Outline

1 CGAL
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
Outline

1. Cgal
   - Introduction
     - Content
     - Literature

2. Extra
   - Art Gallery
   - Lego Decomposition
   - Minkowski Sums
   - Assembly Partitioning
   - Motion Planning
   - Literature
CGAL: Mission

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

CGAL Project Proposal, 1996
Some of CGAL Content

- Bounding Volumes
- Polyhedral Surfaces
- Boolean Operations
- Triangulations
- Voronoi Diagrams
- Mesh Generation
- Subdivision
- Simplification
- Parametrisation
- Streamlines
- Ridge Detection
- Neighbor Search
- Kinetic Data Structures
- Envelopes
- Arrangements
- Intersection Detection
- Minkowski Sums
- PCA
- Polytope Distance
- QP Solver
Some CGAL Commercial Users
**Cgal Facts**

- Written in C++
- Follows the *generic programming* paradigm
- Development started in 1995
- Active European sites:

1. GeometryFactory
2. INRIA Sophia Antipolis
3. INRIA Nancy - Grand Est
4. INRIA Saclay - Île de France
5. CNRS - LIRIS
6. CNRS - Université Paris–Dauphin
7. Tel Aviv University
8. Assembrix
9. MPII Saarbrücken
10. ETH Zürich
11. Universidade Federal de Pernambuco
12. University of Crete and FO.R.T.H.
13. University of California, Davis
14. Università della Svizzera italiana
15. Universidade Federal do Rio de Janeiro
## CGAL History

<table>
<thead>
<tr>
<th>Year</th>
<th>Version Released</th>
<th>Other Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td></td>
<td><strong>CGAL founded</strong></td>
</tr>
<tr>
<td>1998</td>
<td>July 1.1</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>Aug 2.3</td>
<td>Work continued after end of European support</td>
</tr>
<tr>
<td>2001</td>
<td>May 2.4</td>
<td><strong>Editorial Board</strong> established</td>
</tr>
<tr>
<td>2002</td>
<td>Nov 3.0</td>
<td><strong>GEOMETRY FACTORY</strong> founded</td>
</tr>
<tr>
<td>2003</td>
<td>Dec 3.1</td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td><strong>CMake</strong></td>
</tr>
<tr>
<td>2009</td>
<td>Jan 3.4, Oct 3.5</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>Mar 3.6, Oct 3.7</td>
<td><strong>Google Summer of Code (GSoC) 2010</strong></td>
</tr>
<tr>
<td>2011</td>
<td>Apr 3.8, Aug 3.9</td>
<td><strong>GSoC 2011</strong></td>
</tr>
<tr>
<td>2012</td>
<td>Mar 4.0, Oct 4.1</td>
<td><strong>GSoC 2012</strong></td>
</tr>
<tr>
<td>2013</td>
<td>Mar 4.2, Oct 4.3</td>
<td><strong>GSoC 2013, Doxygen</strong></td>
</tr>
<tr>
<td>2014</td>
<td>Apr 4.4, Oct 4.5</td>
<td><strong>GSoC 2014</strong></td>
</tr>
<tr>
<td>2015</td>
<td>Apr 4.6, Oct 4.7</td>
<td><strong>GitHub, HTML5, Main repository</strong> made public</td>
</tr>
<tr>
<td>2016</td>
<td></td>
<td>20&lt;sup&gt;th&lt;/sup&gt; anniversary</td>
</tr>
</tbody>
</table>
**CGAL in Numbers**

1,600,000 lines of C++ code
10,000 downloads per year not including Linux distributions
4,500 manual pages
3,000 subscribers to cgal-announce list
1,000 subscribers to cgal-discuss list
120 packages
60 commercial users
30 active developers
6 months release cycle
7 Google’s page rank for [www.cgal.org](http://www.cgal.org)
2 licenses: Open Source and commercial
**Cgal Properties**

- **Reliability**
  - Explicitly handles degeneracies
  - Follows the Exact Geometric Computation (EGC) paradigm

- **Flexibility**
  - Is an open library
  - Depends on other libraries (e.g., Boost, Gmp, Mpfr, Qt, & Core)
  - Has a modular structure, e.g., geometry and topology are separated
  - Is adaptable to user code
  - Is extensible, e.g., data structures can be extended

- **Ease of Use**
  - Has didactic and exhaustive Manuals
  - Follows standard concepts (e.g., C++ and STL)
  - Characterizes with a smooth learning-curve

- **Efficiency**
  - Adheres to the generic-programming paradigm
    - Polymorphism is resolved at compile time
**CGAL Structure**

**Basic Library**

Algorithms and Data Structures  
e.g., Triangulations, Surfaces, and Arrangements

**Kernel**

Elementary geometric objects  
Elementary geometric computations on them

**Support Library**

Configurations, Assertions,...

