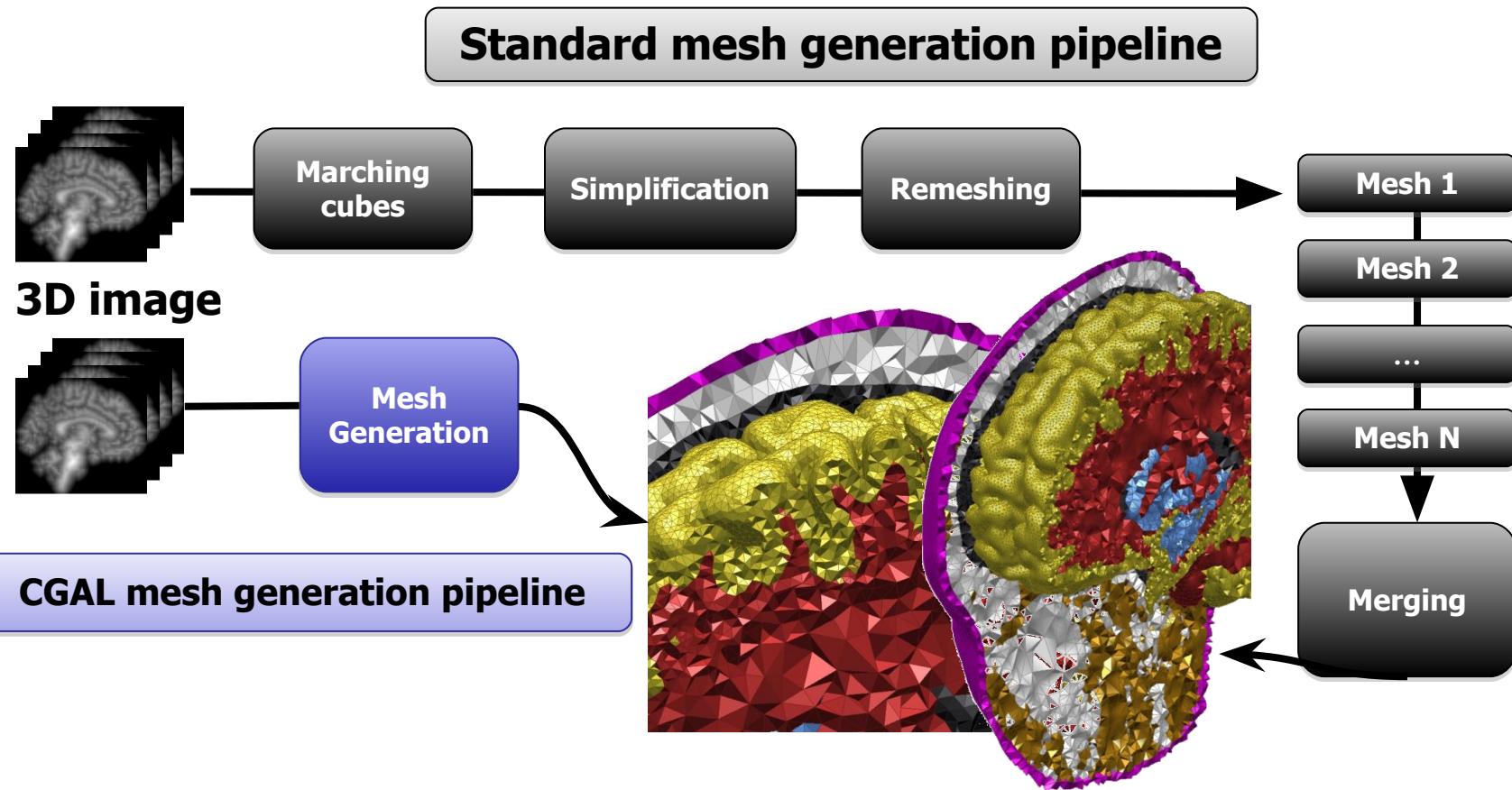
## Multiple types of domains

# Input Domains

- Level set of an implicit function
- Level set grey level image
- Segmented image
- Polyhedral surfaces
- Polyhedral complexes
- NURBS surfaces (in progress)



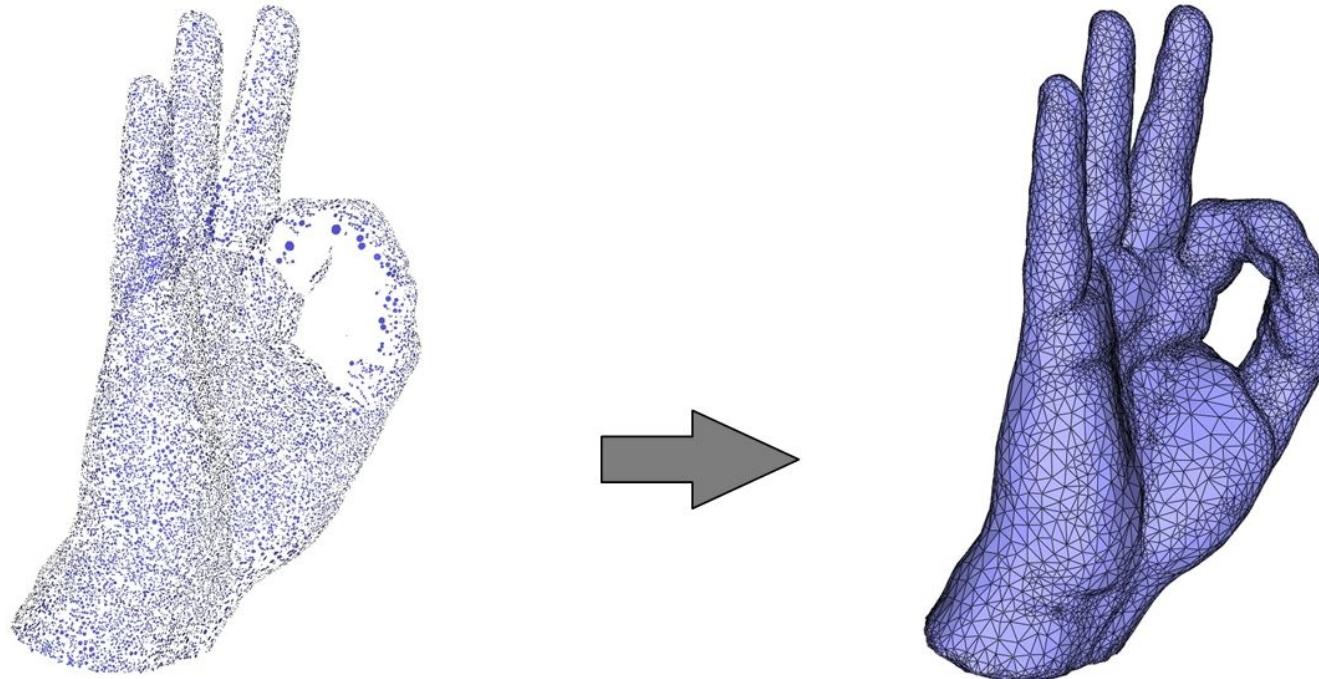
# Added Value: Shortened Pipeline



# Meshing an Implicit function from a Point Set

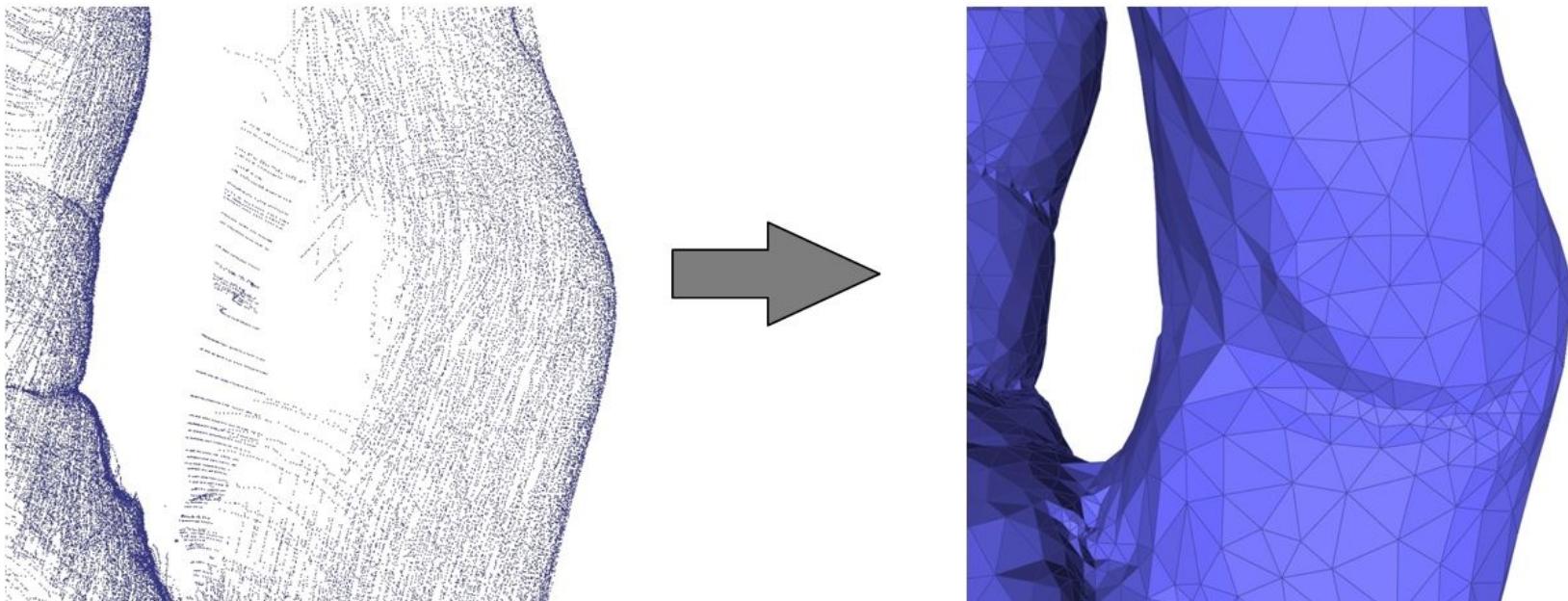
## Poisson Surface Reconstruction

- Construct *Poisson implicit function  $P$*  in ambient space around the input points
- Mesh *the 0-surface of  $P$*



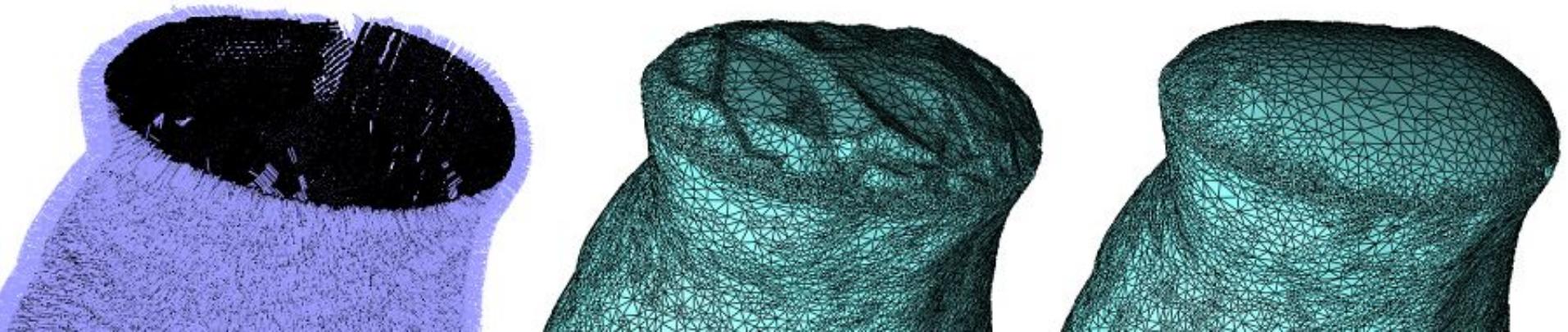
# Poisson Surface Reconstruction

- Works well for uneven distribution of points
- Used for reconstruction of geological bodies



