CGAL - the Computational Geometry Algorithms Library
Pierre Alliez, Andreas Fabri, Efi Fogel

To cite this version:

HAL Id: cel-00340448
https://cel.hal.science/cel-00340448
Submitted on 20 Nov 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Computational Geometry Algorithms Library

Pierre Alliez  Andreas Fabri  Efi Fogel
INRIA  GeometryFactory  Tel-Aviv University
Abstract

The CGAL Open Source Project provides *easy access to* efficient and reliable geometric algorithms in the form of a C++ library, offering geometric data structures and algorithms, which are efficient, robust, easy to use, and easy to integrate in existing software. The usage of de facto standard libraries increases productivity, as it allows software developers to focus on the application layer. This course is targeted at software developers with geometric needs, and course graduates will be able to select and use the appropriate algorithms and data structures provided by CGAL in their current or upcoming projects.

**Key Facts**

CGAL 3.3, released in June 2007: 90 software components, 600,000 lines of code, 3,500 user and reference manual pages, Cross platform support, Annual release with 12,000 downloads, 1,000 subscribers on the user mailing list, 40 subscribers on the developer mailing list. CGAL is used in many application areas by companies as Total (Oil&gas), British Telecom (Telecom), Cadence (VLSI), Leica Geosystems (GIS), Dassault Systèmes (CAD), The Moving Picture Company (Visual effects).

**CGAL Project**

The project is steered by an Editorial Board, it has a well defined development process, and the infrastructure for distributed development. The following research institutes and companies are actively involved or made contributions to the library: INRIA-Sophia-Antipolis, Max-Planck Institute for Computer Science, Tel-Aviv University, GeometryFactory, ETH Zurich, FU Berlin, University of Groningen, University of Utrecht, Stanford University, Athens University, and the Foundation of Research and Technology – Hellas. For more information on the project see [www.cgal.org](http://www.cgal.org) For an overview on what is in CGAL: [http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html](http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/packages.html)
Organization of the Course

This course starts with an overview followed by three in-depth sessions covering central data structures: The overview session presents the CGAL project and the design principles of CGAL. CGAL has adopted the exact computing paradigm, which yields robust and at the same time fast algorithms. CGAL further has adopted the generic programming paradigm, which makes CGAL particularly easy to customize and to integrate. Finally, we show how CGAL fits naturally with the STL, and the Boost graph library.

The session on polyhedral surfaces presents the underlying halfedge data structure and how it can be customized to user needs. We further present algorithms for polyhedral surfaces like parameterization, mesh subdivision and simplification, Boolean operations, and intersection detection.

The session on arrangements presents the arrangement API and several data structures built on top of it. These are 3D Minkowski sums, which can be used for collision detection, and 3D lower envelopes, which can be used for visibility map computations.

The last session covers the 2D and 3D triangulation API as well as the surface and volume mesh generators, which are based on Delaunay refinement.

The source code of the examples will be made available at http://www.cgal.org/Courses/SIGGRAPH2008.
Presenters

**Dr. Andreas Fabri (organizer)**
Chief Officer  
GeometryFactory  
20, Avenue Yves Emmanuel Baudoin 06130 Grasse, FRANCE  
Email: andreas.fabri@geometryfactory.com  
http://www.geometryfactory.com/

**Dr. Efi Fogel**
Senior Researcher  
Faculty of Exact Sciences. Tel Aviv University Tel Aviv 69978, ISRAEL  
Email: efifogel@gmail.com  
http://www.cs.tau.ac.il/~efif/

**Dr. Pierre Alliez**
Senior Researcher  
INRIA Sophia Antipolis – Mediterranee  
2004 route des Lucioles BP 93 - GEOMETRICA, FRANCE  
Email: pierre.alliez@sophia.inria.fr  
http://www-sop.inria.fr/geometrica/team/Pierre.Alliez/
Biographies

Andreas Fabri, PhD, GeometryFactory As member of the initial development team of the CGAL project, Andreas Fabri is one of the architects of the CGAL software. For several years he chaired the CGAL Editorial Board. In 2003 Andreas founded the GeometryFactory as spin-off of the CGAL project, offering licenses, service and support to commercial users, who cannot comply with the Open Source license of CGAL. Andreas received his PhD in computer science in 2004 from Ecole de Mines de Paris while working on geometric algorithms for parallel machines at INRIA.

Pierre Alliez obtained his PhD from Ecole nationale supérieure des Télécommunications, did his postdoc at Caltech, and is researcher at INRIA since 2001. His main research interests are on topics commonly referred to as Geometry Processing: geometry compression, surface approximation, mesh parameterization, surface remeshing and mesh generation. He is this year co-chair of the EUROGRAPHICS Symposium on Geometry Processing. In 2005 Pierre Alliez received the Eurographics Young Researcher Award.

Efi Fogel, MSc, Tel-Aviv University Efi Fogel is a co-founder of LucidLogix Ltd., a startup company that intends to deliver high performance 3D graphics systems. Efi Fogel is completing his Ph.D. studies at Tel-Aviv University. 3D Graphics and Computational Geometry are his main areas of interests. He is a member of the CGAL Editorial Board, and he is deeply involved with the design and implementation of the arrangement package of CGAL and its derivatives. Efi Fogel received his M.Sc. from Stanford University in 1989. He worked for Silicon Graphics Inc. (SGI) between 1989-1997 at the Advanced Graphics Devision, where he contributed to the specification of OpenGl among the other. After that Efi worked for Immersia Ltd, and he served as the CTO of Enbaya Ltd.
Course Syllabus

Overview                  Andreas      30’
Polyhedron                Pierre       40’
Arrangements              Efi          40’
Break                     20’
Triangulations & Meshes   Andreas, Pierre 80’
Wrap-up, Q&A              All          15’
Bibliography

- Bibliographic entries for individual chapters of CGAL manuals
- CGAL User and Reference Manual Bibliography
CGAL Contributors

CGAL Editorial Board
Sylvain Pion, Pierre Alliez, Eric Berberich, Andreas Fabri, Efi Fogel, Bernd Gärtner, Michael Hemmer, Michael Hoffmann, Menelaos Karavelas, Marc Pouget, Monique Teillaud, Ron Wein

CGAL Developers
Fernando Cacciola, Frédéric Cazals, Raphaëlle Chaine, Pavel Emeliyanenko, Marc Glisse, Luc Habert, Peter Hachemerger, Idit Haran, Thomas Herrmann, Samuel Hornus, Michael Kerber, Nico Kruithof, Sebastian Limbach, Sébastien Loriot, Pedro Machado Manhães de Castro, Andreas Meyer, Michal Meyerovitch, Thanh-Trung Nguyen, Luis Peñaranda, Joachim Reichel, Laurent Rineau, Daniel Russel, Laurent Saboret, Ophir Setter, Tel-Aviv University, George Tzoumas

