Introduction to the
Computational Geometry Algorithms Library

Monique Teillaud

CGAL

www.cgal.org

october 2008
Overview

- The CGAL Open Source Project
- Structure of CGAL
- The Kernel
Part I

The CGAL Open Source Project
Goals

- Promote the research in Computational Geometry (CG)
- "make the large body of geometric algorithms developed in the field of CG available for industrial applications"

⇒ robust programs
Goals

- Promote the research in Computational Geometry (CG)
- “make the large body of geometric algorithms developed in the field of CG available for industrial applications”

⇒ robust programs

CG Impact Task Force Report, 1996
Among the key recommendations:
- Production and distribution of usable (and useful) geometric codes
- Reward structure for implementations in academia
History

Development started in 1995
Consortium of 8 European sites
Two ESPRIT LTR European Projects (1996-1999)
History

Development started in 1995
Consortium of 8 European sites
Two ESPRIT LTR European Projects (1996-1999)

Utrecht University (Plageo)
INRIA Sophia Antipolis (C++GAL)
ETH Zürich (XYZ Geobench)
MPI Saarbrücken (LEDA)
Tel Aviv University
Freie Universität Berlin
  RISC Linz
  Martin-Luther-Universität Halle
History

- Work continued after the end of Galia (1999) in several sites.
  - partial support of ECG, ACS, Aim@Shape

- January, 2003: creation of **GEOMETRY FACTORY**
  INRIA startup
  sells commercial licenses, support, customized developments

- November, 2003: Release 3.0 - **Open Source Project**

- June, 2007: Release 3.3
- soon: Release 3.4
License

- a few basic packages under LGPL
- most packages under QPL
  - free use for Open Source code
  - commercial license needed otherwise

- A guarantee for CGAL users
- Allows CGAL to become a “standard”
- Opens CGAL for new contributions
CGAL in numbers

- 500,000 lines of C++ code
- 3,500 pages manual
- 120 packages
CGAL in numbers

- 500,000 lines of C++ code
- 3,500 pages manual
- 120 packages
- release cycle of \( \sim 12 \) months
- \( \sim 1,000 \) download per month
- several platforms
  - g++ (Linux MacOS Windows)
  - VC++
CGAL in numbers

- 500,000 lines of C++ code
- 3,500 pages manual
- 120 packages
- release cycle of \sim 12 months
- \sim 1,000 download per month
- several platforms
  - g++ (Linux MacOS Windows)
  - VC++
- 4,000 subscribers to announcement list (7,000 for gcc)
- 1,000 subscribers to discussion list (600 in gcc-help)
- 50 developers registered on developer list (20 active)
Development process

**Editorial Board** created in 2001.

- responsible for the **quality** of CGAL

  New packages are **reviewed**.

→ helps authors to get **credit** for their work.

*CG Impact Task Force Report, 1996*

Reward structure for implementations in academia

- decides about technical matters
- coordinates communication and promotion
- ...

Development process

Editorial Board created in 2001.
Editorial Board created in 2001.

Pierre Alliez (INRIA Sophia Antipolis - Méditerranée)
Eric Berberich (Max-Planck-Institut für Informatik)
Andreas Fabri (GEOMETRY FACTORY)
Efi Fogel (Tel Aviv University)
Bernd Gärtner (ETH Zürich)
Michael Hemmer (Max-Planck-Institut für Informatik)
Michael Hoffmann (ETH Zürich)
Menelaos Karavelas (Univ Crete)
Sylvain Pion (INRIA Sophia Antipolis - Méditerranée)
Marc Pouget (INRIA Nancy - Grand Est)
Laurent Rineau (GEOMETRY FACTORY)
Monique Teillaud (INRIA Sophia Antipolis - Méditerranée)
Ron Wein (Tel Aviv University)
Mariette Yvinec (INRIA Sophia Antipolis - Méditerranée)
Development tools

- Own manual tools: \LaTeX \rightarrow ps, pdf, html
- svn server (INRIA gforge) for version management
Development tools

- Own manual tools: \LaTeX \rightarrow \text{ps, pdf, html}
- svn server (INRIA gforge) for version management
- Developer manual
- Mailing list for developers
- 1-2 developers meetings per year, 1 week long
- Automatic test suites running on all supported compilers/platforms
Development tools

- Own manual tools: \LaTeX \rightarrow ps, pdf, html
- svn server (INRIA gforge) for version management
- Developer manual
- Mailing list for developers
- 1-2 developers meetings per year, 1 week long
- 1 internal release per day
- Automatic test suites running on all supported compilers/platforms
Contributors keep their identity
Credit

Contributors keep their identity

- **Names of authors** appear at the beginning of each chapter. Section on history of the package at the end of each chapter, with names of all contributors.
- CGAL developers listed on the “People” web page.
Contributors keep their identity

- **Names of authors** appear at the beginning of each chapter. Section on history of the package at the end of each chapter, with names of all contributors.
- CGAL developers listed on the “People” web page.
- Authors publish **papers** (conferences, journals) on their packages.
Credit

Contributors keep their identity

- **Names of authors** appear at the beginning of each chapter. Section on history of the package at the end of each chapter, with names of all contributors.
- CGAL developers listed on the “People” web page.