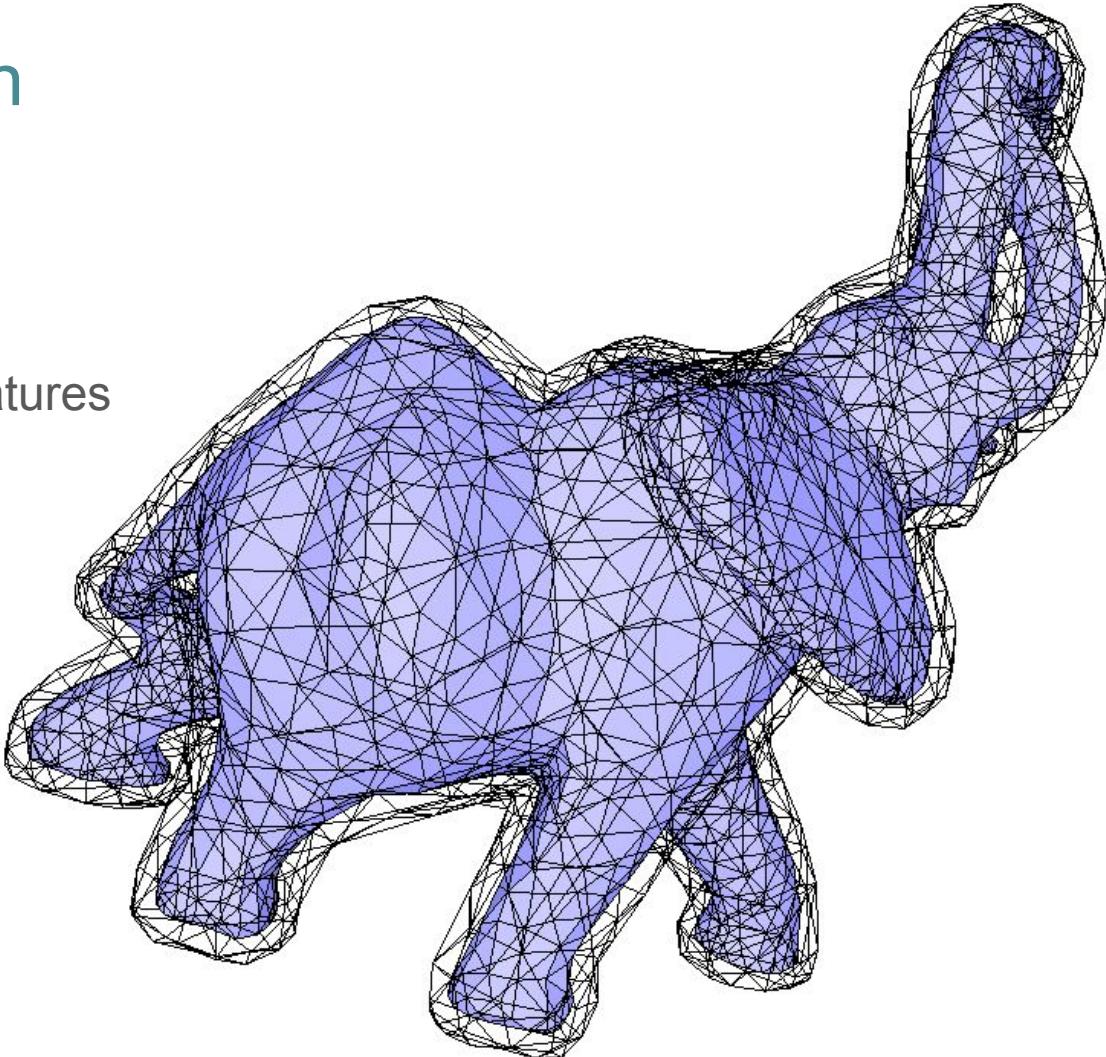
# Poisson Surface Reconstruction

- Algorithm produces watertight surface
- For large holes use two-pass algorithm

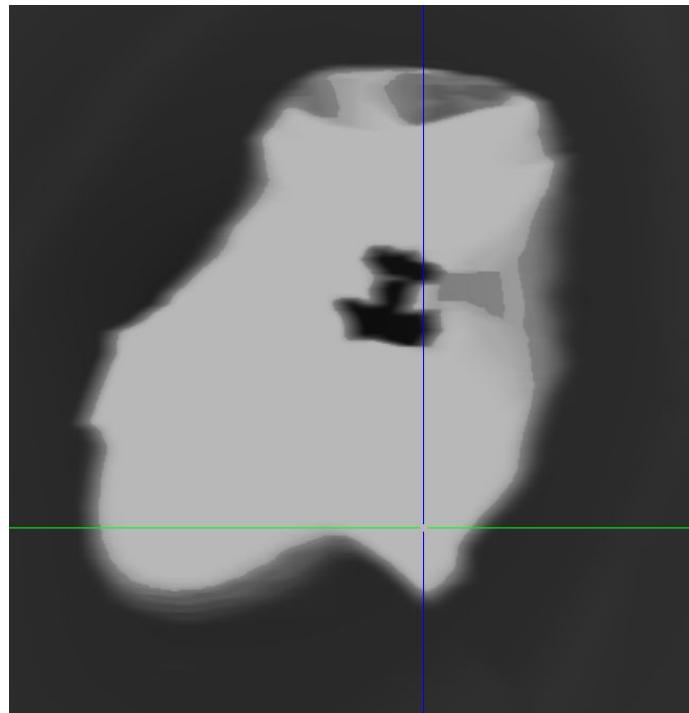


# Offset Mesh Generation

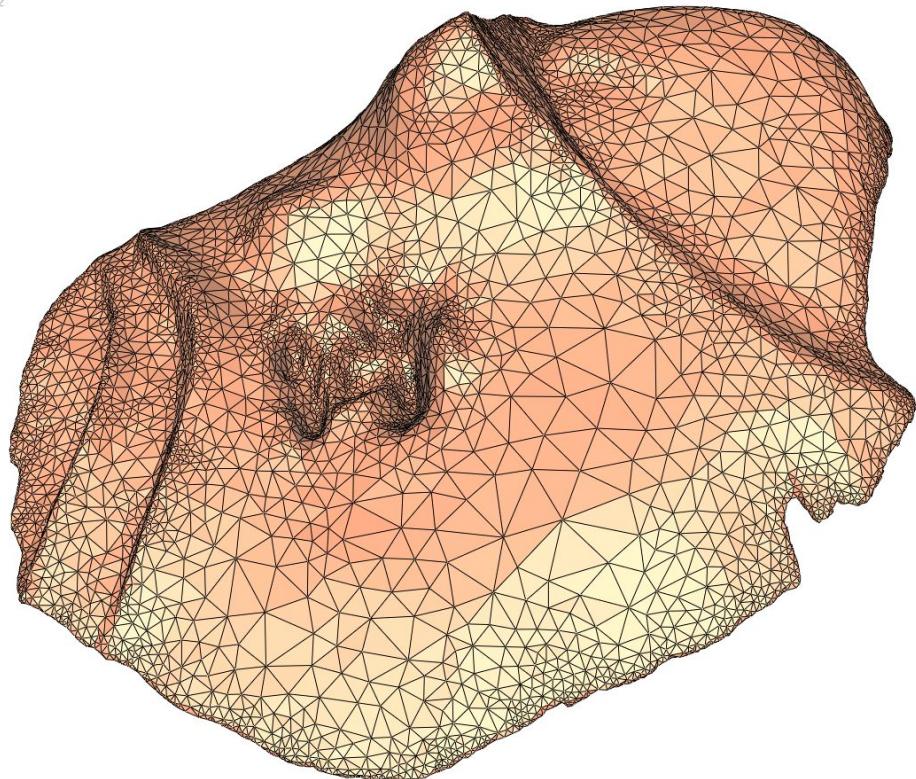
- Meshing an Implicit Function
- Can deal with a variable offset distance field
- No direct handling of sharp features



# Level Set of Grey Level Image



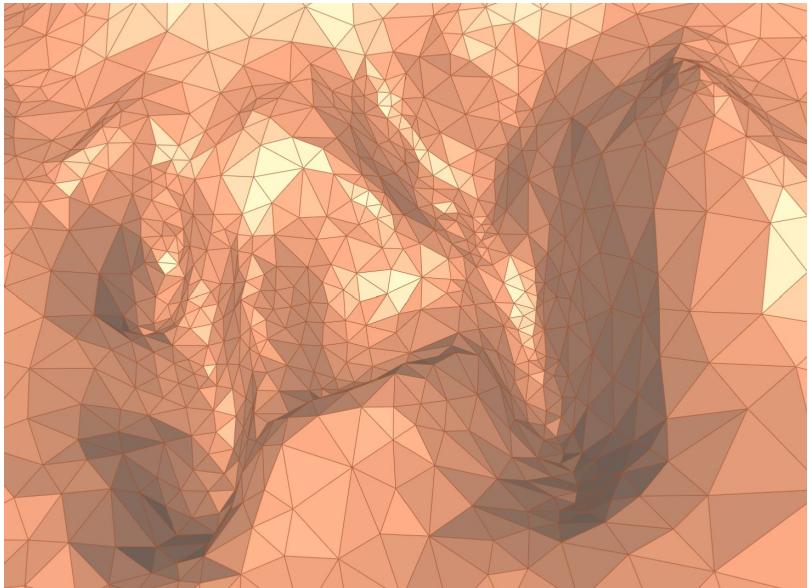
13Hz



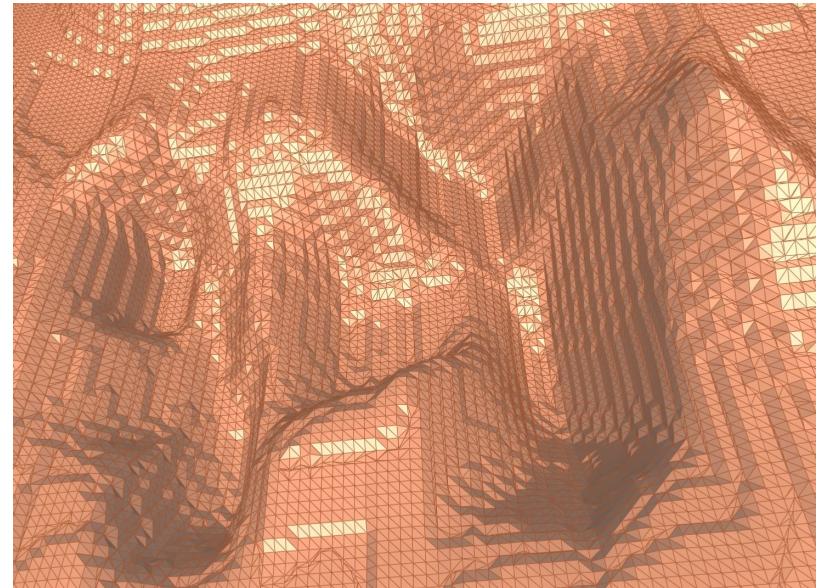
Input: 3D voxel data for SEG Salt Model

# Comparison with Marching Cubes

Delaunay Refinement



Marching Cubes in an octree



# Segmented Image

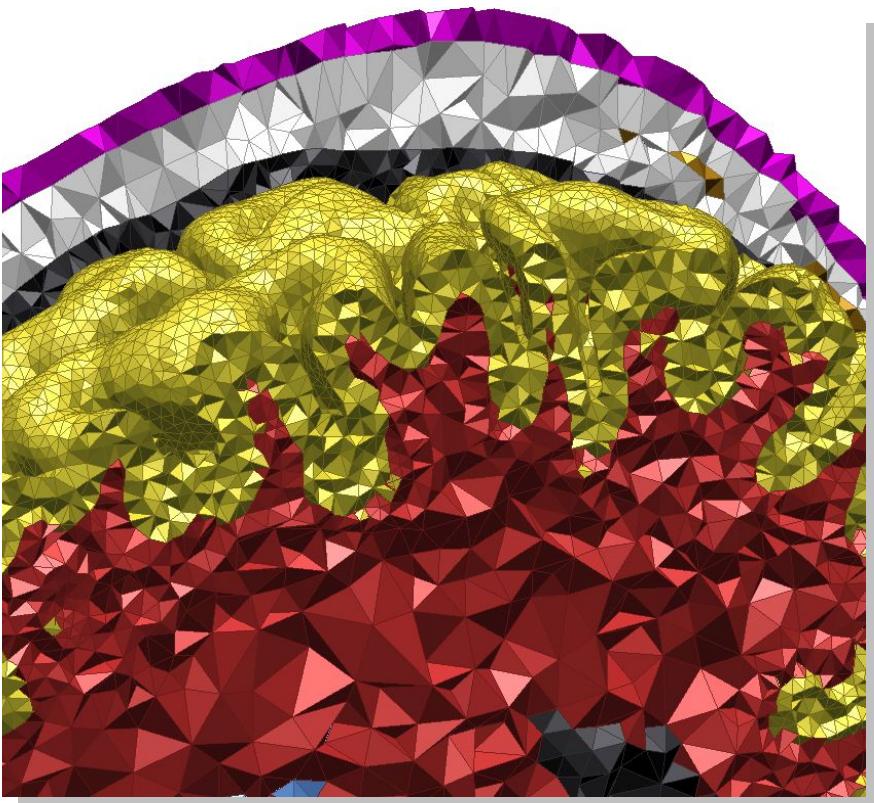
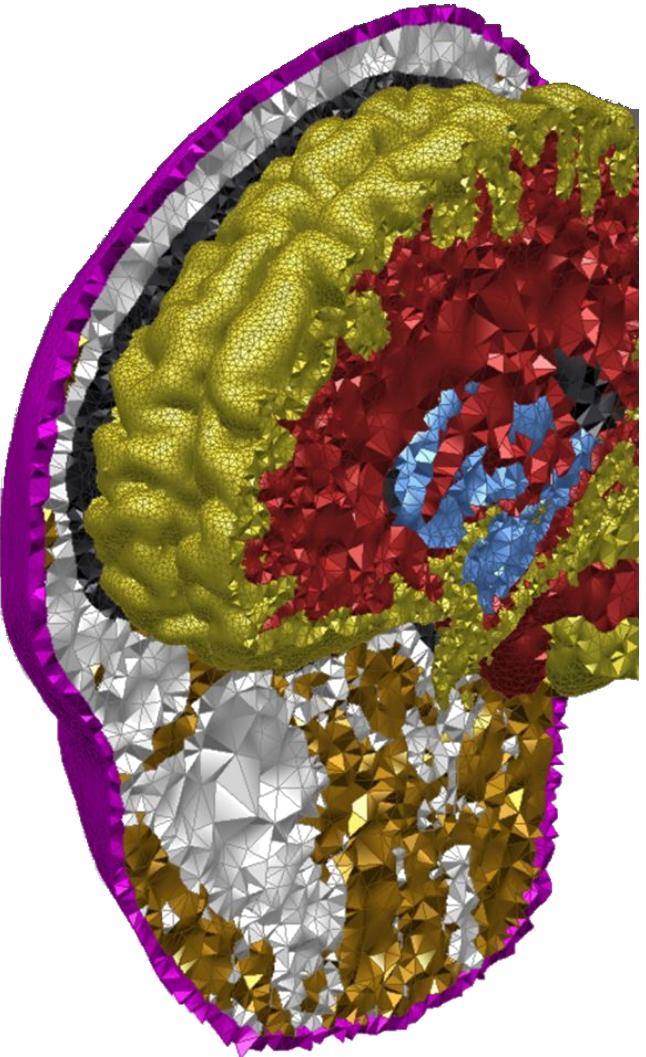
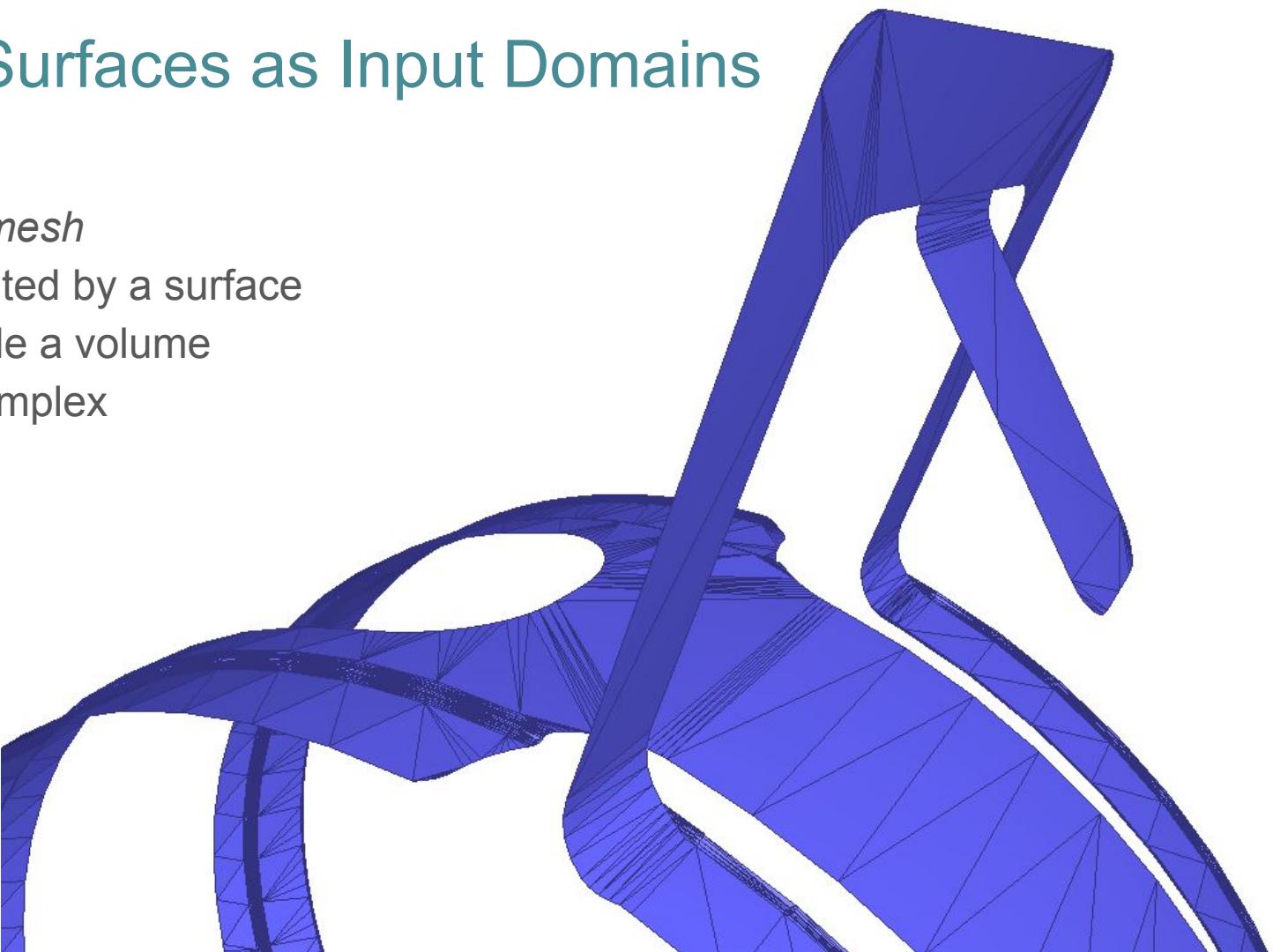


Image from Pons et al.