Alumni

EU Project Board Members
Helmut Alt, Jean-Daniel Boissonnat, Dan Halperin, Kurt Mehlhorn, Stefan Näher, Mark Overmars, Geert Vegter, Emo Welzl, Peter Widmayer
Computational Geometry Algorithms Library

Pierre Alliez  Andreas Fabri  Efi Fogel
INRIA  GeometryFactory  Tel Aviv University
# Course Outline

<table>
<thead>
<tr>
<th>Topic</th>
<th>Instructor</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>Andreas</td>
<td>30’</td>
</tr>
<tr>
<td>Polyhedron</td>
<td>Pierre</td>
<td>40’</td>
</tr>
<tr>
<td>Arrangements</td>
<td>Efi</td>
<td>40’</td>
</tr>
<tr>
<td>Break</td>
<td></td>
<td>15’</td>
</tr>
<tr>
<td>2D Triangulations &amp; Meshes</td>
<td>Andreas</td>
<td>40’</td>
</tr>
<tr>
<td>3D Triangulations &amp; Meshes</td>
<td>Pierre</td>
<td>40’</td>
</tr>
<tr>
<td>Wrap-up, Q&amp;A</td>
<td></td>
<td>15’</td>
</tr>
</tbody>
</table>
Overview

Andreas Fabri
GeometryFactory
Mission Statement

“Make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications”

CGAL Project Proposal, 1996
Algorithms and Datastructures

Bounding Volumes
Polyhedral Surface
Boolean Operations

Triangulations
Voronoi Diagrams
Mesh Generation

Subdivision
Simplification
Parametrisation
Streamlines

Ridge Detection
Neighbour Search
Kinetic Datastructures

Lower Envelope
Arrangement
Intersection Detection
Minkowski Sum

PCA
Polytope distance
QP Sover
CGAL in Numbers

500,000 lines of C++ code
10,000 downloads/year (+ Linux distributions)
3,500 manual pages
3,000 subscribers to cgal-announce
1,000 subscribers to cgal-discuss
120 packages
60 commercial users
20 active developers
12 months release cycle
2 licenses: Open Source and commercial
Some Commercial Users

- Cadence
- Pulsic
- VDRC
- OHIO STATE UNIVERSITY
- ProtoMold
- Toshiba
- Indus Research Limited
- Agilent Technologies
- BAE Systems
- QinetiQ
- Leica Geosystems
- SAFE Software
- RM Data
- TRUEPOSITION
- ECL
- Monsanto
- MapInfo
- TOSCA
- Dassault Systemes
- DSXM
- ECL
- MPC
- TOSCA
- ST.JUDE MEDICAL
- TOMEC
- The MathWorks
- The MathWorks
- Agio
- TGS
- Midland Valley
- TOTAL
- BSAP
- Total
- Total
Why They Use CGAL

“I recommended to the senior management that we start a policy of buying-in as much functionality as possible to reduce the quantity of code that our development team would have to maintain.

This means that we can concentrate on the application layer and concentrate on our own problem domain.”

Senior Development Engineer
& Structural Geologist

Midland Valley Exploration
"My research group JYAMITI at the Ohio State University uses CGAL because it provides an efficient and robust code for Delaunay triangulations and other primitive geometric predicates. Delaunay triangulation is the building block for many of the shape related computations that we do. [...]"

Without the robust and efficient codes of CGAL, these codes could not have been developed."

Tamal Dey
Professor, Ohio State University
Project = « Planned Undertaking »

- Institutional members make a long term commitment: Inria, MPI, Tel-Aviv U, Utrecht U, Groningen U, ETHZ, GeometryFactory, FU Berlin, Forth, U Athens

- Editorial Board
  - Steers and animates the project
  - Reviews submissions

- Development Infrastructure
  - Gforge: svn, tracker, nightly testsuite,
  - 120p developer manual and mailing list
  - Two 1-week developer meetings per year
Contributions

- Submission of specifications of new contributions
- Review and decision by the Editorial Board

- Value for contributor
  - Integration in the CGAL community
  - Gain visibility in a mature project
  - Publication value for accepted contributions
Exact Geometric Computing
Predicates and Constructions

Predicates
- orientation
- in_circle

Constructions
- intersection
- circumcenter
Robustness Issues

• Naive use of floating-point arithmetic causes geometric algorithms to:
  – Produce [slightly] wrong output
  – Crash after invariant violation
  – Infinite loop

• There is a gap between
  – Geometry in theory
  – Geometry with floating-point arithmetic
Geometry in Theory

$\text{ccw}(s,q,r) \land \text{ccw}(p,s,r) \land \text{ccw}(p,q,s) \Rightarrow \text{ccw}(p,q,r)$

Correctness proofs of algorithms rely on such theorems.
The Trouble with Double

orientation\((p, q, r)\) = sign\(((p_x - r_x)(q_y - r_y) - (p_y - r_y)(q_x - r_x))\)
Make sure that the control flow in the implementation corresponds to the control flow with exact real arithmetic.
Filtered Predicates

• Generic functor adaptor Filtered_predicate<>
  – Try the predicate instantiated with intervals
  – In case of uncertainty, evaluate the predicate with multiple precision arithmetic

• Refinements:
  – Static error analysis
  – Progressively increase precision
Filtered Constructions

Lazy number = interval and arithmetic expression tree

\[(3.2 + 1.5) * 13\]

Lazy object = approximated object and geometric operation tree

Test that may trigger an exact re-evaluation:

\[
\text{if ( n' < m' )}
\]

\[
\text{if (collinear(a',m',b'))}
\]
The User Perspective

- **Convenience Kernels**
  - `Exact_predicates_inexact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel`
  - `Exact_predicates_exact_constructions_kernel_with_sqrt`

- **Number Types**
  - `double`, `float`
  - `CGAL::Gmpq` (rational), `Core` (algebraic)
  - `CGAL::Lazy_exact_nt<ExactNT>`

- **Kernels**
  - `CGAL::Cartesian<NT>`
  - `CGAL::Filtered_kernel<Kernel>`
  - `CGAL::Lazy_kernel<NT>`
Merits and Limitations

- Ultimate robustness inside the black box
- The time penalty is reasonable, e.g. 10% for 3D Delauny triangulation of 1M random points

Limitations of Exact Geometric Computing
- Topology preserving rounding is non-trivial
- Construction depth must be reasonable
- Cannot handle trigonometric functions
Generic Programming
template <class Key, class Less>
class set {
    Less less;

    insert(Key k)
    {
        if (less(k, treenode.key))
            insertLeft(k);
        else
            insertRight(k);
    }
};
template < class Geometry >
class Delaunay_triangulation_2 {
    Geometry::Orientation orientation;
    Geometry::In_circle in_circle;

    void insert(Geometry::Point t) {
        ...
        if(in_circle(p,q,r,t)) {...}
        ...
        if(orientation(p,q,r){...}
    }
};
CGAL Genericity

template < class Geometry, class TDS >
class Delaunay_triangulation_2 { }

};

template < class Vertex, class Face >
class Triangulation_data_structure_2 { }

};
Iterators

template <class Geometry>
class Delaunay_triangulation_2 {

    typedef .. Vertex_iterator;
    typedef .. Face_iterator;