**Visualization**

Files  
I/O  
Number Types  
Generators  
...
Outline

1 Cgal
   - Introduction
   - Content
   - Literature

2 Extra
   - Art Gallery
   - Lego Decomposition
   - Minkowski Sums
   - Assembly Partitioning
   - Motion Planning
   - Literature
**CGAL Kernel Concept**

- Geometric objects of constant size.
- Geometric operations on object of constant size.

<table>
<thead>
<tr>
<th>Primitives 2D, 3D, dD</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicates</td>
</tr>
<tr>
<td>point</td>
<td>comparison</td>
</tr>
<tr>
<td>vector</td>
<td>orientation</td>
</tr>
<tr>
<td>triangle</td>
<td>containment</td>
</tr>
<tr>
<td>iso rectangle</td>
<td>. . .</td>
</tr>
<tr>
<td>circle</td>
<td>. . .</td>
</tr>
<tr>
<td>. . .</td>
<td>. . .</td>
</tr>
</tbody>
</table>
Cgal Kernel Affine Geometry

point - origin \rightarrow vector
point - point \rightarrow vector
point + vector \rightarrow point
point + point \leftarrow Illegal
midpoint(a, b) = a + 1/2 \times (b - a)
CGAL Kernel Classification

- Dimension: 2, 3, arbitrary
- Number types:
  - Ring: $+, -, \times$
  - Euclidean ring (adds integer division and gcd) (e.g., CGAL::Gmpz).
  - Field: $+, -, \times, /$ (e.g., CGAL::Gmpq).
  - Exact sign evaluation for expressions with roots (Field_with_sqr).
- Coordinate representation
  - Cartesian — requires a field number type or Euclidean ring if no constructions are performed.
  - Homogeneous — requires Euclidean ring.
- Reference counting
- Exact, Filtered
**CGAL Kernels and Number Types**

### Cartesian representation

| point | $x = \frac{hx}{hw}$ | $y = \frac{hy}{hw}$ |

### Homogeneous representation

| point | $hx$ | $hy$ | $hw$ |

### Intersection of two lines

\[
\begin{align*}
\begin{cases}
    a_1x + b_1y + c_1 = 0 \\
    a_2x + b_2y + c_2 = 0
\end{cases}
\end{align*}
\]

\[
\begin{align*}
(hx, hy, hw) = & \\
\begin{pmatrix}
    b_1 & c_1 \\
    b_2 & c_2 \\
    a_1 & b_1 \\
    a_2 & b_2
\end{pmatrix}^{-1}
& \begin{pmatrix}
a_1 & c_1 \\
a_2 & c_2
\end{pmatrix}
\end{align*}
\]

### Field operations

### Ring operations
Example: Kernels $<$NumberType$>$

- **Cartesian $<$FieldNumberType$>$**
  - typedef CGAL::Cartesian $<$Gmpq$>$ Kernel;
  - typedef CGAL::Simple_cartesian $<$double$>$ Kernel;
    - No reference-counting, inexact instantiation
- **Homogeneous $<$RingNumberType$>$**
  - typedef CGAL::Homogeneous $<$Core::BigInt$>$ Kernel;
- **d-dimensional Cartesian_d and Homogeneous_d**
- **Types + Operations**
  - Kernel::Point_2, Kernel::Segment_3
  - Kernel::Less_xy_2, Kernel::Construct_bisector_3
**CGAL Numerical Issues**

```cpp
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt(NT(2));
Kernel::Point_2 p(0,0), q(sqrt2, sqrt2);
Kernel::Circle_2 C(p, 4);
assert(C.has_on_boundary(q));
```

- OK if NT supports exact sqrt.
- **Assertion violation** otherwise.
CGAL Pre-defined Cartesian Kernels

- Support construction of points from `double` Cartesian coordinates.
- Support exact geometric predicates.
- Handle geometric constructions differently:
  - `CGAL::Exact_predicates_inexact_constructions_kernel`
    - Geometric constructions may be inexact due to round-off errors.
    - It is however more efficient and sufficient for most CGAL algorithms.
  - `CGAL::Exact_predicates_exact_constructions_kernel`
  - `CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt`
    - Its number type supports the exact square-root operation.
Cgal Special Kernels