# Polyhedral Surfaces as Input Domains

- *Surface to remesh*
- Volume delimited by a surface
- Surfaces inside a volume
- Polyhedral complex



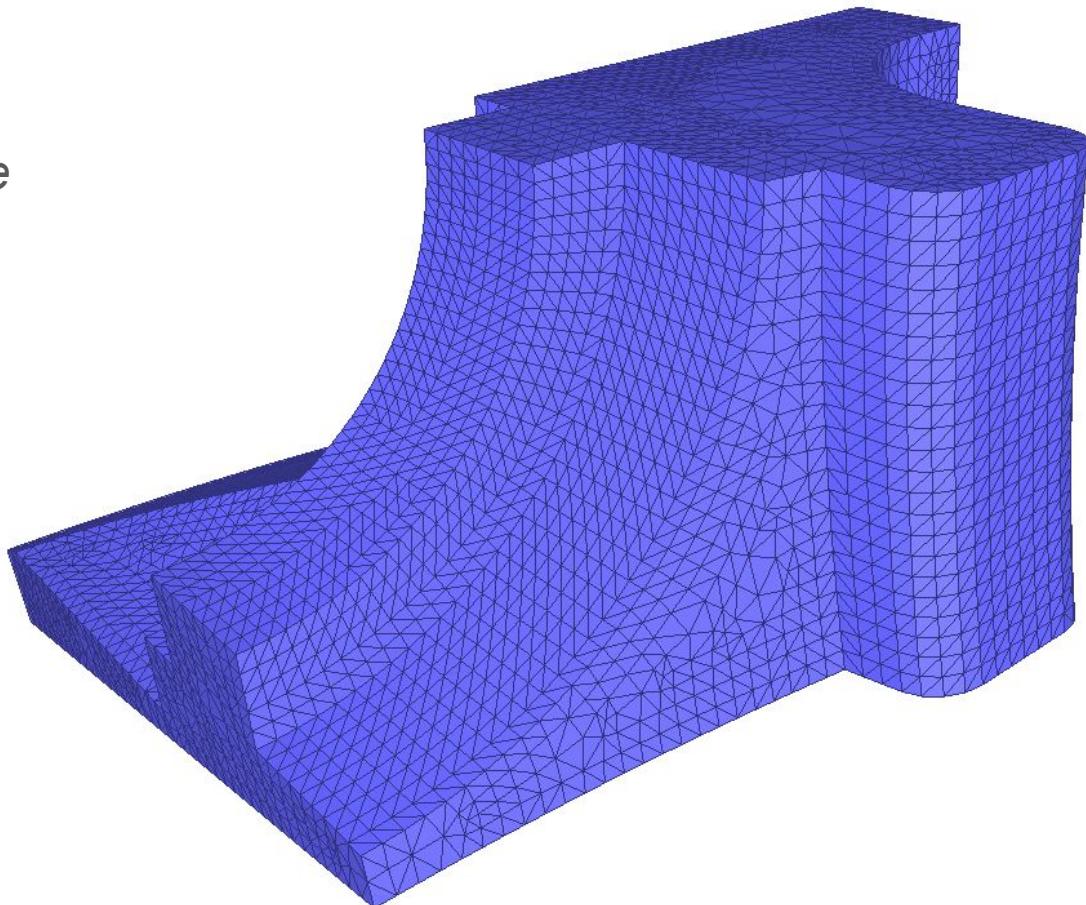
# Polyhedral Surfaces as Input Domains

- *Surface to remesh*
- Volume delimited by a surface
- Surfaces inside a volume
- Polyhedral complex



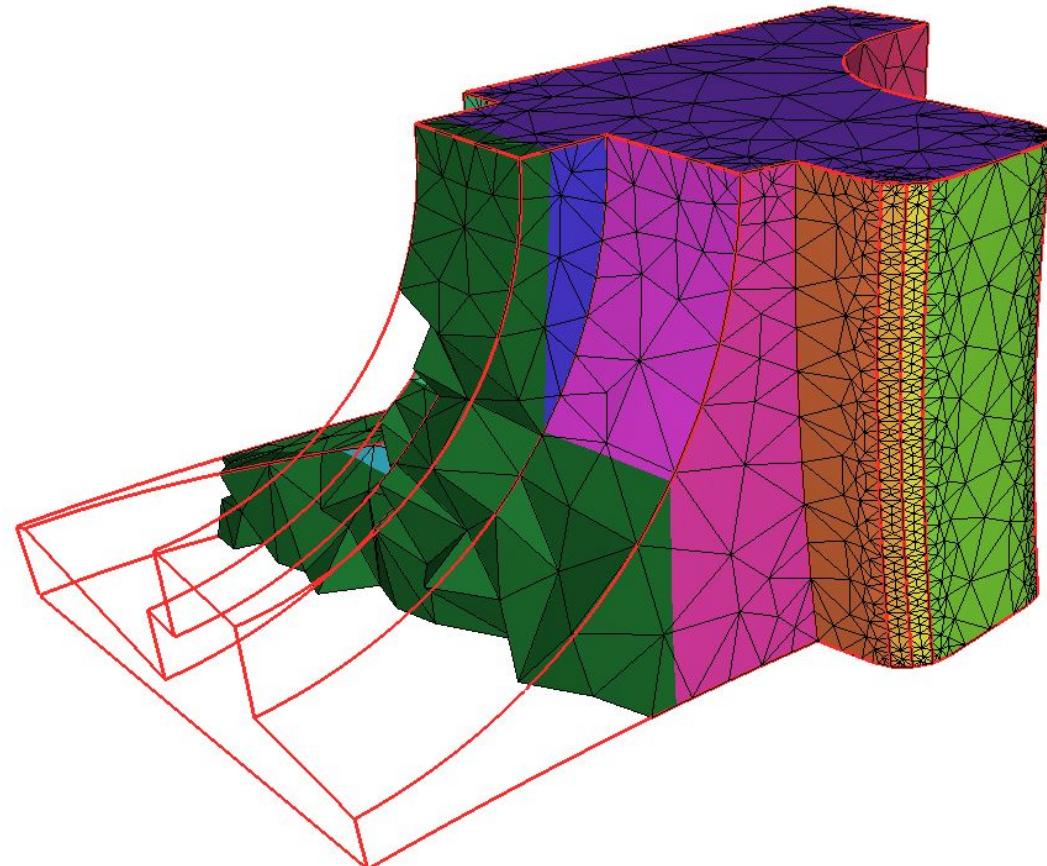
# Polyhedral Surfaces as Input Domains

- Surface to remesh
- *Volume delimited by a surface*
- Surfaces inside a volume
- Polyhedral complex



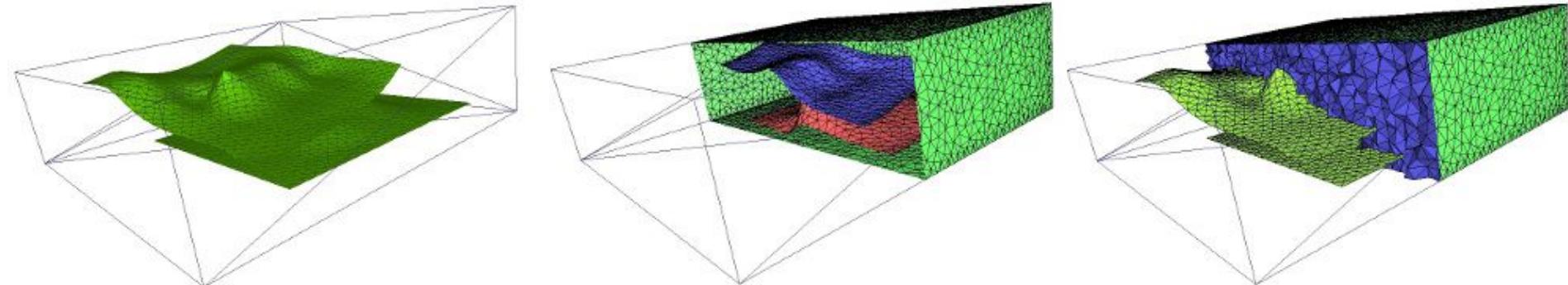
# Polyhedral Surfaces as Input Domains

- Surface to remesh
- *Volume delimited by a surface*
- Surfaces inside a volume
- Polyhedral complex



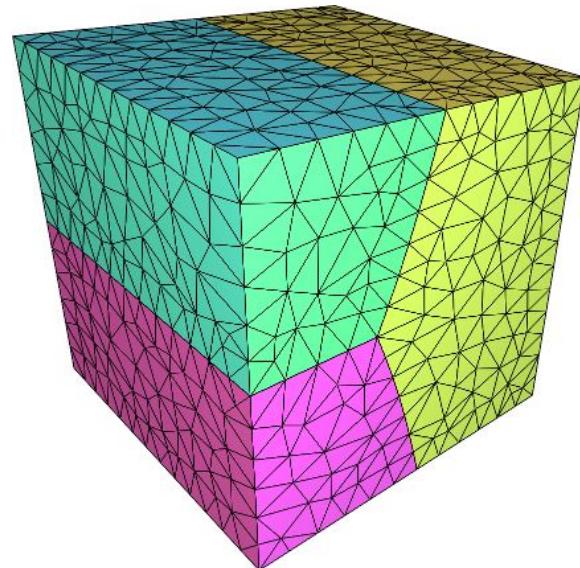
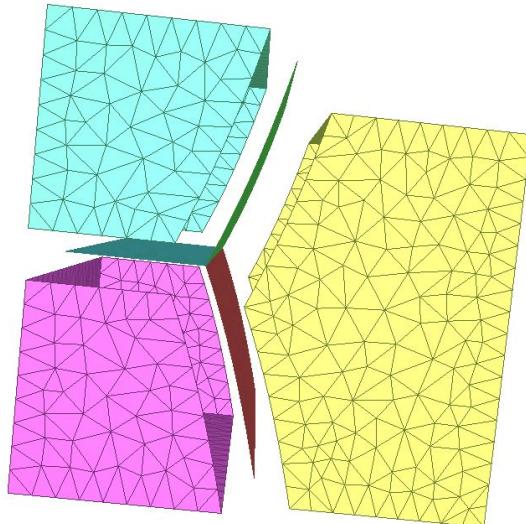
# Polyhedral Surfaces as Input Domains

- Surface to remesh
- *Volume delimited by a surface*
- *Surfaces inside a volume*
- Polyhedral complex

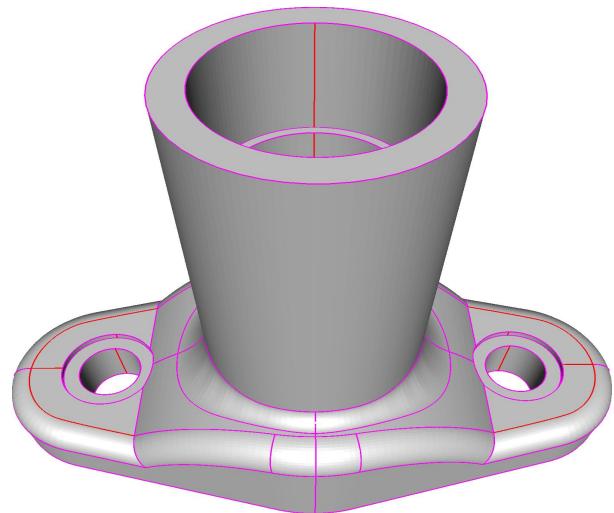


# Polyhedral Surfaces as Input Domains

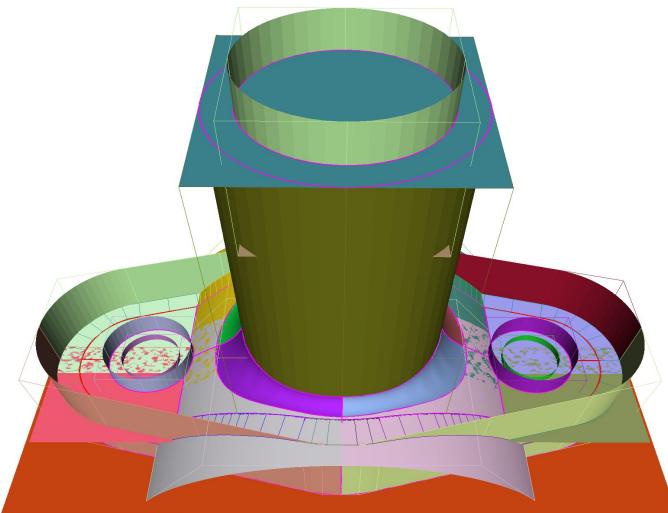
- Surface to remesh
- Volume delimited by a surface
- Surfaces inside a volume
- *Polyhedral complex*



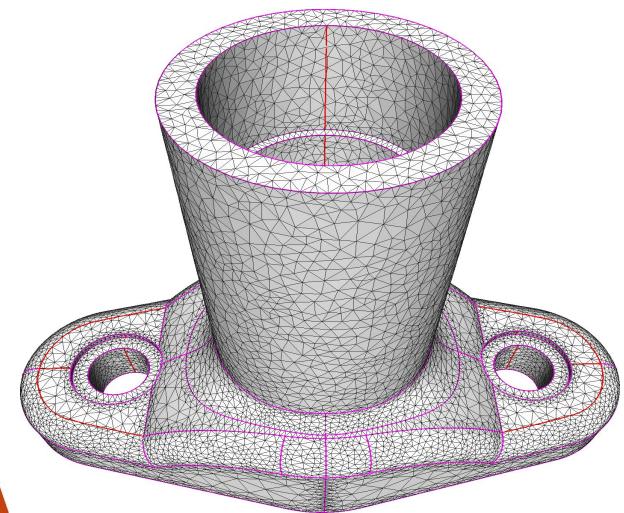
# NURBS surfaces from CAD (in progress)



Input NURBS model



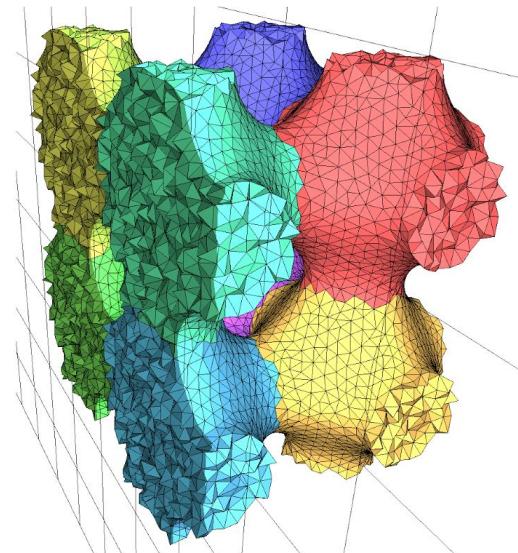
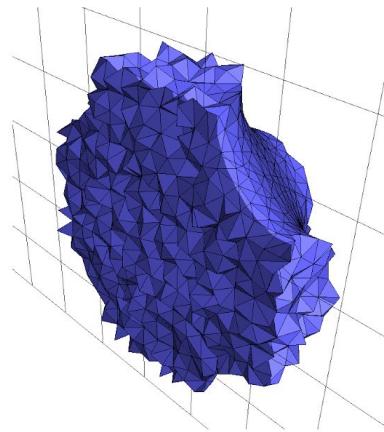
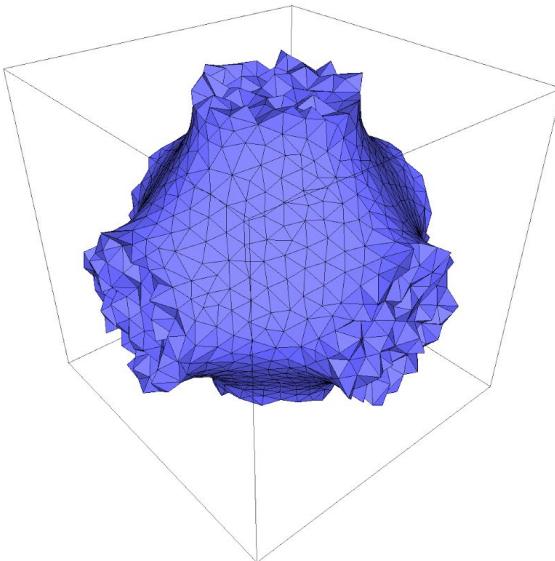
41 NURBS patches