    Vertex_iterator vertices_begin();
    Vertex_iterator vertices_end();

    template <class OutputIterator>
    incident_faces(Vertex_handle v, OutputIterator it);
}

std::list<Face_handle> faces;
dt.incident_faces(v, std::back_inserter(faces));
Iterators

template <class Geometry>

class Delaunay_triangulation_2 {

    template <class T>
        void insert(T begin, T end);// typeof(*begin)==Point
    

    list<Kernel::Point_2> points;

    Delaunay_triangulation<Kernel> dt;

    dt.insert(points.begin(), points.end());
Boost Graph Library (BGL)

- Rich collection of graph algorithms: shortest paths, minimum spanning tree, flow, etc.
- Design that
  - decouples data structure from algorithm
  - links them through a thin glue layer
- BGL and CGAL
  - Provide glue layer for CGAL
  - Extension to embedded graphs inducing the notion of faces
template <typename Graph >
struct boost::graph_traits {
    typedef ... vertex_descriptor;
    typedef ... edge_descriptor;
    typedef ... vertex_iterator;
    typedef ... out_edge_iterator;
};
vertex_descriptor v, w;
edge_descriptor e;

v = source(e, G);
w = target(e, G);

std::pair<out_edge_iterator, out_edge_iterator> ipair;

ipair = out_edges(v, G);
BGL Glue Layer for CGAL

CGAL provides partial specializations:

```cpp
template <typename T>
graph_traits<Polyhedron<T>>;
```

```cpp
template <typename T>
Polyhedron<T>::Vertex
source(Polyhedron<T>::Edge);
```

Users can run:

```cpp
boost::kruskal_mst(P);
```

Courtesy: P.Schroeder, Caltech
From A BGL Glue Layer for CGAL

BGL::Algorithm

<<Concept>>
Graph

boost::adjacency_list

CGAL::HDS
To BGL Style CGAL Algorithms

BGL::Algorithm

<<Concept>>
Graph

boost::adjacency_list

CGAL::Algorithm

<<Concept>>
EmbeddedGraph

CGAL::HDS

User::Polyhedron
Summary: Overview

- Huge collection with uniform APIs
- Modular and not monolithic
- Open Source and commercial licenses

- Clear focus on geometry
- Interfaces with de facto standards/leaders: STL, Boost, GMP, Qt, blas

- Robust and fast through exact geometric computing
- Easy to integrate through generic programming
Polyhedral Surfaces

Pierre Alliez
INRIA
Outline

- Halfedge Data Structure and Polyhedron
- Euler Operators
- Customization
- Algorithms for Geometric Modelling and Geometry Processing
Halfedge Data Structure

Represented by vertices, edges, facets and an **incidence relation** on them, restricted to orientable 2-manifolds with boundary.
Polyhedron

Building blocks assembled with C++ templates
Default Polyhedron

**Vertex**
- Halfedge_handle halfedge()
- Point& point()

**Halfedge**
- Halfedge_handle opposite()
- Halfedge_handle next()
- Halfedge_handle prev()
- Vertex_handle vertex()
- Facet_handle facet()

**Facet**
- Halfedge_handle halfedge()
- Plane& plane()
- Normal& normal()
- Color& color()

CGAL manual
Example

```cpp
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_3.h>

typedef CGAL::Simple_cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;
typedef Polyhedron::Vertex_iterator Vertex_iterator;

int main()
{
    Polyhedron p;
    // ... read from file or build

    Vertex_iterator v;
    for (v = p.vertices_begin(); v != p.vertices_end(); ++v)
        std::cout << v->point() << std::endl;
}
```
Flexible Data Structure

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Halfedge</th>
<th>Facet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halfedge_handle</td>
<td>halfedge()</td>
<td>Halfedge_handle</td>
</tr>
<tr>
<td>Point&amp;</td>
<td>point()</td>
<td>opposite()</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td>Halfedge_handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>next()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>prev()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Vertex_handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vertex()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Facet_handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>facet()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Color&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>color()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>....</td>
</tr>
</tbody>
</table>
Iterate over all Vertices

Vertex_iterator v;
for( v = polyhedron.vertices_begin();
    v != polyhedron.vertices_end();
    ++v )
{
    // do something with v
}
Circulate around Facet

```cpp
Halfedge_around_facet_circulator he, end;
he = end = f->facet_begin();
CGAL_For_all(he, end)
{
    // do something with he
}
```
Circulate around Vertex

```cpp
Halfedge_around_vertex_circulator he, end;
he = end = v->vertex_begin();
CGAL_For_all(he, end)
{
    // do something with he
}
```
Euler Operators

- `split_facet`
- `join_facet`
- `create_center_vertex`
- `erase_center_vertex`
- `split_vertex`
- `join_vertex` (aka edge collapse)
- `split_loop`
- `join_loop`
- `add_vertex_and_facet_to_border`
- `erase_facet`
- `add_facet_to_border`
- `erase_facet`
Customization

```cpp
template <class Refs>
struct MyFace : public CGAL::HalfedgeDS_face_base<Refs> {
    CGAL::Color color;
};

struct MyItems : public CGAL::Polyhedron_items_3 {
    template <class Refs, class Traits>
    struct Face_wrapper {
        typedef MyFace<Refs> Face;
    };
};

typedef CGAL::Simple_cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel, MyItems> MyPolyhedron;
```
Algorithms

- Intersection detection
- Convex hull
- Boolean operations
- Kernel
- Parameterization
- Subdivision
- Principal component analysis
- Estimation of curvatures
- Extraction of ridges
- Simplification
Intersection Detection

Efficient algorithm for finding all intersecting pairs for large numbers of axis-aligned bounding boxes.

