Overview

• The CGAL Open Source Project
• Structure of CGAL
• The Kernel
• Numerical Robustness
• Contents of the Basic Library
• Flexibility
• Work in Progress
The COAL 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

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
Development started in 1995

Consortium of 8 European sites
Two ESPRIT LTR European Projects (1996-1999)

Utrecht University (XYZ Geobench)
INRIA Sophia Antipolis (C++GAL)
ETH Zürich (Plageo)
MPI Saarbrücken (LEDA)
Tel Aviv University
Freie Universität Berlin
RISC Linz
Martin-Luther-Universität Halle
• Work continued after the end of European support (1999) in several sites.

• January, 2003: creation of Geometry Factory

INRIA startup
sells commercial licenses, support, customized developments

• November, 2003:

Release 3.0
Open Source Project

• December, 2004: Release 3.1
License

- *kernel* under LGPL

- *basic library* 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
in numbers

- 350,000 lines of C++ code
- ~2000 pages manual
- release cycle of ~12 months
- CGAL 2.4: 9300 downloads (18 months)
- CGAL 3.1: 7329 downloads (9 months)
- 4000 subscribers to the announcement list (7000 for gcc)
- 800 users registered on discussion list (600 in gcc-help)
- 50 developers registered on developer list

Introduction to CGAL - updated november 2005
Supported platforms

- Linux, Irix, Solaris, Windows, Mac OS X
- g++, SGI CC, SunProCC, VC7, Intel
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
- ...

Introduction to CGAL - updated november 2005
Andreas Fabri (Geometry Factory)
Efi Fogel (Tel Aviv University)
Bernd Gärtner (ETH Zürich)
Michael Hoffmann (ETH Zürich)
Menelaos Karavelas (University of Notre Dame, USA → Greece)
Lutz Kettner (Max-Planck-Institut für Informatik)
Sylvain Pion (INRIA Sophia Antipolis)
Monique Teillaud (INRIA Sophia Antipolis)
Remco Veltkamp (Utrecht University)
Ron Wein (Tel Aviv University)
Mariette Yvinec (INRIA Sophia Antipolis)
Tools

- Own manual tools: \LaTeX \rightarrow \text{ps, pdf, html}
- CVS server 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

- up to 3.0.1: names of authors mentioned in the Preface.

- 3.1: **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.
Projects using CGAL

Leonidas J. Guibas' and co-workers, Stanford University.
Tamal K. Dey’s and co-workers, The Ohio State University.
Nina Amenta and co-workers, The University of Texas at Austin.
Xiangmin Jiao, University of Illinois at Urbana-Champaign.
(Surface Mesh Overlay)
Peter Coveney and co-workers, University of London.

Teaching

- Leo Guibas, Siu Wing Cheng, . . .
Commercial customers of Geometry Factory
Structure of Coal
The COAL Kernel
In the kernel

Elementary geometric objects

Elementary computations on them

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

**Predicates**
- comparison
- Orientation
- InSphere

**Constructions**
- intersection
- squared distance

...
Affine geometry

- Point - Origin → Vector
- Point - Point → Vector
- Point + Vector → Point

Point + Point illegal

midpoint(a, b) = a + 1/2 x (b-a)
**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**

\[
\begin{align*}
\left\{ \begin{array}{l}
    a_1x + b_1y + c_1 = 0 \\
    a_2x + b_2y + c_2 = 0
\end{array} \right. \\
(x, y) = \left( \begin{array}{c}
    b_1 & c_1 \\
    b_2 & c_2 \\
    a_1 & b_1 \\
    a_2 & b_2
\end{array} \right)^{-1} \left( \begin{array}{c}
    a_1 & c_1 \\
    a_2 & c_2 \\
\end{array} \right)^{-1}
\end{align*}
\]