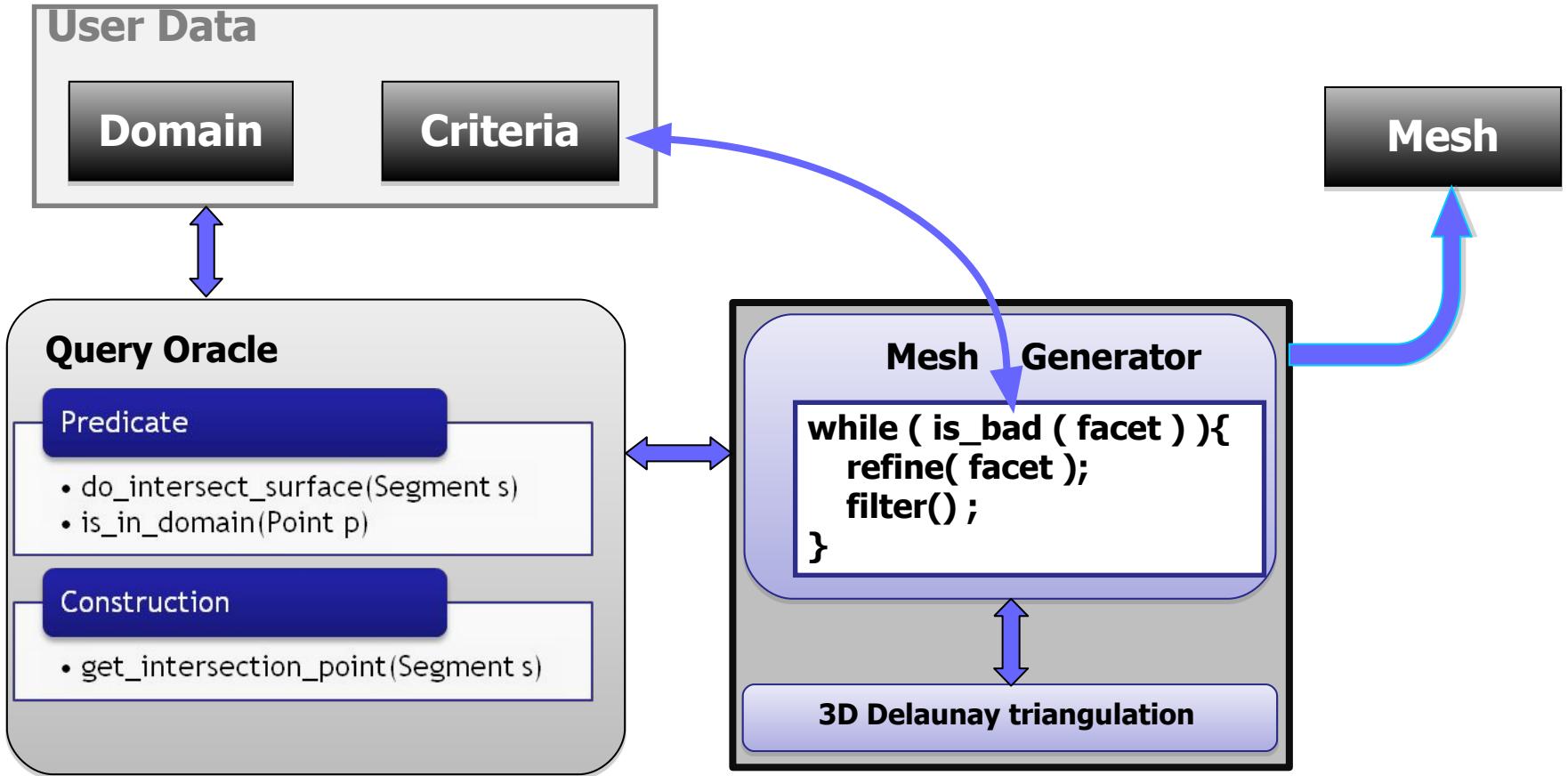
Feature preserving mesh

# Periodic Mesh Generation

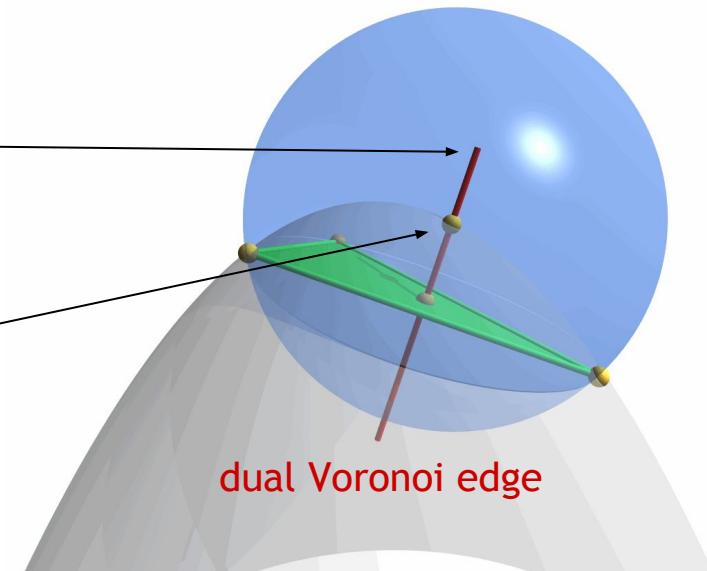
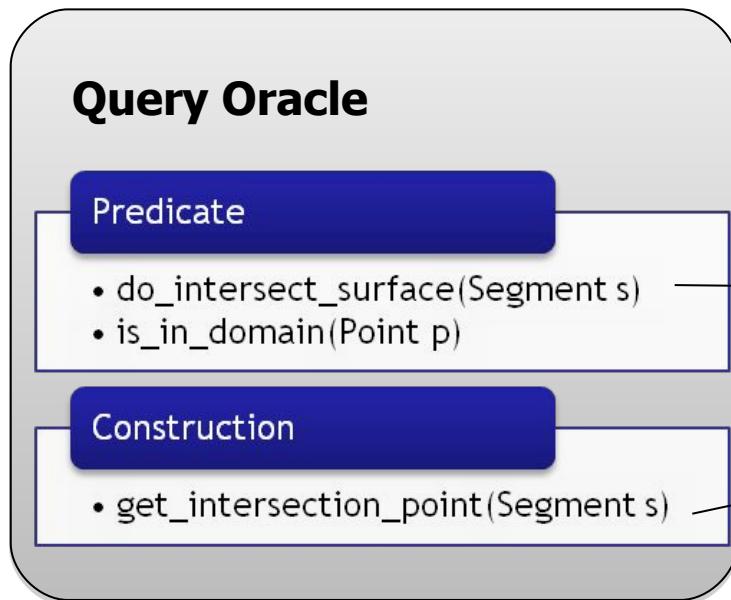
- Efficiency: no duplication of points
- No constraint at the border of the domain
- Same options as the 3D Mesh Generation package  
(protection of features, optimization, etc.)



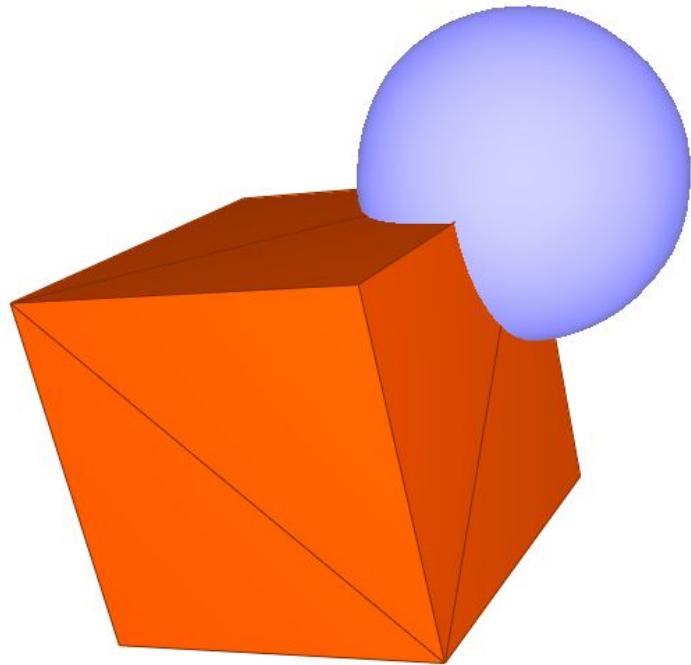
# Defining Yet Another Domain



# Oracle for a Domain



# Develop a Hybrid Domain



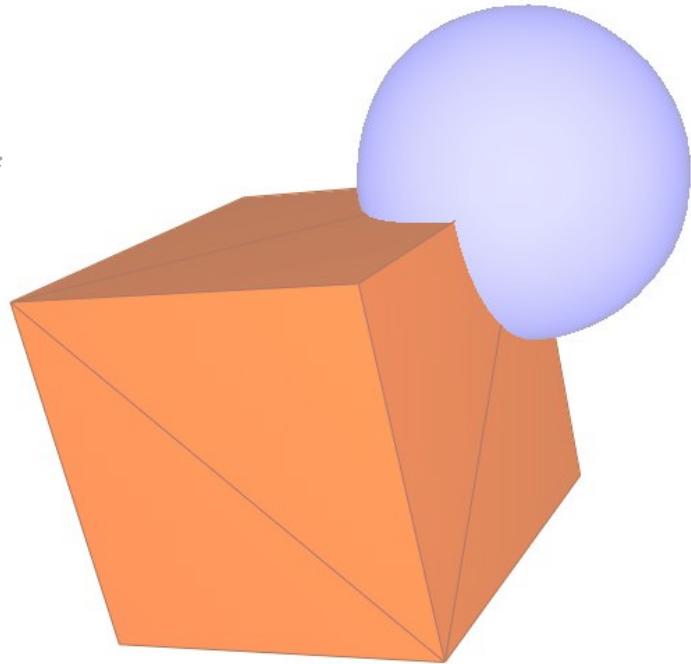
# Develop a Hybrid Domain

```
double sphere_function(const Point& p)
{
    return CGAL::squared_distance(p, CGAL::ORIGIN) - 1.0;
}

Implicit_domain domain(sphere_function);

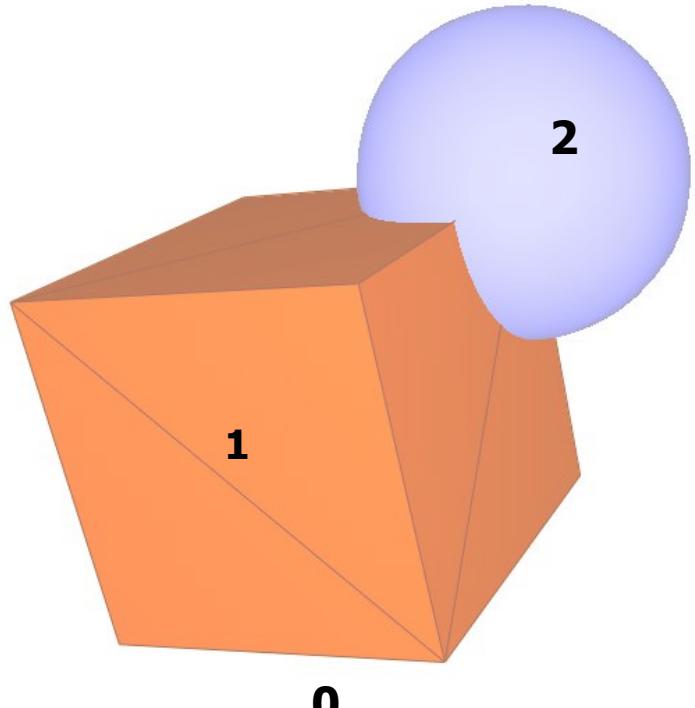
Mesh_criteria criteria( facet_size = 0.1,
                       facet_distance = 0.025 );

C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
```



# Develop a Hybrid Domain

```
class Hybrid_domain {  
  
    Implicit_domain & implicit;  
    Polyhedral_domain & polyhedron;  
  
public:  
    Hybrid_domain(Implicit_domain & implicit,  
                  Polyhedral_domain & polyhedron);  
  
    int is_in_domain(Point_3 p)  
    {  
        if(implicit.is_in_domain(p))  
            return 2;  
        else  
            if(polyhedron.is_in_domain(p))  
                return 1;  
            else  
                return 0;  
    }  
};
```



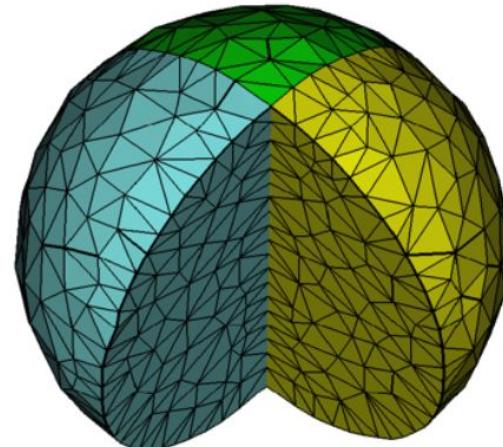
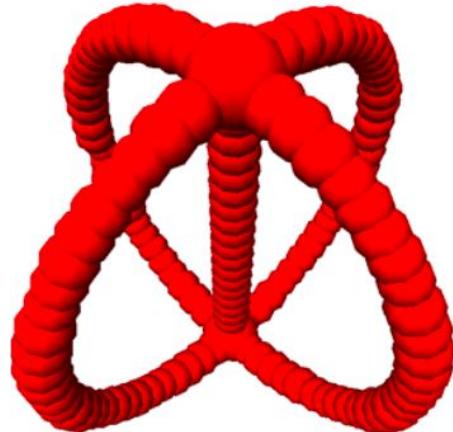
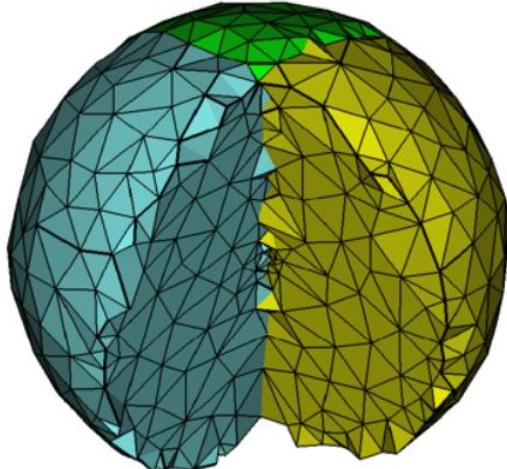
# 3D Mesh Generation Feature Preserving

# Sharp Features

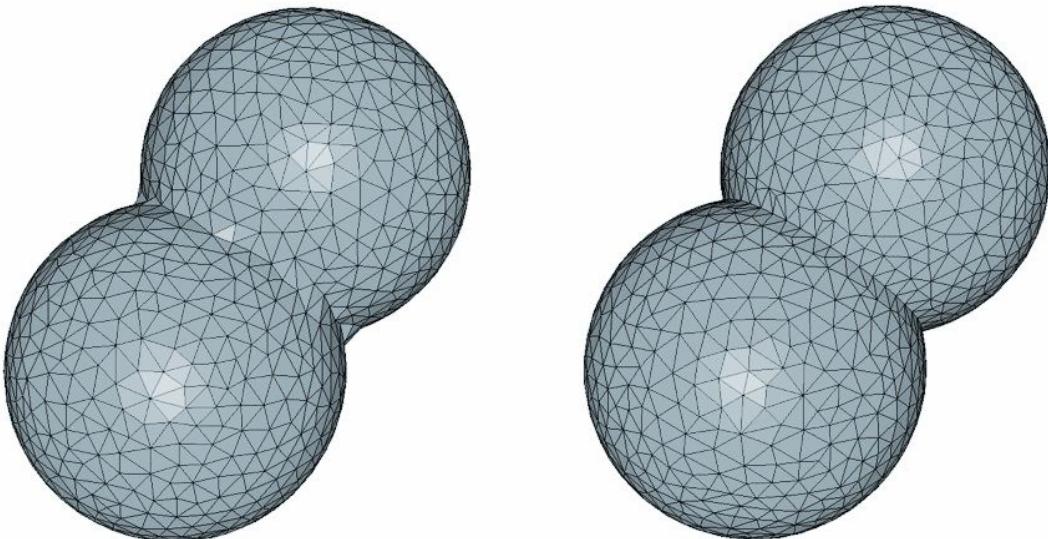
Protecting balls method [Cheng *et al.* 2007]

- Cover the edges with balls
- Run the weighted version of Delaunay refinement

Segments joining centers of consecutive protecting balls  
are guaranteed to be edges in the mesh