Generic programming: Boxes can contain objects of any type
Example: Intersecting 3D Triangles

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/intersections.h>
#include <CGAL/box_intersection_d.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::Triangle_3 Triangle_3;
typedef std::list<Triangle_3>::iterator Iterator;
typedef CGAL::Box_intersection_d::Box_with_handle_d<double, 3, Iterator> Box;

void callback(const Box& a, const Box& b)
{
    Triangle_3 ta = *a.handle();
    Triangle_3 tb = *b.handle();
    if(CGAL::do_intersect(ta, tb)) {
        // do something
    }
}

std::list<Triangle_3> triangles(...);

std::list<Box> boxes;
Iterator it;
for(it = triangles.begin(); it != triangles.end(); ++it)
    boxes.push_back(Box((*it).bbox(), it));

CGAL::box_self_intersection_d(boxes.begin(), boxes.end(), callback);
```
Convex Hull

- From point set
- Outputs a polyhedron
```cpp
#include <CGAL/convex_hull_3.h>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef CGAL::Convex_hull_traits_3<Kernel> Traits;
typedef Traits::Polyhedron_3 Polyhedron;
typedef Kernel::Point_3 Point;

int main()
{
    std::list<Point> points;
    // fill container...
    Polyhedron polyhedron;
    CGAL::convex_hull_3(points.begin(), points.end(), polyhedron);
    return 0;
}
```
Boolean Operations

Operations:

• Union
• Difference
• Intersection
• Complement

Problem: not closed, i.e., result of a Boolean operation is not necessarily a Polyhedron
Nef Polyhedron

The 3D Nef polyhedron is a B-rep data structure which is
• closed under Boolean operations
• without enforcing regularization

Operations:
• Union
• Intersection
• Difference
• Complement
• Interior
• Exterior
• Boundary
• Closure
• Regularization

can evaluate a CSG-tree with halfspaces as primitives and convert it to B-rep

CGAL manual
Example

```cpp
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/Nef_polyhedron_3.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;
typedef CGAL::Nef_polyhedron_3<Kernel> Nef_polyhedron;

Polyhedron p1;
Polyhedron p2;
Nef_polyhedron n1(p1);
Nef_polyhedron n2(p2);
n1 -= n2; // difference

Polyhedron p3;
if(n1.is_simple())
  n1.convert_to_Polyhedron(p3);
else
  // analyze/process n1 and do something...
```
Example

\[ n1 -= n2; // difference \]
Kernel of a Polyhedron

- Intersection of all its interior half-spaces.
Kernel of a Polyhedron

- Intersection of all its interior half-spaces
- Uses linear programming (CGAL::QP_solver)

input polyhedron

kernel
Kernel w/ Linear Programming

Does it have a kernel?
Kernel w/ Linear Programming
Parameterization

**Planar**

- Conformal [Eck et al., Levy et al., Desbrun et al.]
- Mean value coordinates [Floater]
- ...
Example

```cpp
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/Parameterization_polyhedron_adaptor_3.h>
#include <CGAL/parameterize.h>

Polyhedron mesh;
Mesh_adaptor_polyhedron mesh_adaptor(&mesh);
CGAL::parameterize(&mesh_adaptor);
Point_2 uv = mesh_adaptor.info(he)->uv();
```
Parameterization

User-provided cut graph for closed or high genus surfaces.

(conformal distortion)
Parameterization

Fixed boundary

Free boundary
Subdivision

Designed to work on CGAL polyhedron

- Catmull-Clark
- Loop
- Doo-Sabin
- Sqrt3
- ...

CGAL manual
Example

```cpp
#include <CGAL/Cartesian.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/Subdivision_method_3.h>

typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Polyhedron;

using CGAL::Subdivision_method_3;

Polyhedron polyhedron;
int subdivision_depth = 3;
CatmullClark_subdivision(polyhedron, subdivision_depth);
```
Principal Component Analysis

Linear least squares fitting on sets of 3D kernel objects:

- points
- triangles

for triangle meshes

CGAL manual
Example

```cpp
#include <CGAL/linear_least_squares_fitting_3.h>
// more #includes and typedefs

Polyhedron mesh;

std::list<Triangle_3> triangles;
Polyhedron::Facet_iterator f;
for(f = mesh.facets_begin(); f != mesh.facets_end(); ++f) {
    const Point& a = f->halfedge()->vertex()->point();
    const Point& b = f->halfedge()->next()->vertex()->point();
    const Point& c = f->halfedge()->prev()->vertex()->point();
    triangles.push_back(Triangle_3(a,b,c));
}

Plane_3 plane;
CGAL::linear_least_squares_fitting_3( triangles.begin(),
                                       triangles.end(),
                                       plane,
                                       CGAL::PCA_dimension_2_tag() );
```
Same for Boost freaks 😊

```cpp
#include <CGAL/linear_least_squares_fitting_3.h>
#include <boost/iterator/transform_iterator.hpp>

Polyhedron mesh;

class ToTriangle {
  Triangle_3 operator(Polyhedron::FacetIterator f) {
    return Triangle_3( f->halfedge()->vertex()->point();
                        f->halfedge()->next()->vertex()->point();
                        f->halfedge()->prev()->vertex()->point() );
  }
};

Plane_3 plane;

CGAL::linear_least_squares_fitting_3(
    boost::make_transform_iterator(mesh.facets_begin(), ToTriangle()),
    boost::make_transform_iterator(mesh.facets_end(),   ToTriangle()),
    plane, CGAL::PCA_dimension_2_tag());
```
Fitting Points vs Triangles
Estimation of Curvatures

- Estimates general differential properties (Monge form) on point sets.
- Through polynomial (d-jet) fitting

CGAL manual
Example

```cpp
#include <CGAL/Monge_via_jet_fitting.h>
typedef CGAL::Cartesian<double> Kernel;
typedef CGAL::Monge_via_jet_fitting<Kernel> Monge_fit;
typedef Monge_fit::Monge_form Monge_form;

Monge_fit monge_fit;
Monge_form monge_form =
    monge_fit(points.begin(),
              points.end(),
              dim_fitting, dim_monge);

Vector_3 kmin = monge_form.minimal_principal_direction();
Vector_3 kmax = monge_form.maximal_principal_direction();
Vector_3 normal = monge_form.normal_direction();
```
Extraction of Ridges

Ridge: curve along which one of the principal curvatures has an extremum along its curvature line.
Simplification


BGL-style allows simplification of any model of EmbeddedGraph.
Simplification: Example

```cpp
#include <CGAL/Simple_cartesian.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/Surface_mesh_simplification/HalfedgeGraph_Polyhedron_3.h>
#include <CGAL/Surface_mesh_simplification/edge_collapse.h>
#include <CGAL/Surface_mesh_simplification/Policies/EdgeCollapse/Count_stop_predicate.h>

typedef CGAL::Simple_cartesian<double> Kernel;
typedef CGAL::Polyhedron_3<Kernel> Mesh;
namespace SMS = CGAL::Surface_mesh_simplification;