- Authors publish **papers** (conferences, journals) on their packages.

- **Copyright** kept by the institution of the authors.
Contributors

[Map with logos of institutions including Stanford University, University of Notre Dame, University of Florida, INRIA, and others.]

SciSoft

Geometry Factory
Users

Long list of identified users
(see web site)

More non-identified users...
Customers of GEOMETRY FACTORY
Part II

Contents of CGAL
# Contents

- **Bounding Volumes**
  - Triangulations
  - Voronoi Diagrams

- **Polyhedral Surface**
  - Boolean Operations
  - Mesh Generation

- **Subdivision**
  - Simplification
  - Parameterization

- **Streamlines**
  - Ridge Detection
  - Neighbour Search

- **Lower Envelope**
  - Arrangement
  - Intersection Detection

- **Minkowski Sum**
  - PCA
  - Polytope distance

- **QP Solver**

---
Structure

- Kernels
- Various packages
- Support Library
  
  STL extensions, I/O, generators, timers...
Part III

The CGAL Kernels
The CGAL Kernels

- 2D, 3D, dD “Rational” kernels
- 2D circular kernel
- 3D spherical kernel (to appear)
In the kernels

- Elementary geometric objects
- Elementary computations on them

Primitives
- 2D, 3D, dD
- Point
- Vector
- Triangle
- Circle

Predicates
- comparison
- Orientation
- InSphere

Constructions
- intersection
- squared distance
Affine geometry

Point - Origin $\rightarrow$ Vector
Point - Point $\rightarrow$ Vector
Point + Vector $\rightarrow$ Point

Point + Point illegal

midpoint(a,b) = a + 1/2 x (b-a)
<table>
<thead>
<tr>
<th>Kernels and number types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cartesian representation</strong></td>
</tr>
<tr>
<td>Point</td>
</tr>
<tr>
<td>( x = \frac{hx}{hw} )</td>
</tr>
<tr>
<td>( y = \frac{hy}{hw} )</td>
</tr>
</tbody>
</table>
# Kernels and number types

## Cartesian representation

<table>
<thead>
<tr>
<th>Point</th>
<th>( x = \frac{hx}{hw} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( y = \frac{hy}{hw} )</td>
</tr>
</tbody>
</table>

## Homogeneous representation

<table>
<thead>
<tr>
<th>Point</th>
<th>( hx )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( hy )</td>
</tr>
<tr>
<td></td>
<td>( hw )</td>
</tr>
</tbody>
</table>

**- ex: Intersection of two lines -**

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

\[
(x, y) = \frac{1}{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}} \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} - \frac{1}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}
\]

\[
(\text{hx, hy, hw}) = \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} - \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}
\]

Field operations

Ring operations
Kernels and number types

**Cartesian representation**

<table>
<thead>
<tr>
<th>Point</th>
<th>$x = \frac{hx}{hw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$y = \frac{hy}{hw}$</td>
</tr>
</tbody>
</table>

- ex: Intersection of two lines -

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

\[
(x, y) = \begin{pmatrix}
  b_1 & c_1 \\
  b_2 & c_2
\end{pmatrix}
\begin{pmatrix}
  a_1 & c_1 \\
  a_2 & c_2
\end{pmatrix}
\begin{pmatrix}
  a_1 & b_1 \\
  a_2 & b_2
\end{pmatrix}
\]

**Homogeneous representation**

<table>
<thead>
<tr>
<th>Point</th>
<th>$hx$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$hy$</td>
</tr>
<tr>
<td></td>
<td>$hw$</td>
</tr>
</tbody>
</table>

\[
\begin{cases}
  a_1 hx + b_1 hy + c_1 hw = 0 \\
  a_2 hx + b_2 hy + c_2 hw = 0
\end{cases}
\]

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

**Field operations**

**Ring operations**
C++ Templates

CGAL::Cartesian<FT>
CGAL::Homogeneous<RT>

(CGAL::Simple_Cartesian)
(CGAL::Simple_Homogen.)
C++ Templates

**CGAL::Cartesian**< FT >
**CGAL::Homogeneous**< RT >

**Cartesian Kernels:**
- Field type
  - double
  - Quotient<Gmpz>
  - leda_real

**Homogeneous Kernels:**
- Ring type
  - int
  - Gmpz
  - double

-----

(CGAL::Simple_Cartesian)
(CGAL::Simple_Homogenous)
C++ Templates

CGAL::Cartesian< FT >
CGAL::Homogeneous< RT >

(CGAL::Simple_Cartesian)
(CGAL::Simple_Homogen.)