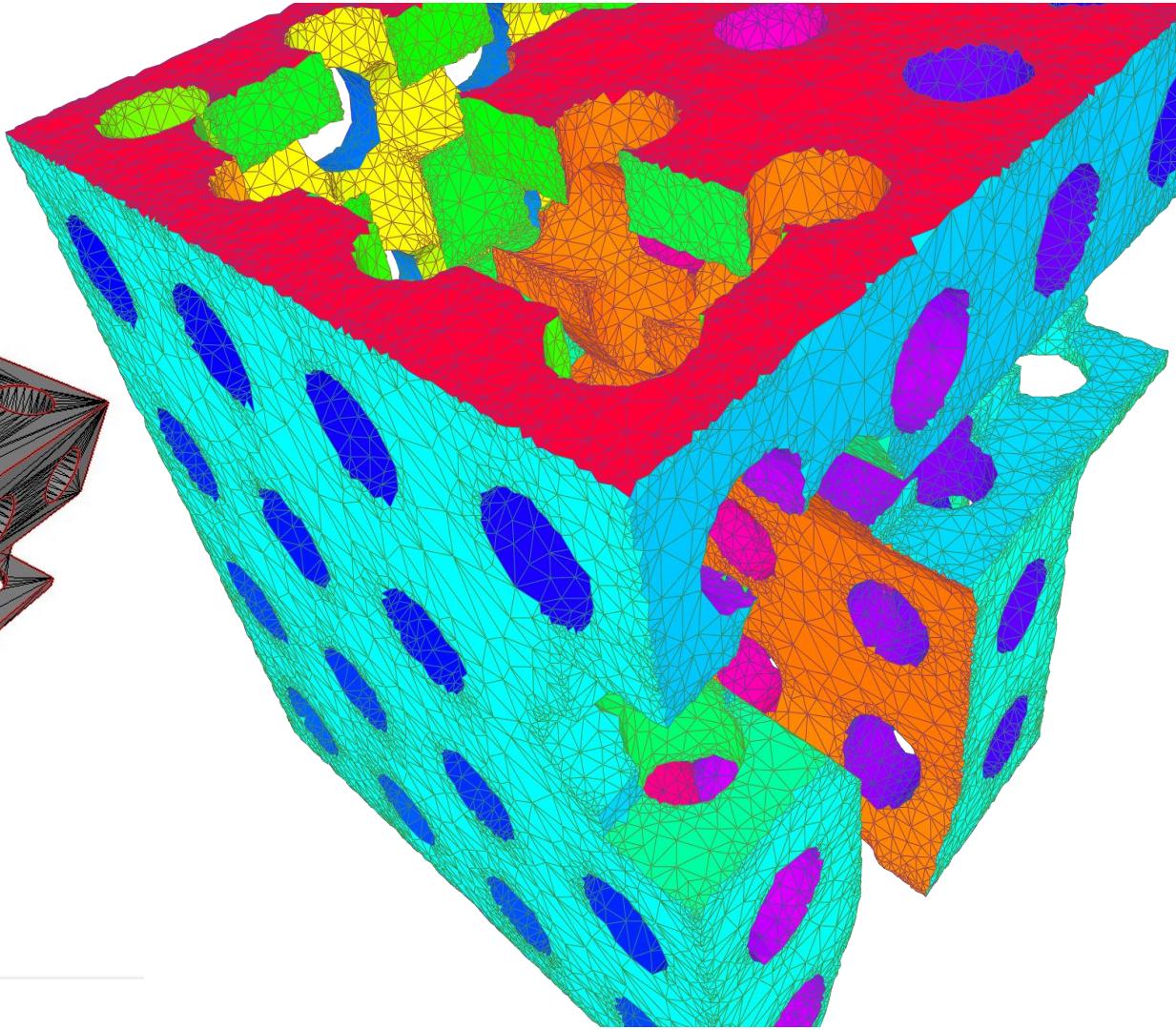
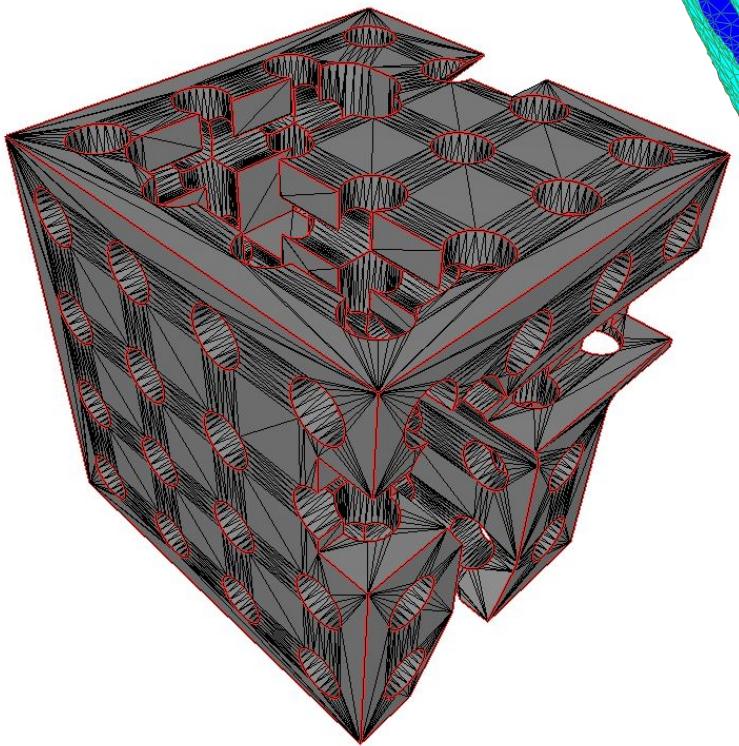


# Meshing an Implicit Function with 1D Features

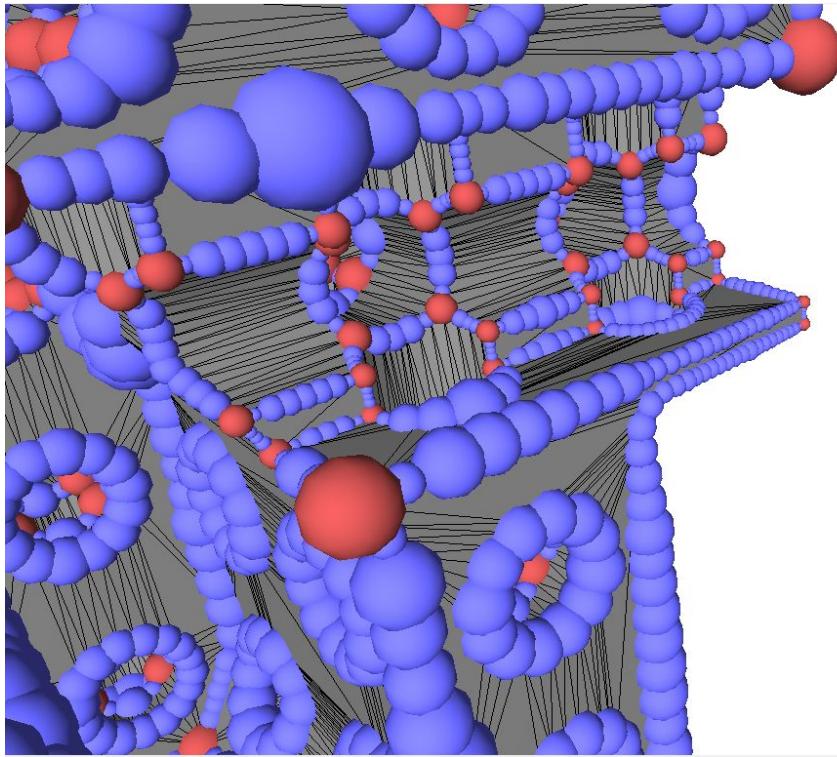
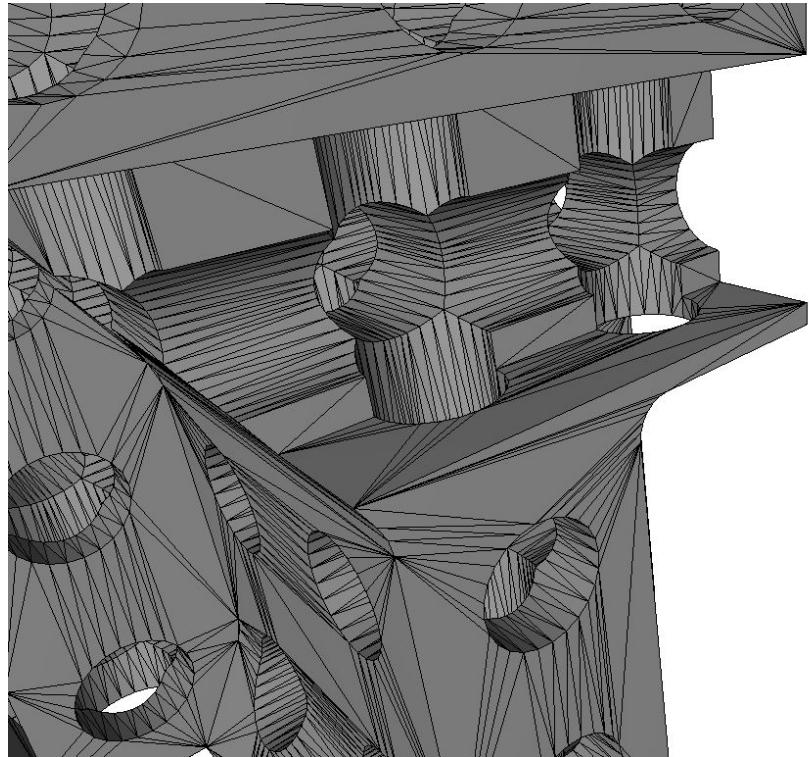


```
Polyline_3 polyline =  
    polylines.front();  
  
for (int i = 0; i < 360; ++i)  
{  
    Point p (1,  
              std::cos(i*CGAL_PI/180),  
              std::sin(i*CGAL_PI/180));  
    polyline.push_back(p);  
}  
polyline.push_back(polyline.front());  
  
// Insert edge in domain  
domain.add_features (polylines.begin(),  
                     polylines.end());
```