Mesh mesh;
SMS::Count_stop_predicate<Mesh> stop(1000); // target # edges
SMS::edge_collapse(mesh, stop,
    CGAL::vertex_index_map(boost::get(CGAL::vertex_external_index, mesh))
    .edge_index_map(boost::get(CGAL::edge_external_index, mesh)));
```

Summary and Outlook

• The halfedge data structure and the polyhedron are highly flexible
• CGAL provides algorithms for geometric modeling and geometry processing
• Polyhedral surface as output of surface mesh generation algorithms (Part IV)
Under Development

- BGL-ization of existing CGAL algorithms
Under Development

- BGL-ization of existing CGAL algorithms
- Remeshing
Questions?
Arrangements

Efi Fogel
Tel Aviv University
Outline

• Arrangements
• Algorithms based on Arrangements
  • Boolean Set Operations
  • Minkowski Sums and Polygon Offset
  • Envelopes
• Arrangements on Surfaces
Arrangement Definition

Given a collection of curves on a surface, the **arrangement** is the partition of the surface into **vertices**, **edges** and **faces** induced by the curves.

- An arrangement of circles in the plane
- An arrangement of lines in the plane
- An arrangement of geodesic arcs on the sphere
```cpp
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Arr_segment_traits_2.h>
#include <CGAL/Arrangement_2.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Arr_segment_traits_2<Kernel> Traits;
typedef Traits::Point_2 Point;
typedef Traits::X_monotone_curve_2 Segment;
typedef CGAL::Arrangement_2<Traits> Arrangement;

int main() {
    Point p1(0, 0), p2(1, 0), p3(0, 1);
    Segment cv[3] = { Segment(p1,p2), Segment(p2,p3), Segment(p3,p1) };
    Arrangement arr;
    CGAL::insert(arr, cv, cv+3);
    return (arr.is_valid()) ? 0 : -1;
}
```

Code: A Simple Arrangement

```
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Arr_segment_traits_2.h>
#include <CGAL/Arrangement_2.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Arr_segment_traits_2<Kernel> Traits;
typedef Traits::Point_2 Point;
typedef Traits::X_monotone_curve_2 Segment;
typedef CGAL::Arrangement_2<Traits> Arrangement;

