Algorithms for 3D Printing and Other Manufacturing Methodologies

Efi Fogel

Tel Aviv University

CGAL
Apr. 3rd, 2017
Outline

1. CGAL
   - Introduction
   - Content
   - Literature
   - Details
Outline

1 CGAL
   • Introduction
   • Content
   • Literature
   • Details
**CGAL: Mission**

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

*CGAL* Project Proposal, 1996
CGAL Facts

- Written in C++
- Adheres the generic programming paradigm
- Development started in 1995
- Several active contributor sites
- High search-engine ranking for www.cgal.org

- Used in a diverse range of domains
  - e.g., computer graphics, scientific visualization, computer aided design and modeling, additive manufacturing, geographic information systems, molecular biology, medical imaging, and VLSI
- The de-facto standard in applied Computational Geometry
Cgal in Numbers

600,000 lines of C++ code
10,000 downloads per year not including Linux distributions
4,500 manual pages (user and reference manual)
1,000 subscribers to user mailing list
200 commercial users
120 packages
30 active developers
6 months release cycle
2 licenses: Open Source and commercial
## 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>CGAL founded</td>
</tr>
<tr>
<td>1998</td>
<td>July 1.1</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td></td>
<td>Work continued after end of European support</td>
</tr>
<tr>
<td>2001</td>
<td>Aug 2.3</td>
<td>Editorial Board established</td>
</tr>
<tr>
<td>2002</td>
<td>May 2.4</td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>Nov 3.0</td>
<td>GEOMETRY FACTORY founded</td>
</tr>
<tr>
<td>2004</td>
<td>Dec 3.1</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>May 3.2</td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>Jun 3.3</td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td>CMake</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>Google Summer of Code (GSoC) 2010</td>
</tr>
<tr>
<td>2011</td>
<td>Apr 3.8, Aug 3.9</td>
<td>GSoC 2011</td>
</tr>
<tr>
<td>2012</td>
<td>Mar 4.0, Oct 4.1</td>
<td>GSoC 2012</td>
</tr>
<tr>
<td>2013</td>
<td>Mar 4.2, Oct 4.3</td>
<td>GSoC 2013, Doxygen</td>
</tr>
<tr>
<td>2014</td>
<td>Apr 4.4, Oct 4.5</td>
<td>GSoC 2014</td>
</tr>
<tr>
<td>2015</td>
<td>Apr 4.6, Oct 4.7</td>
<td>GitHub, HTML5, Main repository made public</td>
</tr>
<tr>
<td>2016</td>
<td>Apr 4.8, Sep 4.9</td>
<td>20^{th} anniversary</td>
</tr>
<tr>
<td>2017</td>
<td></td>
<td>GSoC 2017</td>
</tr>
</tbody>
</table>
**CGAL Properties**

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

- **Efficiency**
  - Depends on leading 3rd party libraries
    - e.g., *Boost*, *Gmp*, *Mpfr*, *Qt*, *Eigen*, *Tbb*, and *Core*
  - Adheres to the generic-programming paradigm
    - Polymorphism is resolved at compile time

→ The best of both worlds ←
CGAL Properties, Cont

- **Flexibility**
  - Adaptable, e.g., graph algorithms can directly be applied to CGAL data structures
  - Extensible, e.g., data structures can be extended

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

1 CGAL
  • Introduction
  • Content
  • Literature
  • Details
2D Algorithms and Data Structures

- Triangulations
- Mesh Generation
- Polyline Simplification
- Voronoi Diagrams
- Arrangements
- Boolean Operations
- Neighborhood Queries
- Minkowski Sums
- Straight Skeleton
3D Algorithms and Data Structures

- Triangulations
- Mesh Generation
- Polyhedral Surface
- Deformation
- Boolean Operations
- Mesh Simplification
- Skeleton
- Segmentation
- Classification
- Hole Filling
Outline

1 CGAL
   • Introduction
   • Content
   • Literature
   • Details
Cgal Bibliography I

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

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Levy.  
Polygon Mesh Processing.  

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.  

Many Many Many papers
Outline

1 CGAL
- Introduction
- Content
- Literature
- Details
**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
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 Envelopes
- Inscribed Areas / 2D Largest empty iso rectangle
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
**CGAL 2D Arrangements**

- The main data structure

  ```cpp
  template <typename Traits , typename Dcel> Arrangement_2 { ... } 
  ```

**Traits** definitions of geometric elements
- geometric-object types e.g., `Point_2`, and
- operations on objects of these types, e.g., `Compare_xy_2`.