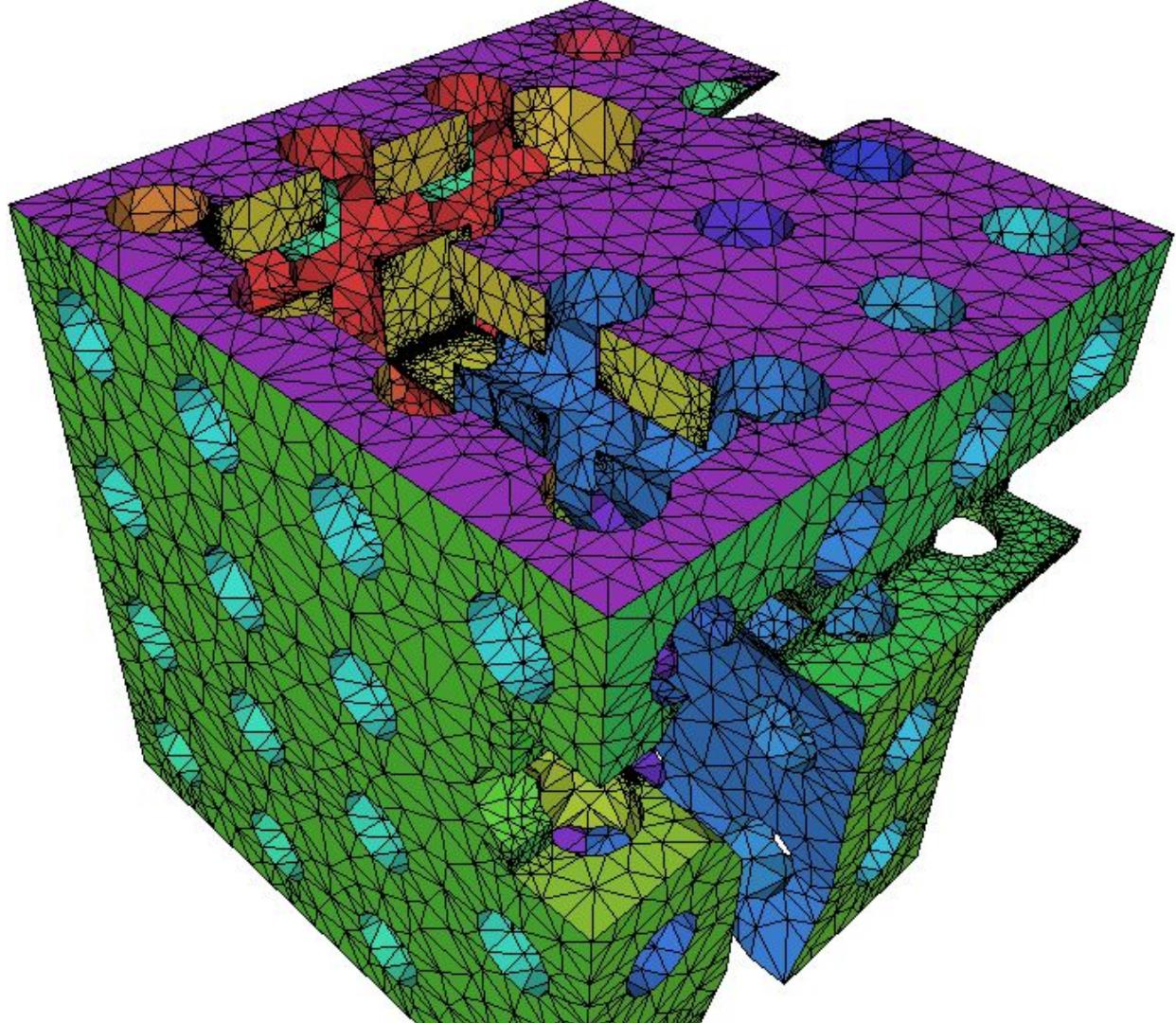
# Sharp Features



# Sharp Features - Adaptive Protecting Balls



# Sharp Features

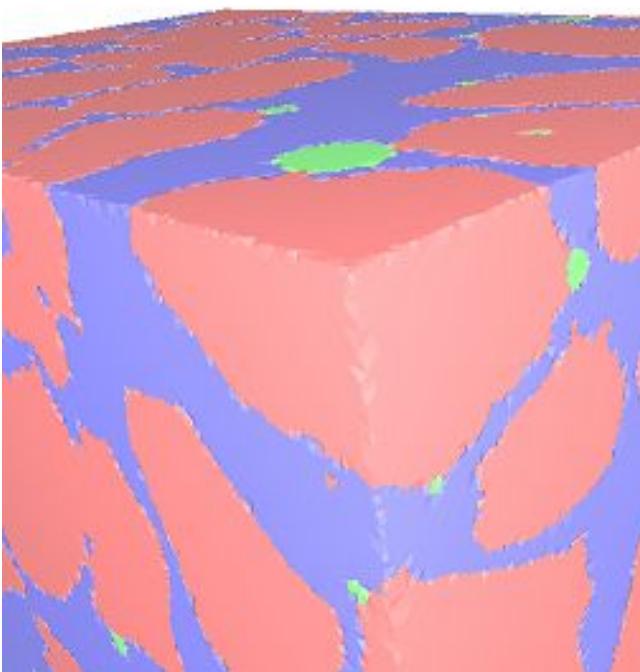


# 3D Mesh Generation

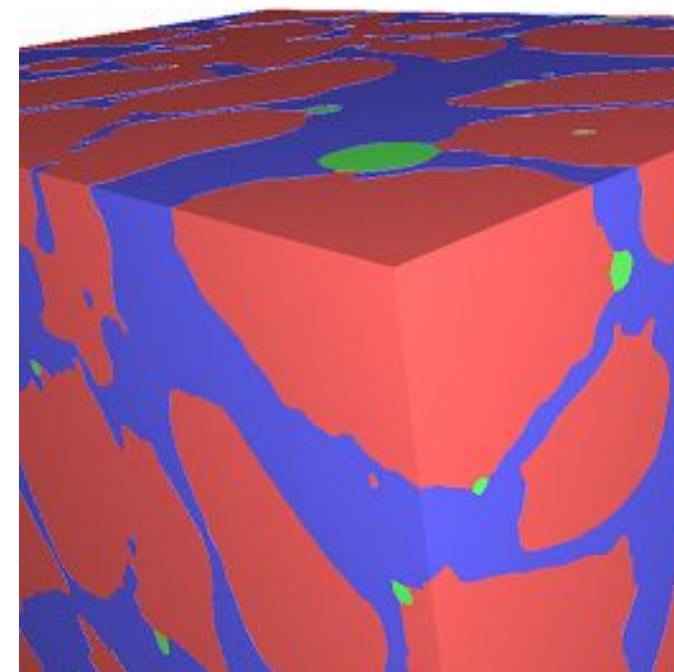
## Meshering from Images

# 1D Features in Images

- Automatic detection and protection of intersections with the bounding box



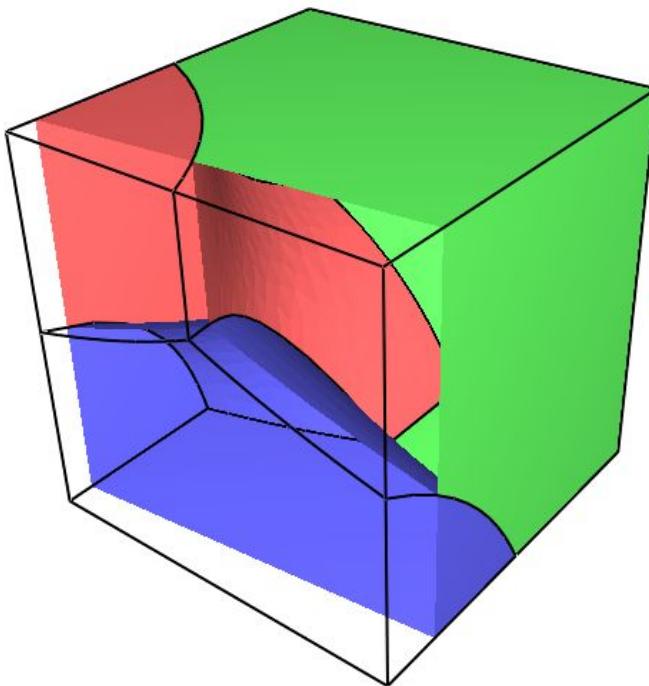
Without features protection



With features protection

# 1D Features in Images

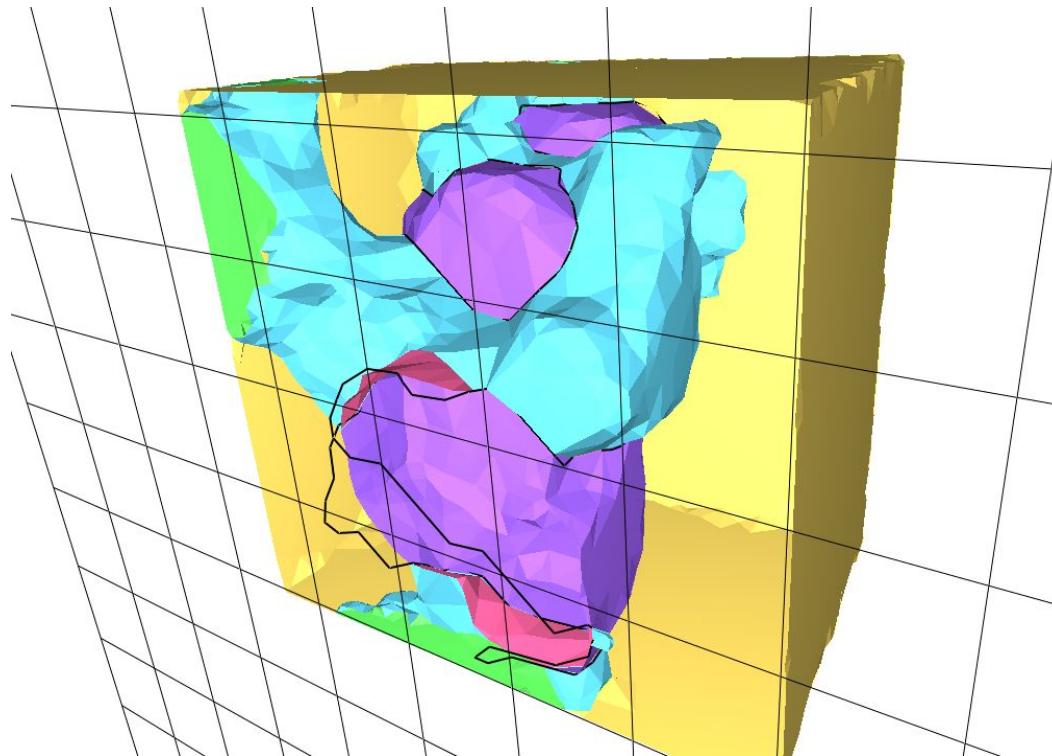
- Detection of triple lines



[Hege, Hans-Christian, et al., 1997]

# 1D Features in Images

- Detection and protection of internal features before meshing



[Hege, Hans-Christian, et al., 1997]

# Deal with Image in memory

## Examples in CGAL

- [Mesh\\_3/random\\_labeled\\_image.h](#)
- [CGALImageIO/extract\\_a\\_sub\\_image.cpp](#)

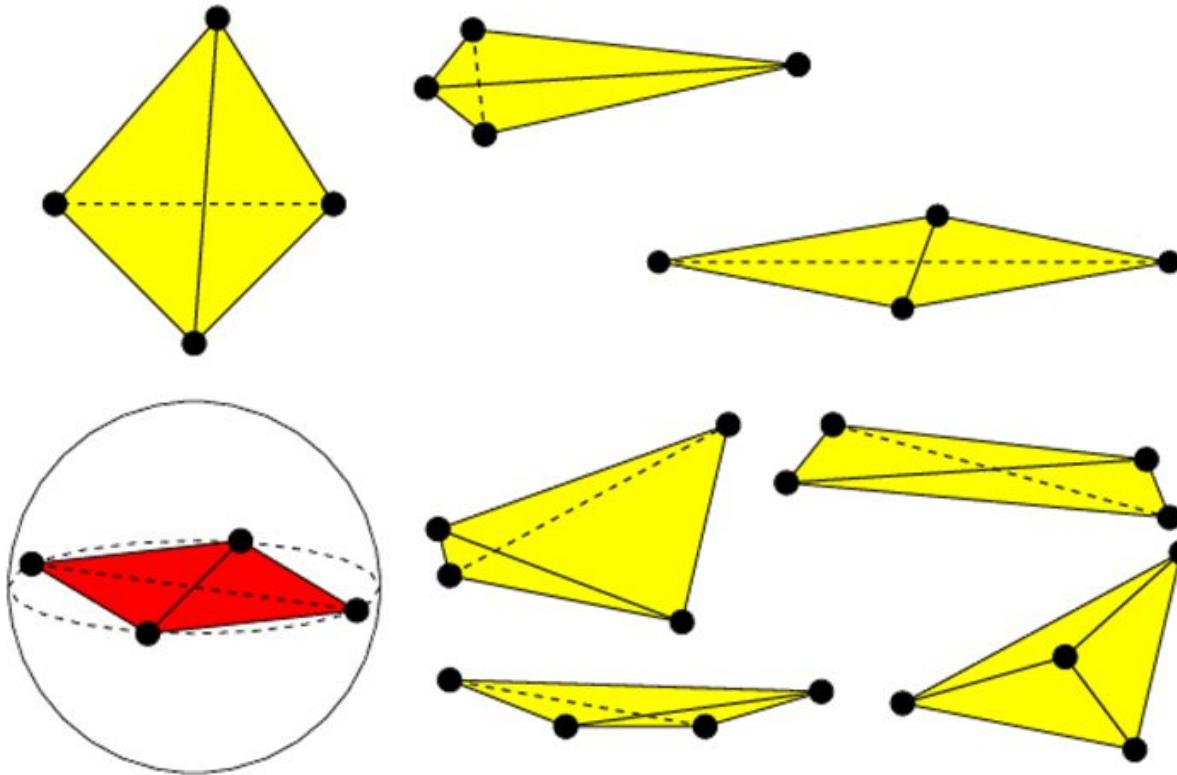
## Non-documented internal API

```
#include <CGAL/ImageIO.h>
auto *new_image = ::_createImage(new_xdim, new_ydim, new_zdim, 1,
    image->vx, image->vy, image->vz, image->wdim,
    image->wordKind, image->sign);
auto* new_data = static_cast<char*>(new_image->data);
...
auto r = ::_writeImage(new_image, argv[2]);
::_freeImage(new_image);
```

# 3D Mesh Generation

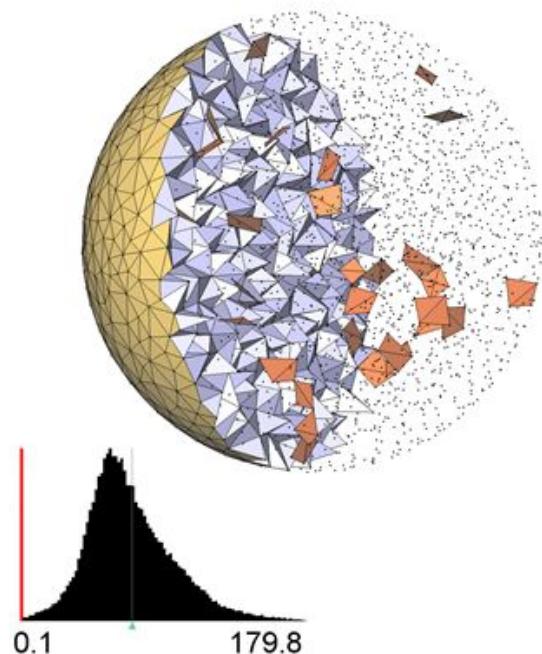
## Mesh Optimization

# Delaunay Refinement may produce Slivers

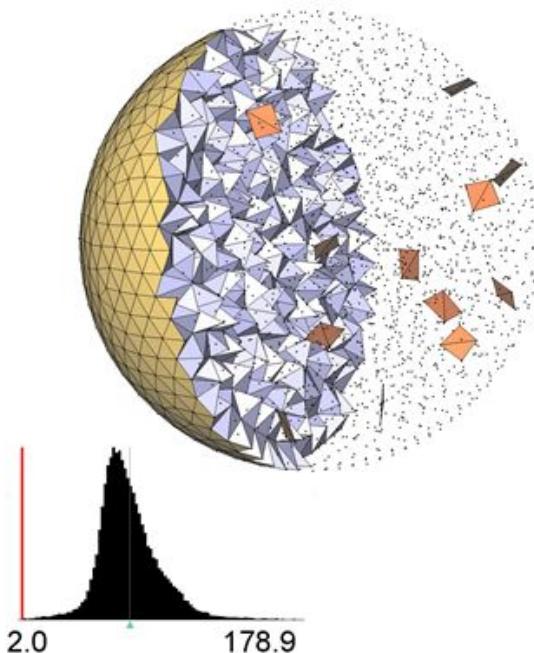


# Mesh Quality Optimization - Global

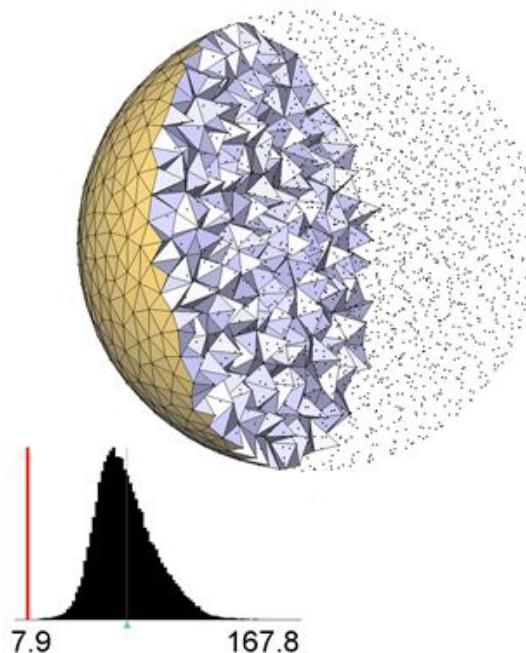
Increase minimal dihedral angle through post-processing



Refinement



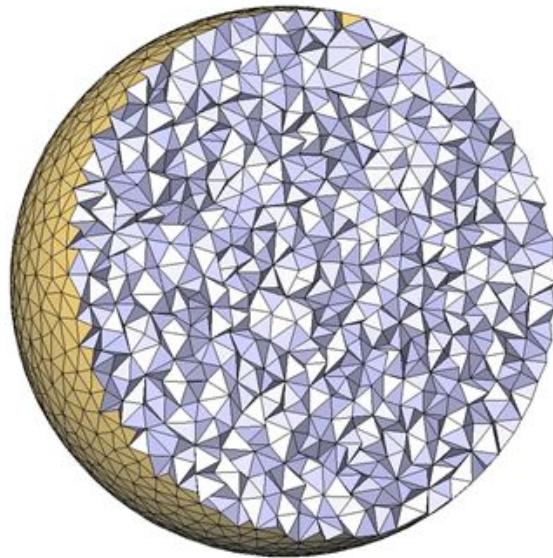
Lloyd



ODT

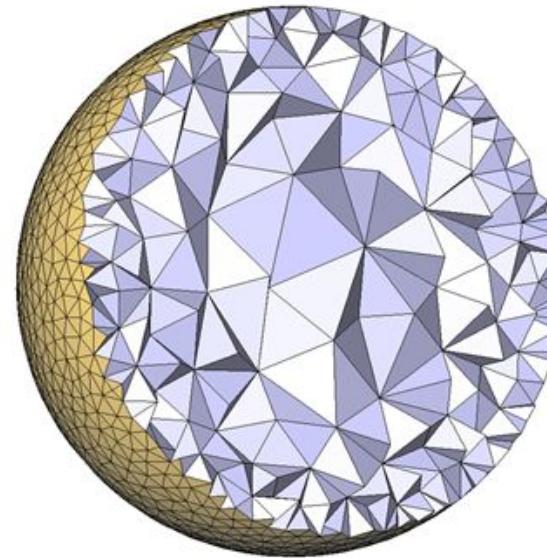
# Mesh Quality Optimization - Local

Sliver Perturbation moves vertices to locally improve dihedral angles



[ $30^\circ$  ;  $138^\circ$  ]

Uniform Sizing Field

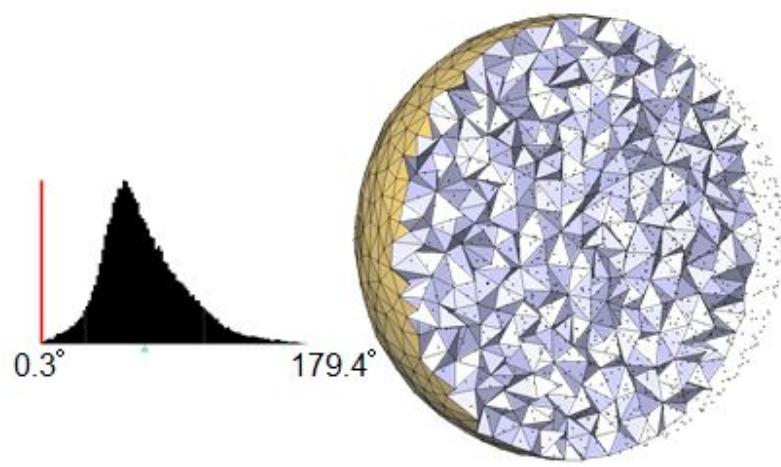


[ $23^\circ$  ;  $142^\circ$  ]