int main() {
    Point p1(0, 0), p2(1, 0), p3(0, 1);
    Segment cv[3] = { Segment(p1,p2), Segment(p2,p3), Segment(p3,p1) };
    Arrangement arr;
    CGAL::insert(arr, cv, cv+3);
    return (arr.is_valid()) ? 0 : -1;
}
```
CGAL::Arrangement_2

- Constructs, maintains, modifies, traverses, queries, and presents subdivisions of the plane
- Robust and exact
  - All inputs are handled correctly (including degenerate)
  - Exact number types are used to achieve exact results
- Efficient
- Generic
  - Easy to interface, extend, and adapt
  - Notification mechanism for change propagation
- Modular
  - Geometric and topological aspects are separated
# Geometric Traits

- Define the family of curves
- Aggregate geometric types and operations over the types
- Compare two points
- Determine the relative position of a point and an $x$-monotone curve
- Determine the relative position of two $x$-monotone curves to the left (right) of a point
- Subdivide a curve into $x$-monotone curves
- Find all intersections of two $x$-monotone curves
# Geometric Traits

- Define the family of curves
- Aggregate geometric types and operations over the types

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare two points</td>
<td><img src="image1.png" alt="Compare two points" /></td>
</tr>
<tr>
<td>Determine the relative position of a point and an (x)-monotone curve</td>
<td><img src="image2.png" alt="Determine relative position" /></td>
</tr>
<tr>
<td>Determine the relative position of two (x)-monotone curves to the left (right) of a point</td>
<td><img src="image3.png" alt="Determine relative position" /></td>
</tr>
<tr>
<td>Subdivide a curve into (x)-monotone curves</td>
<td><img src="image4.png" alt="Subdivide" /></td>
</tr>
<tr>
<td>Find all intersections of two (x)-monotone curves</td>
<td><img src="image5.png" alt="Find intersections" /></td>
</tr>
</tbody>
</table>
**Geometric Traits**

- Define the family of curves
- Aggregate geometric types and operations over the types

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare two points</strong></td>
</tr>
<tr>
<td><strong>Determine the relative position of a point and an x-monotone curve</strong></td>
</tr>
<tr>
<td><strong>Determine the relative position of two x-monotone curves to the left (right) of a point</strong></td>
</tr>
<tr>
<td><strong>Subdivide a curve into x-monotone curves</strong></td>
</tr>
<tr>
<td><strong>Find all intersections of two x-monotone curves</strong></td>
</tr>
</tbody>
</table>
Geometric Traits

- Define the family of curves
- Aggregate geometric types and operations over the types
- Compare two points
- Determine the relative position of a point and an x-monotone curve
- Determine the relative position of two x-monotone curves to the left (right) of a point
- Subdivide a curve into x-monotone curves
- Find all intersections of two x-monotone curves
# Geometric Traits

- Define the family of curves
- Aggregate geometric types and operations over the types

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare two points</td>
<td></td>
</tr>
<tr>
<td>Determine the relative position of a point and an x-monotone curve</td>
<td></td>
</tr>
<tr>
<td>Determine the relative position of two x-monotone curves to the left (right) of a point</td>
<td></td>
</tr>
<tr>
<td>Subdivide a curve into x-monotone curves</td>
<td></td>
</tr>
<tr>
<td>Find all intersections of two x-monotone curves</td>
<td></td>
</tr>
</tbody>
</table>
## Arrangement Traits Classes

<table>
<thead>
<tr>
<th>Curve Family</th>
<th>Degree</th>
<th>Surface</th>
<th>Boundness</th>
<th>Arithmetic</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear segments</td>
<td>1</td>
<td>plane</td>
<td>bounded</td>
<td>rational</td>
<td>caching noncaching</td>
</tr>
<tr>
<td>linear segments, rays, lines</td>
<td>1</td>
<td>plane</td>
<td>unbounded</td>
<td>rational</td>
<td></td>
</tr>
<tr>
<td>piecewise linear curves</td>
<td>∞</td>
<td>plane</td>
<td>bounded</td>
<td>rational</td>
<td>caching noncaching</td>
</tr>
<tr>
<td>circular arcs, linear segments</td>
<td>≤ 2</td>
<td>plane</td>
<td>bounded</td>
<td>rational</td>
<td>CK</td>
</tr>
<tr>
<td>algebraic curves</td>
<td>≤ 2</td>
<td>plane</td>
<td>Bounded unbounded</td>
<td>algebraic</td>
<td>CORE CKvA_2</td>
</tr>
<tr>
<td>quadric projections</td>
<td>≤ 2</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>algebraic curves</td>
<td>≤ 3</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>algebraic curves</td>
<td>≤ n</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>planar Bézier curves</td>
<td>≤ n</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>univariate polynomials</td>
<td>≤ n</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td>RS</td>
</tr>
<tr>
<td>rational function arcs</td>
<td>≤ n</td>
<td>plane</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>geodesic arcs on sphere</td>
<td>≤ 2</td>
<td>sphere</td>
<td>bounded</td>
<td>rational</td>
<td></td>
</tr>
<tr>
<td>quadric intersection arcs</td>
<td>≤ 2</td>
<td>quadric</td>
<td>unbounded</td>
<td>algebraic</td>
<td></td>
</tr>
<tr>
<td>dupin cyclide intersection. arcs</td>
<td>≤ 2</td>
<td>dupin cyclides</td>
<td>bounded</td>
<td>algebraic</td>
<td></td>
</tr>
</tbody>
</table>
Enrich Vertices, Edges, Faces

template <typename Traits, typename Dcel = Arr_default_dcel<Traits_> >
class Arrangement_2 {
    . . .
};

enum Color {BLUE, RED, WHITE};

typedef CGAL::Arr_segment_traits_2<Kernel>                        Traits;
typedef Traits::Point_2                                          Point;
typedef Traits::X_monotone_curve_2                               Segment;
typedef CGAL::Arr_extended_dcel<Traits, Color, Color, Color>     Dcel;
typedef CGAL::Arrangement_2<Traits, Dcel>                        Arrangement;
Point Location

Given a subdivision $A$ of the space into cells and a query point $q$, find the cell of $A$ containing $q$.

- Fast query processing
- Reasonably fast preprocessing
- Small space data structure

<table>
<thead>
<tr>
<th></th>
<th>Naive</th>
<th>Walk</th>
<th>RIC</th>
<th>Landmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing time</td>
<td>none</td>
<td>none</td>
<td>$O(n \log n)$</td>
<td>$O(k \log k)$</td>
</tr>
<tr>
<td>Memory space</td>
<td>none</td>
<td>none</td>
<td>$O(n)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>Query time</td>
<td>bad</td>
<td>reasonable</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>Applicability</td>
<td>all</td>
<td>limited</td>
<td>limited</td>
<td>limited</td>
</tr>
</tbody>
</table>

**Walk** — Walk along a line  

**RIC** — Random Incremental Construction based on trapezoidal decomposition

$k$ — number of landmarks
typedef CGAL::Arr_naive_point_location<Arrangement_2> Naive_pl;
typedef CGAL::Arr_landmarks_point_location<Arrangement_2> Landmarks_pl;

int main() {
    Arrangement arr;
    construct_arr(arr);
    Point p(1, 4);

    Naive_pl naive_pl(arr);  // Associate arrangement to naïve point location
    CGAL::Object obj1 = naive_pl.locate(p);

    Landmarks landmarks_pl;
    landmarks_pl.attach(arr);  // Attach landmarks point location to arrangement
    CGAL::Object obj2 = landmarks_pl.locate(p);

    return (obj1 == obj2) ? 0 : -1;
}
### Boolean Set Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complement</td>
<td>$R = \overline{P}$</td>
</tr>
<tr>
<td>Union</td>
<td>$R = P \cup Q$</td>
</tr>
</tbody>
</table>
| Intersection         | $R = P \cap Q$  
                      | $= \overline{P} \cup \overline{Q}$ |
| Difference           | $R = P \setminus Q$  
                      | $= P \cap \overline{Q}$ |
| Symmetric Difference | $R = (P \setminus Q) \cup (Q \setminus P)$ |
| Intersection predicate | $P \cap Q \neq \emptyset$  
                      | Overlapping cell(s) are not explicitly computed |
| Containment predicate | $r \in P$ |
| Interior, Boundary, Closure | $R = \text{closure} (\text{interior}(P))$ |

For two point sets $P$ and $Q$ and a point $r$:
int main() {
    Polygon p, q;
    p.push_back(Point(0, 0)); p.push_back(Point(2, 0));
    p.push_back(Point(1, 1)); p.push_back(Point(0, 2));
    q.push_back(Point(1, 1)); q.push_back(Point(3, 1));
    q.push_back(Point(2, 2)); q.push_back(Point(1, 3));

    Polygon_with_holes comp_p, comp_q;
    CGAL::complement(p, comp_p);
    CGAL::complement(q, comp_q);

    Polygon_with_holes a;
    CGAL::join(comp_p, comp_q, a);

    std::list<Polygon_with_holes> l1, l2;
    CGAL::complement(a, std::back_inserter(l1));
    CGAL::intersection(p, q, l2);

    return std::compare(l1, l2);
}
CGAL::Boolean_set_operation_2

- Supports
  - regularized Boolean set-operations
  - intersection predicates
  - point containment predicates
- Operands and results are regularized point sets bounded by \( x \)-monotone curves referred to as general polygons
  - General polygons may have holes
- Extremely efficient aggregated operations
- Based on the \texttt{Arrangement_2} and \texttt{Polygon_2} packages
Minkowski Sum in $\mathbb{R}^d$

$P$ and $Q$ are 2 polytopes in $\mathbb{R}^d$

$$P \oplus Q = \{p + q \mid p \in P, q \in Q\}$$

$P \cap Q \neq \emptyset \iff \text{Origin} \in P \oplus (-Q)$

Collision detection
CGAL::Minkowski_sum_2

• Based on the **Arrangement_2**, **Polygon_2**, and **Partition_2** packages

• Supports Minkowski sums of two simple polygons
  – Implemented using either decomposition or convolution
  – Exact

• Interoperable with **Boolean_set_operations_2**, e.g., Compute the union of offset polygons

• Supports Minkowski sums of a simple polygon and a disc (polygon offsetting)
  – Offers either an exact computation or a conservative approximation scheme
  – Disk radius can be negative (inner offset)
Motion Planning

- The input robot and the obstacle are non-convex
- Exploits the convolution method
- The output sum contains four holes, isolated points, and antennas
Envelopes in $\mathbb{R}^3$

The **lower envelope** of a set of $xy$-monotone surfaces $S = \{S_1, S_2, \ldots, S_n\}$, is the point-wise minimum of all surfaces. The **minimization diagram** of $S$ is an arrangement.

- The identity of the surfaces that induce the lower envelope over a specific cell (vertex, edge, face) of the arrangement is the same.
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Env_triangle_traits_3.h>
#include <CGAL/Env_surface_data_traits_3.h>
#include <CGAL/envelope_3.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Env_triangle_traits_3<Kernel>               Traits;
typedef Traits::Point_3                                   Point;
typedef Traits::Surface_3                                 Tri;
enum Color {RED, BLUE};
typedef CGAL::Env_surface_data_traits_3<Traits, Color>    Data_traits;
typedef Data_traits::Surface_3                            Dtri;
typedef CGAL::Envelope_diagram_2<Data_traits>             Arrangement;

int main() {
    Point p1(0,0,1), p2(0,6,1), p3(5,3,5), p4(6,0,1), p5(6,6,1), p6(1,3,5);
    Tri tris[] = {Dtri(Tri(p1,p2,p3), RED), Dtri(Tri(p4,p5,p6), BLUE)};
    Arrangement arr;
    CGAL::lower_envelope_3(tris, tris+2, arr);
    return (arr.is_valid()) ? 0 : -1;
}
CGAL::Envelope_3

- Constructs lower and upper envelopes of surfaces

<table>
<thead>
<tr>
<th>Surface Family</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>triangles</td>
<td>Env_triangle_traits_3</td>
</tr>
<tr>
<td>spheres</td>
<td>Env_sphere_traits_3</td>
</tr>
<tr>
<td>planes and half planes</td>
<td>Env_plane_traits_3</td>
</tr>
<tr>
<td>quadrics</td>
<td>Env_quadric_traits_3</td>
</tr>
</tbody>
</table>

- Based on the **Arrangement_2** package

- Exploits
  - Overlay computation (using the sweep line framework)
  - Isolated points
  - Zone traversal
Lower Envelopes
Voronoi Diagrams via Envelopes

- Computed as lower envelopes of planes
- Represented as planar arrangements of unbounded curves

points along a line segment
points on a grid inside a square
points on a circle
points on a square boundary
Arrangements on Surfaces in $\mathbb{R}^3$

A **parametric surface** $S$ of two parameters is a surface defined by parametric equations involving two parameters $u$ and $v$:

$$f_S(u, v) = (x(u, v), y(u, v), z(u, v))$$

Thus, $f_S : P \to \mathbb{R}^3$ and $S = f_S(P)$, where $P$ is a continuous and simply connected two-dimensional parameter space.

We deal with orientable parametric surfaces.
Minkowski-Sums of Polytopes

- The Gaussian map of a polytope is the decomposition of $S^2$ into maximal connected regions so that the extremal point is the same for all directions within one region.
- The overlay of the Gaussian maps of two polytopes $P$ and $Q$ is the Gaussian map of the Minkowski sum of $P$ and $Q$. 
Voronoi Diagrams on the Sphere

- All algorithms supported by the Arrangement_2 package can also be used on the sphere.
- We compute lower envelopes defined over the sphere.
- We can compute Voronoi diagrams on the sphere, the edges of which are geodesic arcs.
Arrangements on the Sphere

- The overlay of
  - An arrangement on the sphere induced by
    - the continents and some of the islands on earth
    - 5 cities
      - New Orleans
      - Los Angeles
      - San Antonio
      - San Diego
      - Boston
    - Voronoi diagram of the cities

- Diagrams, envelopes, etc. are represented as arrangements
  - Can be passed as input to consecutive operations

The sphere is oriented such that Cambridge is at the center
Video

- Arrangements of Geodesic Arcs on the Sphere
- Was presented at the 24th ACM Symposium on Computational Geometry, College Park, Maryland, July 2008
Summary

- Arrangements are versatile tools
- Arrangements are used as foundation for higher level geometric data structures
- Arrangements are not bound to the plane
Triangulations and Mesh Generation

Andreas Fabri
GeometryFactory

Pierre Alliez
INRIA
Outline

• **2D**
  • From triangulations to quality meshes
  • Related components

• **3D**
  • Triangulations
  • Mesh generation
    • Key concepts
    • Surfaces
    • Volumes
    • Next release and work in progress

• **Summary**
2D
From Triangulations to Quality Meshes
Delaunay Triangulation

- A triangulation is a **Delaunay triangulation**, if the circumscribing circle of any facet of the triangulation contains no vertex in its interior.
```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point;