- Filtered kernels
- 2D circular kernel
- 3D spherical kernel

Refer to Cgal’s manual for more details.
Computing the Orientation

- Imperative style

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

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

int main()
{
    Point_2 p(0,0), q(10,3), r(12,19);
    return (CGAL::orientation(q,p,r) == CGAL::LEFT_TURN) ? 0 : 1;
}
```

- Precative style

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

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point_2;
typedef Kernel::Orientation_2 Orientation_2;

int main()
{
    Kernel kernel;
    Orientation_2 orientation = kernel.orientation_2_object();

    Point_2 p(0,0), q(10,3), r(12,19);
    return (orientation(q,p,r) == CGAL::LEFT_TURN) ? 0 : 1;
}
```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/intersections.h>

typedef CGAL::Exact_predicates_inexact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point_2;
typedef Kernel::Segment_2 Segment_2;
typedef Kernel::Line_2 Line_2;

int main() {
    Point_2 p(1,1), q(2,3), r(-12,19);
    Line_2 line(p,q);
    Segment_2 seg(r,p);
    auto result = CGAL::intersection(seg, lin);
    if (result) {
        if (const Segment_2* s = boost::get<Segment_2>(&*result)) {
            // handle segment
        }
        else {
            const Point_2* p = boost::get<Point_2>(&*result);
            // handle point
        }
    }
    return 0;
}
Cgal Basic Library

- Generic data structures are parameterized with Traits
  - Separates algorithms and data structures from the geometric kernel.
- Generic algorithms are parameterized with iterator ranges
  - Decouples the algorithm from the data structure.
CGAL Components Developed at Tel Aviv University

- 2D Arrangements
- 2D Regularized Boolean Set-Operations
- 2D Minkowski Sums
- 2D Envelopes
- 3D Envelopes
- 2D Snap Rounding
- Inscribed Areas / 2D Largest empty iso rectangle
Outline

1 Cgal
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
CGAL Bibliography

A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr.
On the design of CGAL a computational geometry algorithms library.

A. Fabri and S. Pion.
A generic lazy evaluation scheme for exact geometric computations.
In 2nd Library-Centric Software Design Workshop, 2006.

M. H. Overmars.
Designing the computational geometry algorithms library CGAL.

The CGAL Project.

Efi Fogel, Ron Wein, and Dan Halperin.
CGAL Arrangements and Their Applications, A Step-by-Step Guide.
Outline

1. **CGAL**
   - Introduction
   - Content
   - Literature

2. **Extra**
   - Art Gallery
   - Lego Decomposition
   - Minkowski Sums
   - Assembly Partitioning
   - Motion Planning
   - Literature
Outline

1. **CGAL**
   - Introduction
   - Content
   - Literature

2. **Extra**
   - Art Gallery
   - Lego Decomposition
   - Minkowski Sums
   - Assembly Partitioning
   - Motion Planning
   - Literature
Application: Art Gallery

Application (Art Gallery)

Given the floor plan of an art gallery modeled as a simple polygon with $n$ vertices. Find out how many (and where) guards are needed to see the entire gallery, where each guard is stationed at a fixed point, has $360^\circ$ vision, and cannot see through the walls.

Problem posed to Vasek Chvatal by Victor Klee at a math conference in 1973. Chvatal solved it quickly with a complicated proof, which has since been simplified significantly using triangulation.
Art Gallery in $\mathbb{R}^3$

- Even $n$-vertex guards do not suffice!
- Different triangulations can have different number of tetrahedra.
- Determining whether a polyhedron requires Steiner vertices for triangulation is NP-Complete.
  - Smallest example of a polyhedron that cannot be triangulated without adding new vertices. (Schoenhardt [1928]).
- Every 3D polyhedron with $n$ vertices can be triangulated with $O(n^2)$ tetrahedra. [Cha84]

5 tetrahedra

6 tetrahedra
A picture is worth a thousand words, but an object is worth a thousand pictures.
Outline

1 Cgal
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
Model Decomposition into Lego Bricks
Outline

1 Cgal
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
Movies

Eyal Flato, Efi Fogel, Dan Halperin, and Eyal Leiserowitz.
Movie: Exact Minkowski Sums and Applications.

Efi Fogel and Dan Halperin.
Movie: Exact Minkowski sums of convex polyhedra.