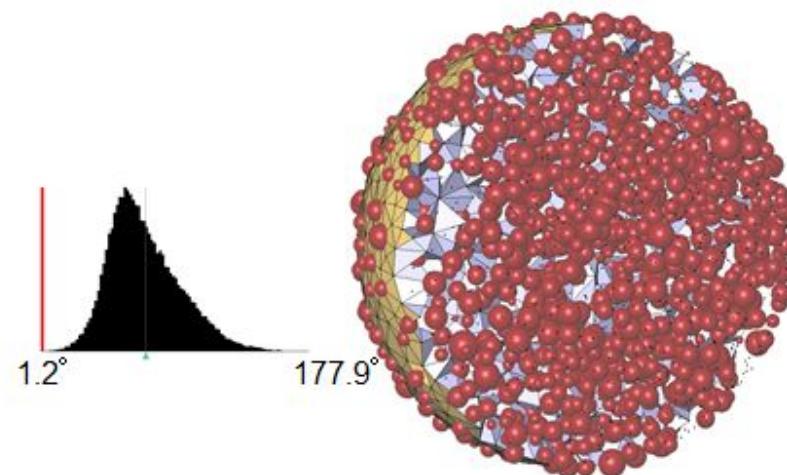
Lipschitz Sizing Field

# Mesh Quality Optimization - Local

Sliver Exudation adds weights to change connectivity

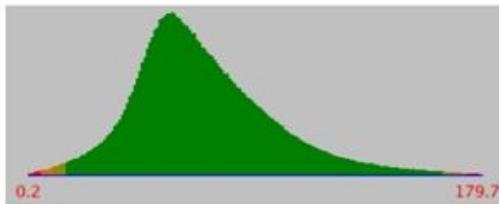


Refinement

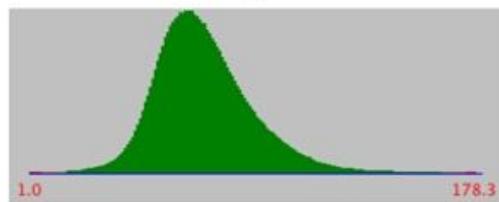


Exudation

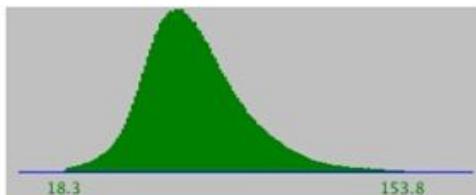
# Mesh Quality Optimization - Combined



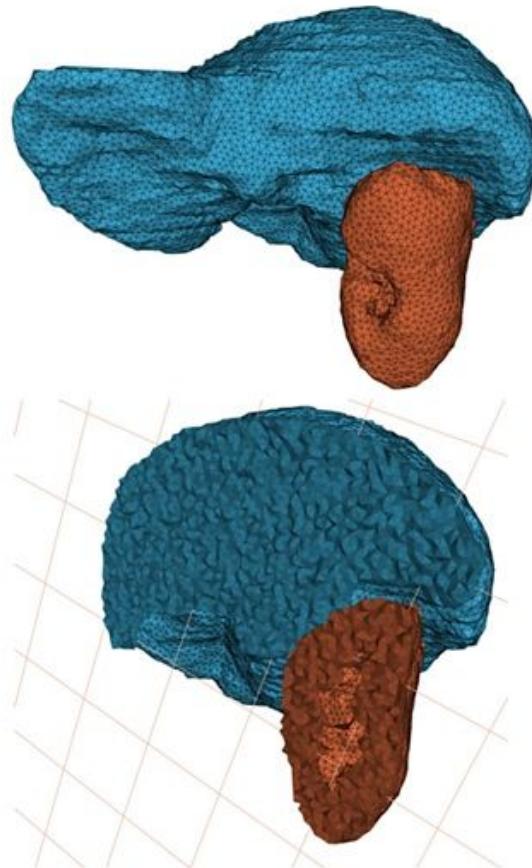
**Original Mesh**  
(50k vertices, 290k tets,  
10 seconds)



**ODT smoothing**  
(global optimization,  
110s)

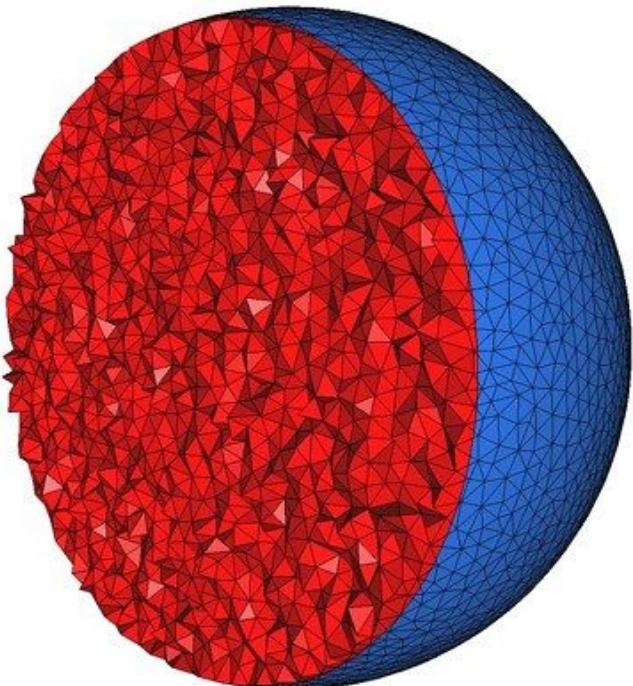


**ODT + Sliver  
perturbation**  
(local optimization, 40s)



# 3D Mesh Generation API

# Named Parameters



```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Labeled_mesh_domain_3<K> Mesh_domain;
typedef CGAL::Mesh_triangulation_3<Mesh_domain>::type Tr;
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;
typedef CGAL::Mesh_criteria_3<Tr> Mesh_criteria;

using CGAL::parameters;

int main()
{
    // Domain
    Mesh_domain domain =
        Mesh_domain::create_implicit_mesh_domain(
            function = [] (const K::Point_3& p)
            { return CGAL::squared_distance(
                p, K::Point_3(CGAL::ORIGIN))-1; },
            bounding_object = K::Sphere_3(CGAL::ORIGIN, 2.));

    // Set mesh criteria
    Mesh_criteria criteria(facet_angle=30, facet_size=0.1,
facet_distance=0.025,
cell_size=0.1);

    // Mesh generation
    C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
}
```

## Named Parameters

```
facet_angle=30, facet_size=0.1,
facet_distance=0.025,
cell_size=0.1;
```

# Named Parameters

```
// Set mesh criteria
Mesh_criteria criteria(    facet_angle=30,
                            facet_size=0.1,
                            facet_distance=0.025,
                            cell_size=0.1);

// Mesh generation
template<class C3T3 , class MD , class MC >
C3T3 CGAL::make\_mesh\_3(const MD& domain,
                      const MC&      criteria,
                      parameters::internal::Features_options features =
                        parameters::features(domain),
                      parameters::internal::Lloyd_options      lloyd = parameters::no_lloyd(),
                      parameters::internal::Odt_options        odt = parameters::no_odt(),
                      parameters::internal::Perturb_options     perturb = parameters::perturb(),
                      parameters::internal::Exude_options       exude = parameters::exude(),
                      parameters::internal::Manifold_options   manifold = parameters::non_manifold())
```

# Named Parameters and Parameter Functions

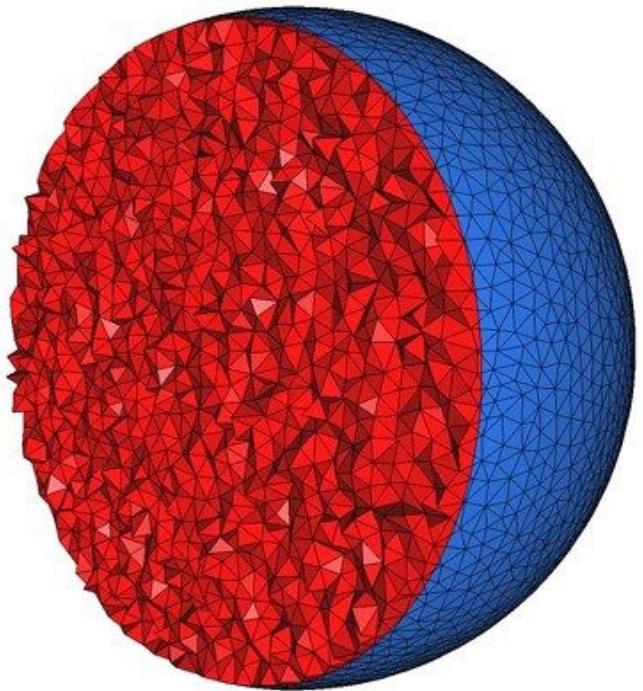
## Documentation of Parameter Functions

```
// Lloyd Optimization
parameters::internal::Lloyd_options
CGAL::parameters::lloyd(
    double parameters::time_limit = 0,
    std::size_t parameters::max_iteration_number = 0,
    double parameters::convergence = 0.02,
    double parameters::freeze_bound = 0.01,
    bool parameters::do_freeze = true)
```

## Usage code example

```
// Mesh generation with lloyd optimization step
C3t3 c3t3 = make_mesh_3<C3t3>(domain,
                                  criteria,
                                  parameters::lloyd(parameters::time_limit = 10));
```

# Mesh\_complex\_3\_in\_triangulation\_3



```
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Labeled_mesh_domain_3<K> Mesh_domain;
typedef CGAL::Mesh_triangulation_3<Mesh_domain>::type Tr;
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;
typedef CGAL::Mesh_criteria_3<Tr> Mesh_criteria;

using CGAL::parameters;

int main()
{
    // Domain
    Mesh_domain domain =
        Mesh_domain::create_implicit_mesh_domain(
            function = [] (const K::Point_3& p)
            { return CGAL::squared_distance(
                p, K::Point_3(CGAL::ORIGIN))-1; },
            bounding_object = K::Sphere_3(CGAL::ORIGIN, 2.));

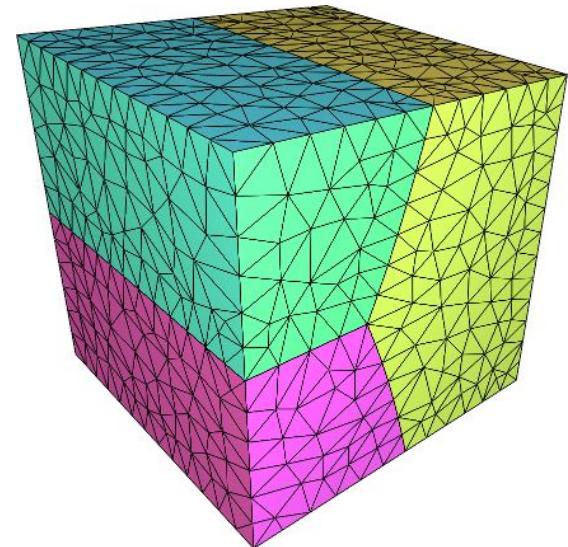
    // Set mesh criteria
    Mesh_criteria criteria(facet_angle=30, facet_size=0.1,
                           facet_distance=0.025,
                           cell_size=0.1);

    Mesh_complex_3_in_triangulation_3
    C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
}
```

# Mesh\_complex\_3\_in\_triangulation\_3

```
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;  
...  
...  
C3t3 c3t3 = CGAL::make_mesh_3<C3t3>(domain, criteria);
```

- A wrapper around a Triangulation\_3
- Description of a subset of geometric and combinatorial features
  - 0D - Corner vertices
  - 1D - Feature edges
  - 2D - Surface facets
  - 3D - Volume cells



# Mesh\_complex\_3\_in\_triangulation\_3 - API

```
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;  
C3t3 c3t3;
```

- API - for each dimension of simplex

- Accessors

```
bool b = c3t3.is_in_complex(simplex);  
Std::size_t n = c3t3.number_of_cells();
```

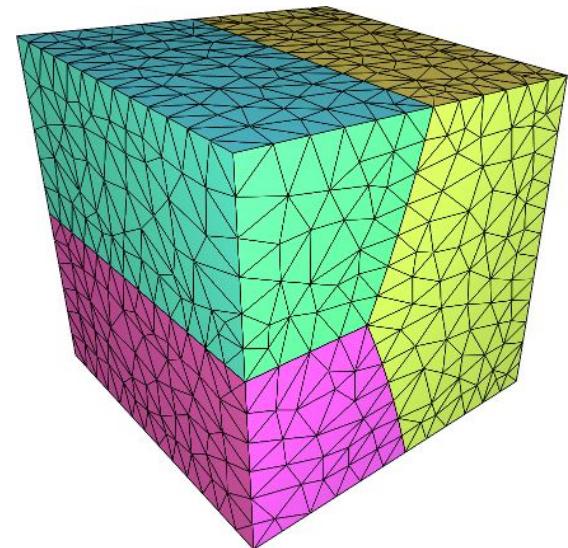
- Modifiers

```
c3t3.add_to_complex(simplex, simplex_index);  
c3t3.remove_from_complex(simplex);
```

- Iterators

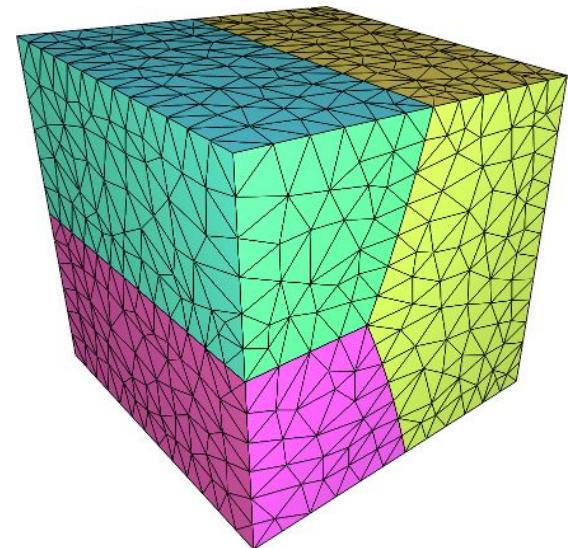
```
c3t3.cells_in_complex_begin();  
c3t3.cells_in_complex_end();
```

- C3t3 describes a complex  
of each dimension 0, 1, 2, 3



# Mesh\_complex\_3\_in\_triangulation\_3 - API

- Two Concepts
  - [MeshComplex\\_3InTriangulation\\_3](#)  
Cells and Facets
  - [MeshComplexWithFeatures\\_3InTriangulation\\_3](#)  
Edges and Vertices
- One Class
  - [CGAL::Mesh\\_complex\\_3\\_in\\_triangulation\\_3](#)

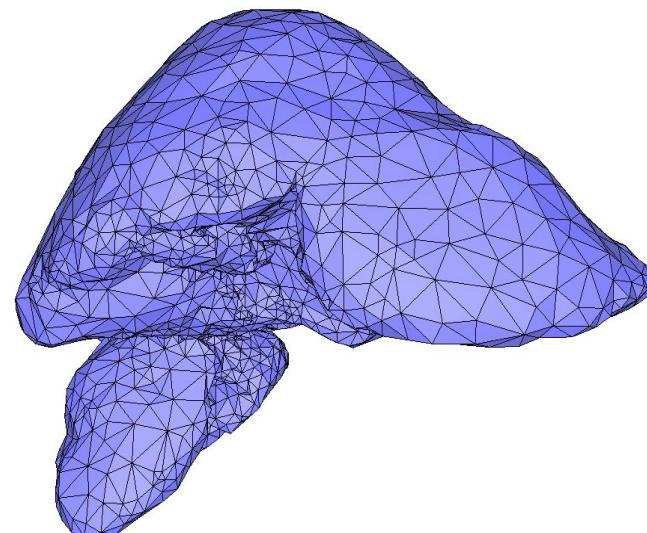
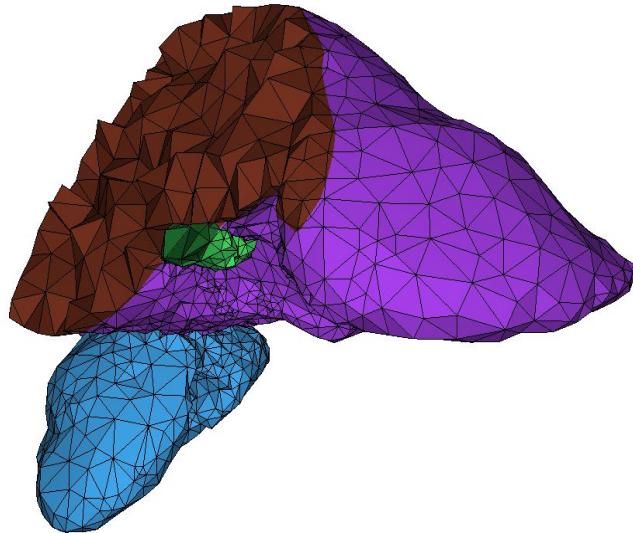


# Mesh\_complex\_3\_in\_triangulation\_3 - Export

```
typedef CGAL::Mesh_complex_3_in_triangulation_3<Tr> C3t3;  
C3t3 c3t3;
```

- Export facets complex

- to CGAL::Surface\_mesh      `CGAL::facets_in_complex_3_to_triangle_mesh(c3t3,surface_mesh);`
- to OFF file                  `c3t3.output_boundary_to_off(ostream);`