typedef CGAL::Delaunay_triangulation_2<Kernel> Delaunay;
typedef Delaunay::Vertex_handle Vertex_handle;

int main()
{
    Delaunay dt;

    dt.insert( std::istream_iterator<Point>(std::cin),
               std::istream_iterator<Point>() );

    Vertex_handle v = dt.nearest_vertex(Point(0.0,0.0));

    std::cout << "Nearest vertex to origin: " << v->point() << std::endl;

    return 0;
}
```
Adding Constraints
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Constrained_Delaunay_triangulation_2.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point;

typedef CGAL::Constrained_Delaunay_triangulation_2<Kernel> CDT;
typedef CDT::Vertex_handle Vertex_handle;

int main()
{
  CDT cdt;

  // from points
  cdt.insert_constraint(Point(0.0,0.0), Point(1.0,0.0));

  // from vertices
  Vertex_handle v1 = cdt.insert(Point(2.0,3.0));
  Vertex_handle v2 = cdt.insert(Point(4.0,5.0));
  cdt.insert_constraint(v1,v2);

  return 0;
}
Conforming Delaunay
#include <CGAL/Triangulation_conformer_2.h>

// constrained Delaunay triangulation
CDT cdt;
... // insert points & constraints

CGAL::make_conforming_Delaunay_2(cdt);
```c++
#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;

int main()
{
    CDT cdt;
    ... // insert points and constraints
    CGAL::refine_Delaunay_mesh_2(cdt);
    return 0;
}
```
Background
Delaunay Edge

An edge is said to be a **Delaunay edge**, if it is inscribed in an empty circle
An edge is said to be a *Gabriel edge*, if its diametral circle is empty.
Conforming Delaunay Triangulation

A constrained Delaunay triangulation is a **conforming Delaunay triangulation**, if every constrained edge is a Delaunay edge.

![Non conforming Delaunay triangulation](image1)

![Conforming Delaunay triangulation](image2)
A constrained Delaunay triangulation is a conforming Gabriel triangulation, if every constrained edge is a Gabriel edge.
Steiner Vertices

Any constrained Delaunay triangulation can be refined into a conforming Delaunay or Gabriel triangulation by adding **Steiner vertices**.
Delaunay Refinement

**Rule #1**: break *bad* elements by inserting circumcenters (Voronoi vertices)
- "bad" in terms of *size* or *shape* (too big or skinny)

Picture taken from [Shewchuk]
Delaunay Refinement

**Rule #2:** Midpoint vertex insertion

A constrained segment is said to be *encroached*, if there is a vertex inside its diametrical circle.

Picture taken from [Shewchuk]
Delaunay Refinement

Encroached subsegments have priority over skinny triangles

Picture taken from [Shewchuk]
API
Rich API

- Traversal
Rich API

- Traversal
- Localization
Rich API

- Traversal
- Localization
- Dynamic: insertion & removal
Rich API

- Traversal
- Localization
- Dynamic: insertion & removal
- Parameters for mesh generation
Parameters for Mesh Generation

- **Shape**
  - Lower bound on triangle angles

Input PLSG  |  5 deg  |  20.7 deg

Online manual
Parameters for Mesh Generation

- **Shape**
  - Lower bound on triangle angles

- **Size**
  - No constraint
  - Uniform sizing
  - Sizing function
Sizing Parameter

No constraint | Uniform | Sizing function
Example Code

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Constrained_Delaunay_triangulation_2.h>
#include <CGAL/Delaunay_mesher_2.h>
#include <CGAL/Delaunay_mesh_size_criteria_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> Criteria;
typedef CGAL::Delaunay_mesher_2<CDT, Criteria> Meshing_engine;

int main()
{
    CDT cdt;
    Meshing_engine engine(cdt);
    engine.refine_mesh();
    engine.set_criteria(Criteria(0.125, 0.5));  // min 20.6 deg
    // 0.5 for sizing
    engine.refine_mesh();  // refine once more, etc.
    return 0;
}
```
Parameters for Mesh Generation