Cartesian Kernels:
Field type
- double
- Quotient<Gmpz>
- leda_real

Homogeneous Kernels:
Ring type
- int
- Gmpz
- double

→ Flexibility
typedef double NumberType;
typedef Cartesian< NumberType > Kernel;
typedef Kernel::Point_2 Point;
Numerical robustness issues

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

OK if NT gives exact sqrt
assertion violation otherwise
Numerical robustness issues

Orientation of 2D points

orientation\((p, q, r)\) = \text{sign}\left(\det\begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix}\right)

= \text{sign}\left((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)\right)
Numerical robustness issues

Orientation of 2D points

$$p = (0.5 + x.u, 0.5 + y.u)$$
$$0 \leq x, y < 256, \quad u = 2^{-53}$$
$$q = (12, 12)$$
$$r = (24, 24)$$

orientation($p, q, r$)
evaluated with double

256 x 256 pixel image

\[> 0, \quad = 0, \quad < 0\]

\[\rightarrow\text{inconsistencies}\text{ in predicate evaluations}\]
Numerical robustness issues

solved in CGAL using

Exact Geometric Computation
Speed and exactness
Numerical robustness issues

solved in CGAL using

Exact Geometric Computation
Speed and exactness

≠

exact arithmetics
More number types

- Detailed hierarchy of **algebraic** and **arithmetic concepts** and **classes**
Circular/spherical kernels

- solve needs for e.g. intersection of circles.
- extend the CGAL (linear) kernels
The circular/spherical kernels

Circular/spherical kernels
- solve needs for e.g. intersection of circles.
- extend the CGAL (linear) kernels

Guidelines
- **code reuse:**
  - ability to *reuse the CGAL kernel* for points, circles, number types,…
- **flexibility:**
  - possibility to *use other implementations* for points, circles, number types,…
  - possibility to *use several algebraic implementations*
The circular/spherical kernels

Circular/spherical kernels

• solve needs for e.g. intersection of circles.
• extend the CGAL (linear) kernels

Guidelines

• code reuse:
  • ability to reuse the CGAL kernel for points, circles, number types,
  • flexibility:
    • possibility to use other implementations for points, circles, number types,
    • possibility to use several algebraic implementations

```cpp
template < LinearKernel, AlgebraicKernel >
class Circular_kernel : public LinearKernel
```
2D circular kernel design

template < LinearKernel, AlgebraicKernel > 
class Circular_kernel

Types
- Must be defined by Linear_kernel
  basic number types, points, lines,...
- Must be defined by Algebraic_kernel
  algebraic numbers, polynomials
- Defined by Circular_kernel
  Circular_arc_2, Circular_arc_point_2
2D circular kernel design

template < LinearKernel, AlgebraicKernel >
class Circular_kernel

Types
- Must be defined by Linear_kernel
  basic number types, points, lines, ...
- Must be defined by Algebraic_kernel
  algebraic numbers, polynomials
- Defined by Circular_kernel
  Circular_arc_2, Circular_arc_point_2

Predicates
e.g. intersection tests, comparisons of intersection points, ...
  exactness is crucial for geometric algorithms

Constructions
e.g. computation of intersection points
Representation

- **CGAL Circle_2**:  
  - center  
  - squared radius (rational)
Representation

- **CGAL Circle_2**:  
  - center  
  - squared radius (rational)

- **Circular_arc_2**:  
  - supporting circle Circle_2  
  - 2 Circular_arc_point_2 (algebraic)

- **Circular_arc_point_2**  
  - root of system  
  (system = 2 equations of circles) (algebraic)
Number types

For linear objects: \textbf{RT} or \textbf{FT} ring or field type (\(+\), \(-\), \(\times\), \(/\))

For circles: \texttt{Root\_of\_2< RT >} (<, =, >)

Exact computations on algebraic numbers of degree 2 (not a field!!!)

Polynomial representation of \texttt{Root\_of\_2< RT >}:
3 coefficients \texttt{RT} + 1 boolean

Sturm sequences, resultants, Descartes’ rule,… reduce \textbf{comparisons} to computations of \textbf{signs of polynomial expressions}
Application

Computation of arrangements of 2D circular arcs and line segments
Application of the 3D spherical kernel

Computation of arrangements of 3D spheres
Part IV

Flexibility
"Traits" classes

```cpp
convex_hull_2<InputIt., OutputIt., Traits>
Polygon_2<Traits, Container>
Polyhedron_3<Traits, HDS>
Triangulation_3<Traits, TDS>
...
```
“Traits” classes

convex_hull_2<InputIt., OutputIt., Traits>
Polygon_2<Traits, Container>
Polyhedron_3<Traits, HDS>
Triangulation_3<Traits, TDS>
...

Geometric traits classes provide:
  Geometric objects + predicates + constructors

- The **Kernel** can be used as a traits class for several algorithms
- Otherwise: **Default traits classes** provided
- The **user** can plug his own traits class
Playing with traits classes

Delaunay Triangulation

Requirements for a traits class:
- Point
- orientation test, in_circle test

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2< K > Delaunay;
Delaunay Triangulation

- 3D points: coordinates \((x, y, z)\)
- orientation, in_circle: on \(x\) and \(y\) coordinates

```cpp
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Triangulation_euclidean_traits_xy_3<K> Traits;
typedef CGAL::Delaunay_triangulation_2<Traits> Terrain;
```
More flexibility

The user can add information in vertices and cells

...
To know more

www.cgal.org