**Dcel** definitions of topological elements
- topological-object types, e.g., `vertex`, `halfedge`, and `face`, and
- operations required to maintain the incidence relations among objects of these types.

- A traits class for line segments.

  ```cpp
  Arr_non_cached_segment_traits_2<Kernel> : public Kernel { ... } 
  ```

  - All object types and most operations are inherited from the derived kernel.
Two Dimensional Arrangements

Definition (Arrangement)

Given a collection $\mathcal{C}$ of curves on a surface, the arrangement $A(\mathcal{C})$ is the partition of the surface into vertices, edges and faces induced by the curves of $\mathcal{C}$.

An arrangement of circles in the plane.

An arrangement of lines in the plane.

An arrangement of great-circle arcs on a sphere.
**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**

<table>
<thead>
<tr>
<th>Cartesian representation</th>
<th>Homogeneous representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>$x = \frac{hx}{hw}$, $y = \frac{hy}{hw}$</td>
</tr>
</tbody>
</table>

Intersection of two lines

\[
\begin{align*}
\begin{cases}
    a_1 x + b_1 y + c_1 = 0 \\
    a_2 x + b_2 y + c_2 = 0
\end{cases}
\end{align*}
\]

\[
(x, y) = \left( \frac{b_1 c_1 - a_1 c_2}{\det(a_1, b_1, a_2, b_2)}, \frac{a_1 b_1 - a_2 c_1}{\det(a_1, b_1, a_2, b_2)} \right)
\]

Field operations

\[
(hx, hy, hw) = \left( \frac{b_1 c_1 - a_1 c_2}{\det(a_1, b_1, a_2, b_2)}, \frac{a_1 b_1 - a_2 c_1}{\det(a_1, b_1, a_2, b_2)}, -\frac{a_1 b_1 + a_2 c_1}{\det(a_1, b_1, a_2, b_2)} \right)
\]

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
#if 1
typedef CORE::Expr NT;
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = CGAL::sqrt(NT(2));
#else
typedef double NT;
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt(2);
#endif

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;
}
```
CGAL Adaptable & Extensible Kernel

Geometric class templates are parameterized with the kernel

```cpp
template <typename K> struct MyPoint { ... };
template <typename K> struct MyLine { ... };
template <typename K> struct MyConstruct { ... };
```

Geometric class definitions are nested in the kernel

```cpp
struct Kernel {
    typedef MyPoint<Kernel> Point_2;
    typedef MyLine<Kernel> Line_2;
    typedef MyConstruct<Kernel> Construct_line_2;
};
```

Injecting a class into its nested templates is not a problem, but there is more...
CGAL Adaptable & Extensible Kernel, Cont

We want to define a new kernel where new types can be added and existing ones can be exchanged

```cpp
struct New_kernel : public Kernel {
    typedef NewPoint<New_kernel> Point_2;
    typedef MyLeftTurn<New_kernel> Left_turn_2;
};
```

- Problem: The inherited class MyConstruct is still parameterized with Kernel, hence it operates on the old point class MyPoint.
- Solutions
  - Redefine Construct_line_2 in New_kernel
We want to define a new kernel where new types can be added and existing ones can be exchanged

```cpp
struct New_kernel : public Kernel {
    typedef NewPoint<New_kernel> Point_2;
    typedef MyLeftTurn<New_kernel> Left_turn_2;
};
```

- **Problem:** The inherited class MyConstruct is still parameterized with Kernel, hence it operates on the old point class MyPoint.
- **Solutions**
  - Redefine Construct_line_2 in New_kernel
We want to define a new kernel where new types can be added and existing ones can be exchanged.

```cpp
struct New_kernel : public Kernel {
    typedef NewPoint<New_kernel> Point_2;
    typedef MyLeftTurn<New_kernel> Left_turn_2;
};
```

- **Problem:** The inherited class `MyConstruct` is still parameterized with `Kernel`, hence it operates on the old point class `MyPoint`.
- **Solutions**
  - Redefine `Construct_line_2` in `New_kernel`
  - Defer the instantiation of `Construct_line_2`
Defer instantiation once again to be able to extend `New_kernel` in the same way as `Kernel`.

```cpp
template <typename K>
struct New_kernel_base : public Kernel_base<K> {
    typedef NewPoint<K> Point_2;
    typedef MyLeftTurn<K> Left_turn_2;
};
struct New_kernel : public New_kernel_base<New_kernel> {};
```cpp
#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, line);
    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;
}
```