**Field operations**

**Ring operations**

\[
(hx, hy, hw) = \left( \begin{array}{ccc}
    b_1 & c_1 \\
    b_2 & c_2 \\
\end{array} \right)^{-1} \left( \begin{array}{ccc}
    a_1 & c_1 \\
    a_2 & c_2 \\
\end{array} \right)^{-1} \left( \begin{array}{ccc}
    a_1 & b_1 \\
    a_2 & b_2 \\
\end{array} \right)
\]

*Introduction to CGAL - updated November 2005*
C++ Templates

\[
\begin{align*}
\text{CGAL::} & \textbf{Cartesian}<\ \text{FT} \ > & \quad \text{(CGAL:: Simple\_Cartesian)} \\
\text{CGAL::} & \textbf{Homogeneous}<\ \text{RT} \ > & \quad \text{(CGAL:: Simple\_Homogeneous)}
\end{align*}
\]

Cartesian Kernels: Field type
- double
- \texttt{Quotient\textless Gmpz\textgreater}
- \texttt{leda\_real}

Homogeneous Kernels: Ring type
- int
- \texttt{Gmpz}
- double

\[\rightarrow \text{Flexibility}\]

```cpp
typedef double NumberType;
typedef \textbf{Cartesian}<\ \texttt{NumberType} \ > \texttt{Kernel};
typedef \texttt{Kernel::Point\_2} \texttt{Point};
```
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 \texttt{sqrt}
assertion violation otherwise
Orientation of 2D points

\[
\text{orientation}(p, q, r) = \text{sign} \left( \det \begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix} \right) = \text{sign}\left((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)\right)
\]
\[ p = (0.5 + x.u, 0.5 + y.u) \]
\[ 0 \leq x, y < 256, \quad u = 2^{-53} \]
\[ q = (12, 12) \]
\[ r = (24, 24) \]

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

256 x 256 pixel image

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

\[ \longrightarrow \text{inconsistencies in predicate evaluations} \]

[Kettner, Mehlhorn, Pion, Schirra, Yap, ESA’04]
Numerical Robustness in
imprecise numerical evaluations $\rightarrow$ non-robustness

combinatorial result

Exact Geometric Computation $\neq$

exact arithmetics
Predicates and Constructions

Input

Predicates

< 0  = 0  > 0

Combinatorial Structure

Constructions

Geometric embedding
Delaunay triangulation

only **predicates** are used

*orientation, in_sphere*

Voronoi diagram

**constructions** are needed

*circumcenter*
• **Multiprecision integers**
  Exact evaluation of signs / values of polynomial expressions with integer coefficients
  CGAL::MP_Float, GMP::mpz_t, LEDA::integer, ...

• **Multiprecision floats**
  idem, with float coefficients \((n2^m, n, m \in \mathbb{Z})\)
  CGAL::MP_Float, GMP::mpf_t, LEDA::bigfloat, ...

• **Multiprecision rationals**
  Exact evaluation of signs / values of rational expressions
  CGAL::Quotient<·>, GMP::mpq_t, LEDA::rational, ...

• **Algebraic numbers**
  Exact comparison of roots of polynomials
  LEDA::real, Core::Expr (work in progress in CGAL)
Filtering Predicates

\[ \text{sign} \ (P(x)) \ ? \]

Approximate evaluation \( P^a(x) \) + Error \( \varepsilon \)

\[ |P^a(x)| > \varepsilon \]

\( \text{y} \) \n\( \text{n} \)

\[ \text{sign} \ (P(x)) = \text{sign} \ (P^a(x)) \]

Exact computation
Static filtering

Error bound precomputed faster

Dynamic filtering

Interval arithmetic more precise

Number types: CGAL::Interval_nt, MPFR/MPFI, boost::interval

CGAL::Filtered_kernel $< K >$ kernel wrapper [Pion]

Replaces predicates of $K$ by filtered and exact predicates. (exact predicates computed with MP_Float)

Static + Dynamic filtering in CGAL 3.1

$\rightarrow$ more generic generator also available for user’s predicates
Filtering Constructions

Number type `CGAL::Lazy_exact_nt < Exact_NT >` [Pion]