- **Shape**
  - Lower bound on triangle angles

- **Size**
  - No constraint
  - Uniform sizing
  - Sizing function

- **Seeds**
  - Exclude/include components
Performances

Kilimandjaro elevation contour lines (38K segments)
Performances

Refinement: 15K vertices/s
Related Components

- Voronoi diagram
Related Components

- Voronoi diagram
- Elevation (through traits class)

Courtesy IPF, Vienna University of Technology & Inpho GmbH
Related Components

- Voronoi diagram
- Elevation
- Interpolation (natural neighbors)
Related Components

- Voronoi diagram
- Elevation
- Interpolation
- Placement of streamlines
3D Triangulations

- Delaunay
- Regular
- Rich API
- Fully dynamic
- 1M points in 16s

Online manual

Tetrahedron
Mesh Generation

Key concepts:

- Delaunay filtering
- Delaunay refinement
Delaunay Filtering

Dual Voronoi edge

Voronoi edge \cap surface S

Delaunay triangulation restricted to surface S
Delaunay Refinement

Steiner point

**Bad facet** = big or badly shaped or large approximation error
Surface Mesh Generation Algorithm

repeat
{ pick bad facet \( f \)
insert furthest \( \text{dual}(f) \cap S \) in Delaunay triangulation
update Delaunay triangulation restricted to \( S \)
}
until all facets are good
Isosurface from 3D Grey Level Image

```cpp
#include <CGAL/Surface_mesh_default_triangulation_3.h>
#include <CGAL/Complex_2_in_triangulation_3.h>
#include <CGAL/make_surface_mesh.h>
#include <CGAL/Gray_level_image_3.h>
#include <CGAL/Implicit_surface_3.h>

typedef CGAL::Surface_mesh_default_triangulation_3<Tr> Tr;
typedef CGAL::Complex_2_in_triangulation_3<Tr> C2t3;
typedef CGAL::Implicit_surface_3<Kernel, Gray_level_image> Surface_3;

Tr tr; // 3D-Delaunay triangulation
C2t3 c2t3 (tr); // 2D-complex in 3D-Delaunay triangulation
Gray_level_image image("data/brain",128);
Surface_3 surface(image, bounding_sphere, 1e-2);

// Criteria: min triangle angles, size, approximation error,
CGAL::Surface_mesh_default_criteria_3<Tr> criteria(30.,5.,5.);
CGAL::make_surface_mesh(c2t3, surface, criteria, CGAL::Manifold_tag());
```
Output Mesh Properties

• Well shaped triangles
  – Lower bound on triangle angles

• Homeomorphic to input surface

• Manifold
  – not only combinatorially, i.e., no self-intersection

• Faithful Approximation of input surface
  – Hausdorff distance
  – Normals
Delaunay Refinement vs Marching Cubes

Delaunay refinement  Marching cubes in octree
Surface Remeshing

Input is a polyhedral surface

(requires efficient data structures for intersection computations)
Parameters

- Shape of triangles
  - lower bound on triangle angles
- Size
Parameters

- **Shape of triangles**
  - lower bound on triangle angles

- **Size**
  - No constraint
Parameters

- Shape of triangles
  - lower bound on triangle angles
- Size
  - No constraint
  - Uniform sizing
Parameters

- Shape of triangles
  - lower bound on triangle angles
- Size
  - No constraint
  - Uniform sizing
  - Sizing function
Parameters

• Shape of triangles
  – lower bound on triangle angles

• Size
  – No constraint
  – Uniform sizing
  – Sizing function

• Approximation error
Uniform vs Adapted
Mesh Generation Framework

Refinement

while(simplex is bad)
refine(simplex)

Oracle
- 3D image
- Surface mesh
- Implicit function

Constraints
- Size
- Shape
- Approximation

...
A Versatile Framework

- 3D grey level images
- 3D multi-domain images
- Implicit function: $f(x, y, z) = \text{constant}$
- Surface mesh (remeshing)
- Point set (surface reconstruction)
- Anything which provides intersections
Next Release
Volume Mesh Generation
More Delaunay Filtering

Delaunay triangulation restricted to domain $\Omega$

Dual Voronoi vertex inside domain $\Omega$ (“oracle”)

tetrahedron
circumsphere
Delaunay Refinement

Steiner point

Bad tetrahedron = big or badly shaped
Volume Mesh Generation Algorithm

repeat
{
    pick bad simplex
    if (Steiner point encroaches a facet)
        refine facet
    else
        refine simplex
        update Delaunay triangulation restricted to domain
}
until all simplices are good
Exude slivers
Tetrahedron Zoo

sliver
Sliver Exudation [Edelsbrunner-Guoy]

- Delaunay triangulation turned into a regular triangulation with null weights.
- Small increase of weights triggers edge-facets flips to remove slivers.
Sliver Exudation Process

- **Try** improving all tetrahedra with an aspect ratio lower than a given bound
- Never flips a boundary facet
3D Mesh from Multi-Domain Images

Tr tr;  // 3D Delaunay triangulation
C2t3 c2t3(tr);  // 2D complex in 3D-Delaunay triangulation

Image_3 image("segmented_visible_human");
Mesh_traits mesh_traits(image);
Facets_criteria facets_criteria(5, 1);  // facet sizing
    // approximation error
TetsCriteria tets_criteria(5);  // tet sizing

// 0.5 = radius-radius ratio upper bound for sliver exudation
CGAL::make_mesh_3_for_multivolumes(c2t3, mesh_traits,
    facets_criteria,
    tets_criteria, 0.5);
Output Volume Mesh

Visible human
Work in Progress
Piecewise Smooth Surfaces
Input: Piecewise smooth complex corner
## More Delaunay Filtering

<table>
<thead>
<tr>
<th>primitive</th>
<th>dual of</th>
<th>test</th>
<th>against</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voronoi vertex</td>
<td>tetrahedron</td>
<td>inside</td>
<td>domain</td>
</tr>
<tr>
<td>Voronoi edge</td>
<td>facet</td>
<td>intersect</td>
<td>domain boundary</td>
</tr>
<tr>
<td>Voronoi face</td>
<td>edge</td>
<td>intersect</td>
<td>crease</td>
</tr>
</tbody>
</table>

![Diagram](image)
Delaunay Refinement

- Steiner points
Summary
Summary

- From triangulation to quality meshes
- Mesh generation:
  - 2D: *Preserves* constraints exactly.
Summary

- From triangulation to quality meshes
- Mesh generation:
  - 2D: **Preserves** constraints exactly.
  - 3D: **Interpolates** boundary and sharp creases.
Summary

• From triangulation to quality meshes
• Mesh generation:
  – 2D: Preserves constraints exactly.
  – 3D:
    • Interpolates boundary and sharp creases.
    • Versatile through oracle-based design
See Also

**DelPSC software**
(based on CGAL)

[Dey-Levine]