# Isotropic Tetrahedral Remeshing

# Isotropic Tetrahedral Remeshing

- Post-processing
- CGAL::Triangulation\_3
  - Triangulation of the convex hull
  - Non-Delaunay

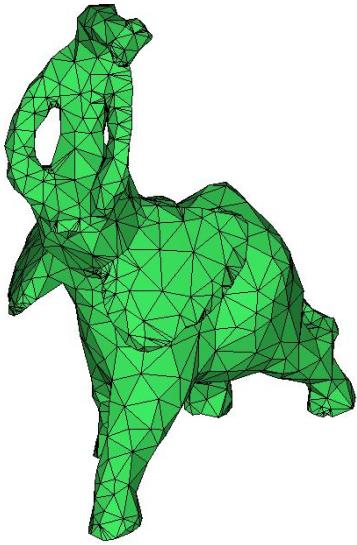
# Isotropic Tetrahedral Remeshing

- Post-processing
- CGAL::Triangulation\_3
- Iterative improvements of tetrahedral meshes by consecutive atomic operations
  - Edge split
  - Edge collapse
  - Edge flip
  - Global smoothing using vertex relocations
  - Re-projection of boundary vertices

# Isotropic Tetrahedral Remeshing

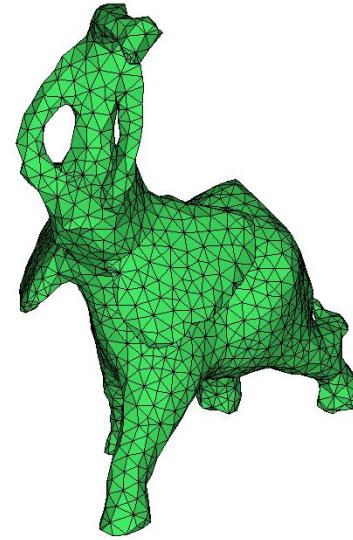
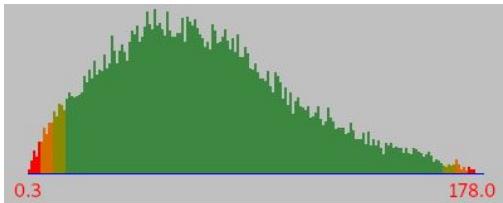
- Post-processing
- CGAL::Triangulation\_3
- Iterative improvements of tetrahedral meshes by consecutive atomic operations
  - Edge split
  - Edge collapse
  - Edge flip
  - Global smoothing using vertex relocations
  - Re-projection of boundary vertices
- Deals with
  - Multi-domains,
  - Boundaries,
  - Features
  - Preserves the geometry of subdomains throughout the remeshing process. [Faraj et al. 2016]

# Isotropic Tetrahedral Remeshing

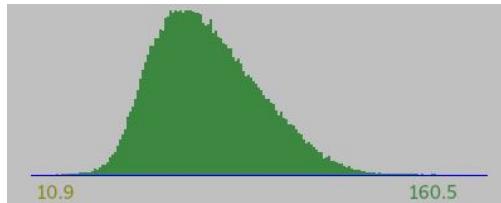


Input (Delaunay Mesh)

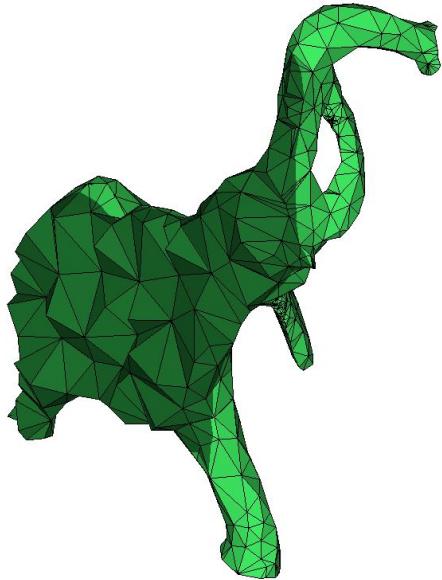
Dihedral  
angle  
distribution



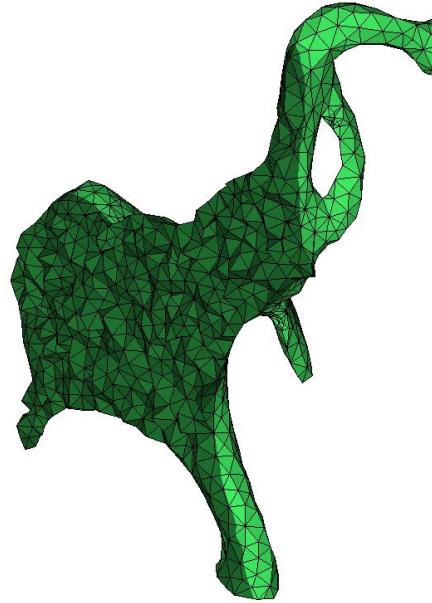
Input  
Remeshed



# Isotropic Tetrahedral Remeshing

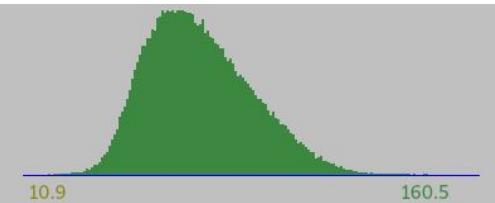
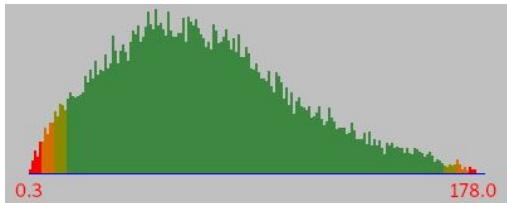


Input (Delaunay Mesh)

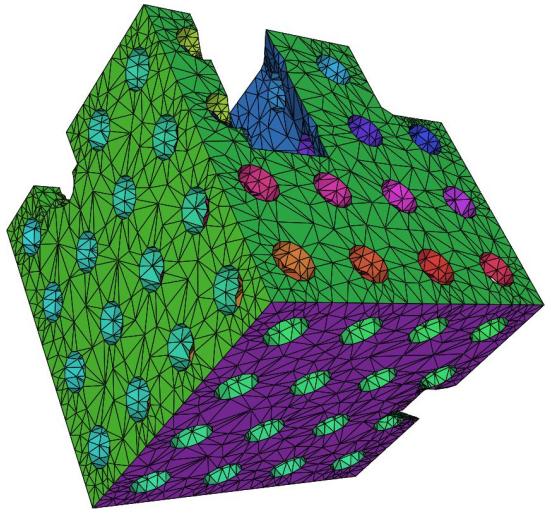


Input Remeshed

Dihedral  
angle  
distribution

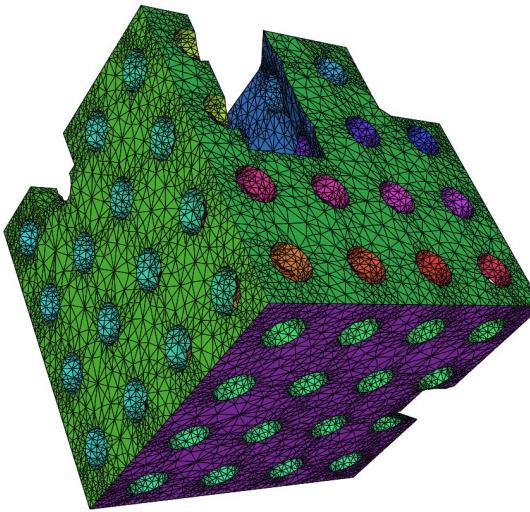
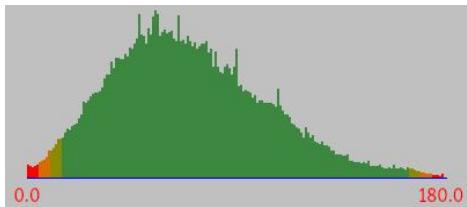


# Isotropic Tetrahedral Remeshing

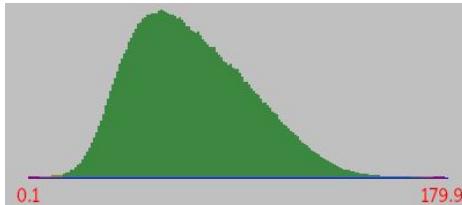


Input (Delaunay Mesh)

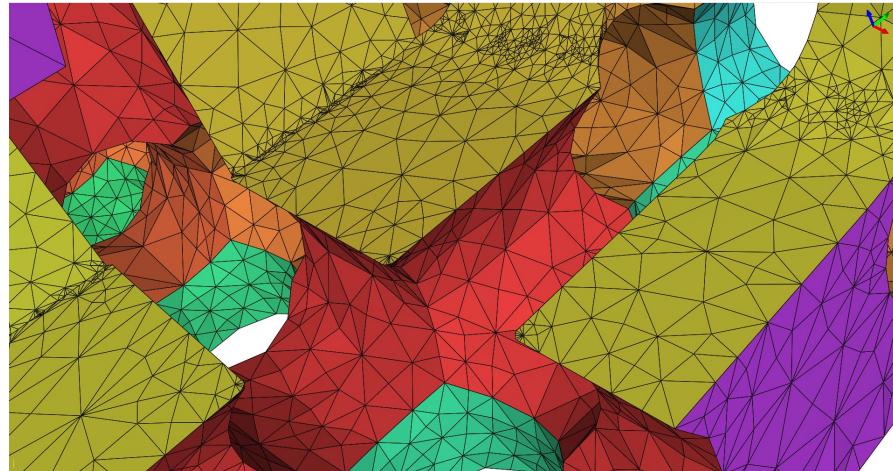
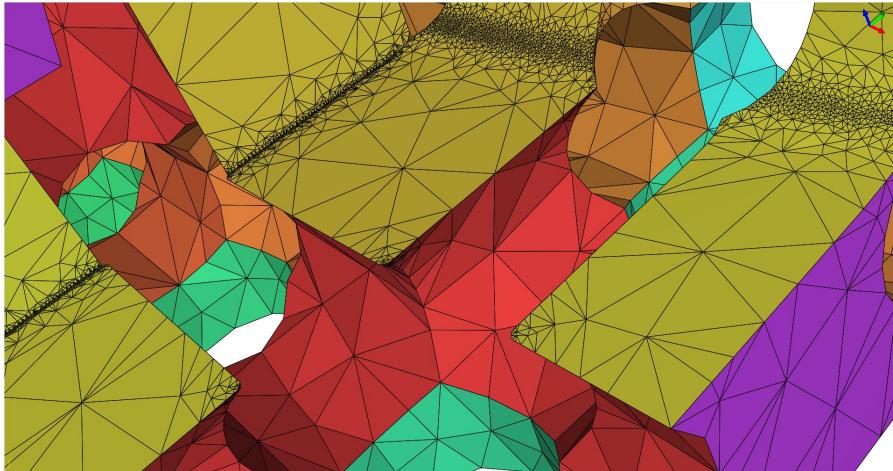
Dihedral  
angle  
distribution



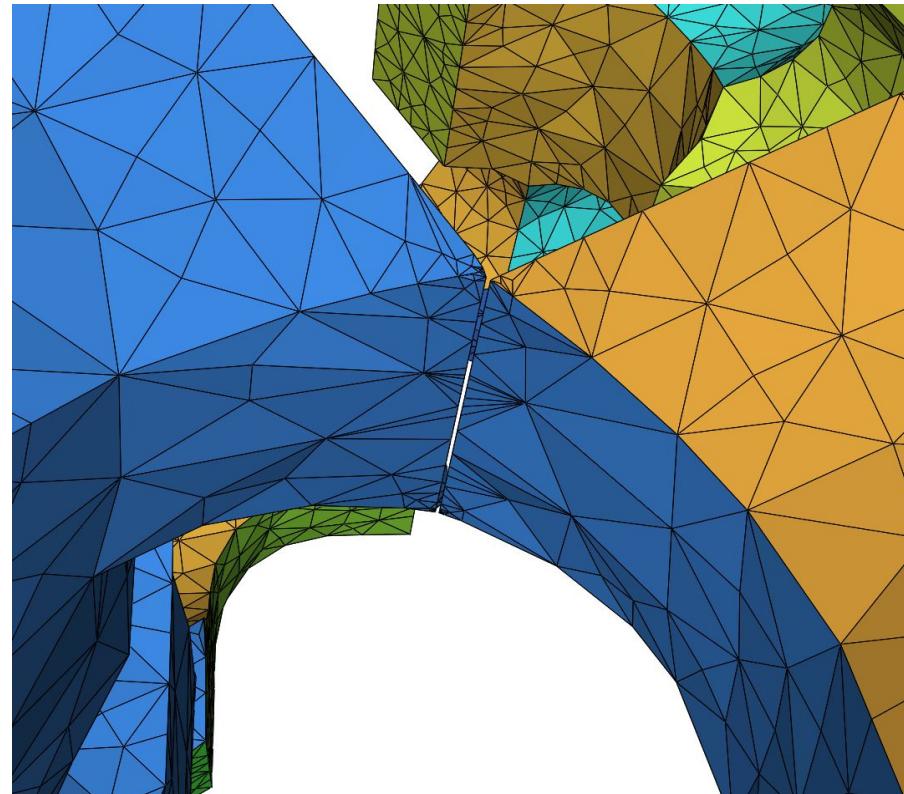
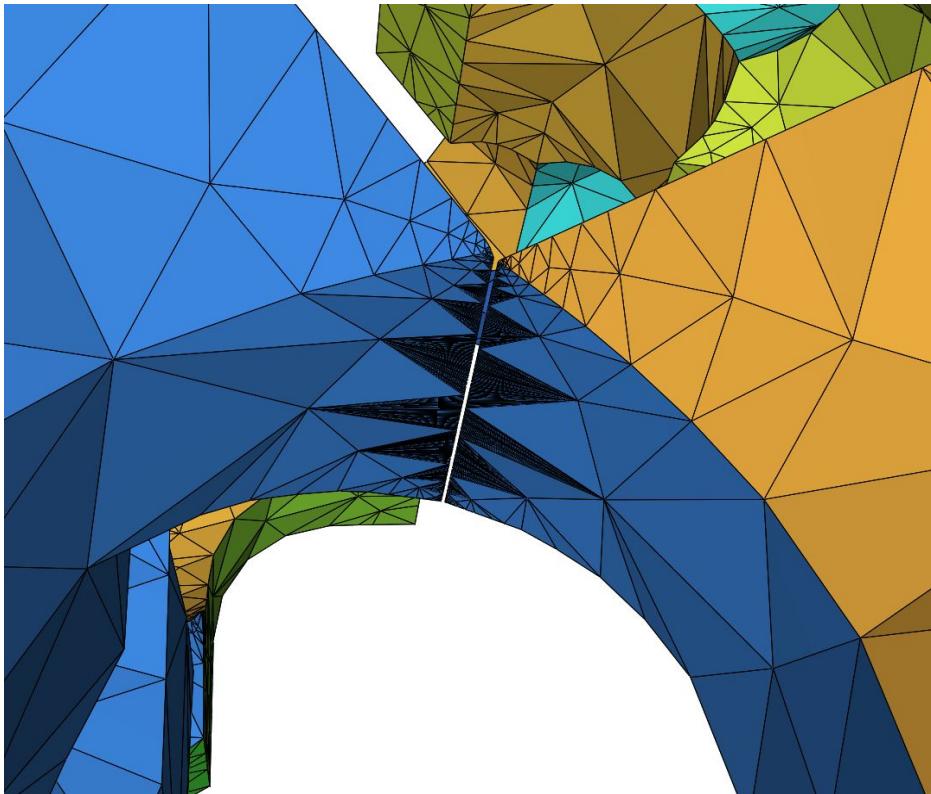
Input  
Remeshed



# Isotropic Tetrahedral Remeshing



# Isotropic Tetrahedral Remeshing



# Isotropic Tetrahedral Remeshing - API

```
CGAL::Triangulation_3 tr;  
...  
CGAL::tetrahedral_isotropic_remeshing(tr,  
                                target_edge_length);
```

## Optional Named Parameters

- number\_of\_iterations
- remesh\_boundaries
- smooth\_constrained\_edges
- edge\_is\_constrained\_map
- facet\_is\_constrained\_map
- cell\_is\_selected\_map

Thank you.