Delays exact evaluation with `Exact_NT`:

- stores a `DAG` of the expression
- computes first an approximation with `Interval_nt`
- allows to control the relative precision of `to_double`

`CGAL::Lazy_kernel` in CGAL 3.2
Predefined kernels

**Exact_predicates_exact_constructions_kernel**
Filtered_kernel< Cartesian< Lazy_exact_nt< Quotient< MP_Float >>>

**Exact_predicates_exact_constructions_kernel_with_sqrt**
Filtered_kernel< Cartesian< Core::Expr >>

**Exact_predicates_inexact_constructions_kernel**
Filtered_kernel< Cartesian< double >>>
**Efficiency**

### 3D Delaunay triangulation

CGAL-3.1-I-124

1.000.000 random points
- Simple\_Cartesian\(<\) double \(>\) \hspace{1cm} 48.1 sec
- Simple\_Cartesian\(<\) MP\_Float \(>\) \hspace{1cm} 2980.2 sec
- Filtered\_kernel (dynamic filtering) \hspace{1cm} 232.1 sec
- Filtered\_kernel (static + dynamic filtering) \hspace{1cm} 58.4 sec

49.787 points (Dassault Systèmes)
- double \hspace{1cm} loop !
- exact and filtered \hspace{1cm} < 8 sec
Robustness of Delaunay triangulations

Kernel and arithmetics $\rightarrow$ Numerical robustness

Algorithms $\rightarrow$ explicit treatment of degenerate cases

Symbolic perturbation for 3D dynamic Delaunay triangulations

[Devillers Teillaud SODA’03]
Contents of the COAL Basic Library
Convex Hull

[MPI]

- 5 different algorithms in 2D
- 3 different algorithms in 3D
Triangulations and related

[INRIA]

- 2D/3D Triangle/Tetrahedron based data-structure
- Fully dynamic 2D/3D Delaunay triangulation Delaunay hierarchy [Devillers ’98 ’02]
- 2D/3D Regular Triangulations (fully dynamic in 3.2?)
- 2D Constrained Delaunay Triangulation
- 2D Apollonius diagram
- 2D Segment Voronoi Diagram
- 2D Meshes
Polyhedra

- Half-edge data-structure
- Polyhedral surface (orientable 2-manifold with boundary)
- 2D Nef polygons
- 3D Nef polyhedra
Smallest enclosing circle and ellipse in 2D
Smallest enclosing sphere in dD
Largest empty rectangle
...
Arrangements

[Tel-Aviv]

- Line segments or polylines
- Conic arcs with Leda or Core

Completely new version in CGAL 3.2
Arbitrary dimension

- Range-tree, Segment-tree, kD-tree
- Window query
- Approximate nearest neighbors
- ...

Search Structures
Flexibility in the Basic Library
convex_hull_2<
InputIterator, OutputIterator, Traits>
Polygon_2<Traits, Container>
Polyhedron_3<Traits, HDS>
Triangulation_3<Traits, TDS>
Min_circle_2<Traits>
Range_tree_k<Traits>

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

```cpp
typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2<K> Delaunay;
```
• 3D points: coordinates \((x, y, z)\)
• orientation, in_circle: on \(x\) and \(y\) coordinates

typedef CGAL::Exact_predicates_inexact_constructions_kernel \(K\);
typedef CGAL::Triangulation_euclidean_traits_xy_3< K > Traits;
typedef CGAL::Delaunay_triangulation_2< Traits > Terrain;
The user can add information in vertices and cells
Work in Progress
Kinetic Data Structures
[Russel Karavelas]

Persistent Homology
[Kettner Zomorodian]

Surface reconstruction
[Oudot Rey]

3D Meshes
[Rineau Yvinec]

Parameterization
[Alliez]

Curved Kernel
Extension of the CGAL kernel
Algebraic issues
[Emiris Kakargias Pion Tsiganidas Teillaud SoCG’04]

...