Efi Fogel, Ophir Setter, and Dan Halperin.
Movie: Arrangements of Geodesic Arcs on the Sphere.
Outline

1 CGAL
   - Introduction
   - Content
   - Literature

2 Extra
   - Art Gallery
   - Lego Decomposition
   - Minkowski Sums
   - Assembly Partitioning
   - Motion Planning
   - Literature
Movable Separability
Terms & Definitions

**Assembly** — A collection of pairwise interior disjoint bodies/parts, say $A$.

**Subassembly** — A subset of parts of $A$ in their relative placements in $A$.

**Assembly Operation** — A motion that merges $s$ subassemblies of $A$ into a new subassembly; $s$ is the number of hands.

**Assembly Partitioning** — The reverse of Assembly Operation (disassembly).

**Assembly Sequence** — A total ordering of assembly operations from separated parts to the full $A$.

**Monotone Operation** — An operation that generates only subassemblies of the final $A$. 
Constraints

We will discuss assembly planning problems constrained to

- 2-handed
- Monotone operations
- Rigid parts

Hence, we can plan assembly by disassembly.
Partitioning with Convex Parts

- In $\mathbb{R}^2$
  - Admit a disassembly sequence translating one part at a time along a fixed (arbitrary) direction to infinity.

- In $\mathbb{R}^3$
  - Depth order does not always exist.
  - Moreover, assemblies of convex parts may be interlocked.

[SS93]
2-Handed Assembly by Disassembly

- **Input:** an assembly \( A \), the allowable motions.
- **Output:** an assembly sequence.

- **Algorithm:**
  - Partition \( A \), and then the two subassemblies recursively, or stop and announce failure.
  - Reverse the disassembly motions into an assembly sequence.
Polyhedral Assembly Partitioning with Infinite Translations

**Input:** \( n \) pairwise interior disjoint polytopes in \( \mathbb{R}^3 \)
\[ A = \{ P_1, P_2, \ldots, P_n \} \]

**Output:** A proper subset \( S \subseteq A \) and a direction \( \vec{d} \) in \( \mathbb{R}^3 \),

- \( S \) can be translated as a rigid body to infinity along \( \vec{d} \) without colliding with \( A \setminus S \)
- Sliding motions of parts over other parts are allowed
Outline

1 Cgal
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
Motion Planning: A Translating Polygonal Robot

Application (A Translating Polygonal Robot)

Given a simple-polygon robot $Q$, which can translate (but not rotate) in a room cluttered with pairwise interior-disjoint polygonal obstacles, devise a data structure that can efficiently answer queries of the following form: Given a start position $s$ of some reference point in $Q$ and a goal position $g$ of the same reference point, plan a collision-free path of the robot from $s$ to $g$. 
The workspace:
- a diamond-shaped robot translating in a house amidst polygonal obstacles

The configuration space:
- the forbidden-configuration space,
- the free-configuration space decomposed into trapezoidal faces
- the queries, and the resulting paths, if exist.
Outline

1 Cgal
   • Introduction
   • Content
   • Literature

2 Extra
   • Art Gallery
   • Lego Decomposition
   • Minkowski Sums
   • Assembly Partitioning
   • Motion Planning
   • Literature
Triangulation Bibliography I

B. Chazelle.
Triangulating a Simple Polygon in Linear Time.

Triangulating a Simple Polygon.

An $O(n \log \log(n))$ Time Algorithm for Triangulating a Simple Polygon.

Mark de Berg, Mark van Kreveld, Mark H. Overmars, and Otfried Cheong.
*Computational Geometry: Algorithms and Applications*.  

Alexey V. Skvortsov, Yuri L. Kostyuk  
Efficient algorithms for Delaunay triangulation.

Bernard Chazelle.
Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm.

Michael J. Laszlo.
*Computational Geometry and Computer Graphics in C++*.  
Prentice-Hall, 1996
组装分割参考文献 I

Jack Snoeyink and Jorge Stolfi,
Objects that cannot be taken apart with two hands,

Randall H. Wilson and Jean-Claude Latombe,
Geometric reasoning about mechanical assembly,

Lydia E. Kavraki and Mihail N. Kolountzakis,
Partitioning a planar assembly into two connected parts is NP-complete,

Randall H. Wilson, Lydia E. Kavraki, Tomás Lozano-Pérez, and Jean-Claude Latombe,
Two-handed assembly sequencing,

Sanjeev Khanna, Rajeev Motwani, and Randall H. Wilson,
On certificates and lookahead in dynamic graph problems,

Dan Halperin, Jean-Claude Latombe, and Randall H. Wilson,
A general framework for assembly planning: The motion space approach,

Efi Fogel and Dan Halperin,
Polyhedral Assembly Partitioning with Infinite Translations or The Importance of Being Exact,

and more ...