Introduction to the
Computational Geometry Algorithms Library

Monique Teillaud

www.cgal.org

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SOPHIA ANTIPOLIS - MÉDITERRANÉE

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”*

$\Rightarrow$ **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"*

$\Rightarrow$ **robust programs**

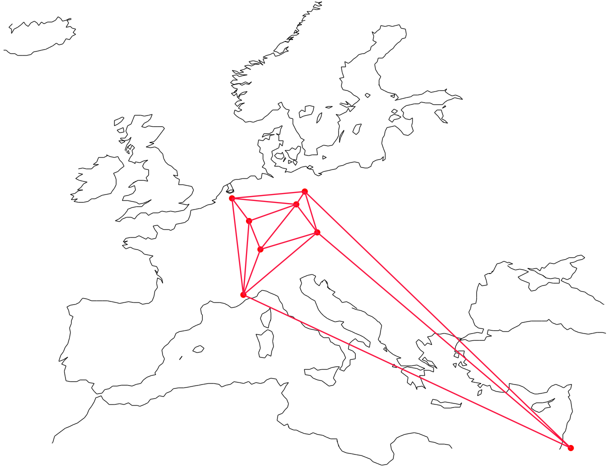
CG Impact Task Force Report, 1996

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 ∼12 months
- ∼ 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**.

$\longrightarrow$ 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.

Development process

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 $\longrightarrow$ ps, pdf, html
- svn server (INRIA gforge) for version management

Development tools

- Own manual tools: LaTeX $\longrightarrow$ 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

Development tools

- Own manual tools: LaTeX $\longrightarrow$ 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

Credit

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.

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.

Credit

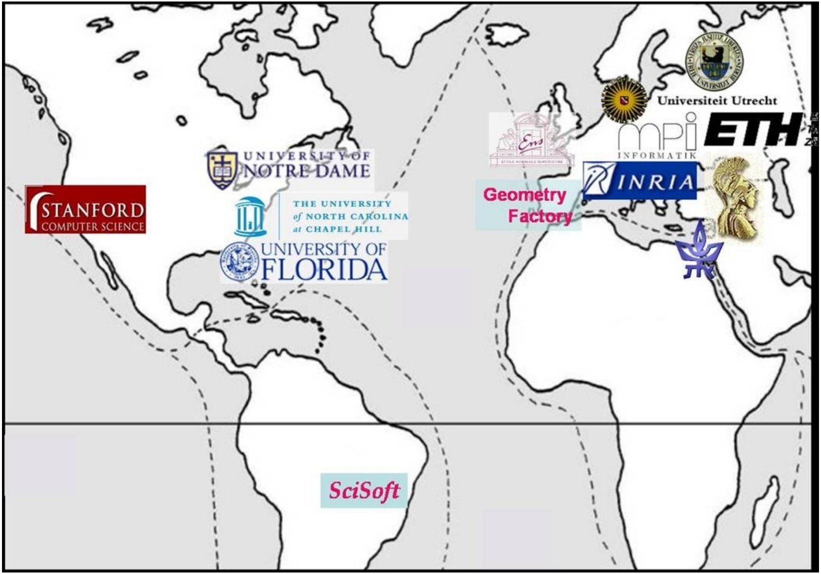
Contributors keep their identity

- **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

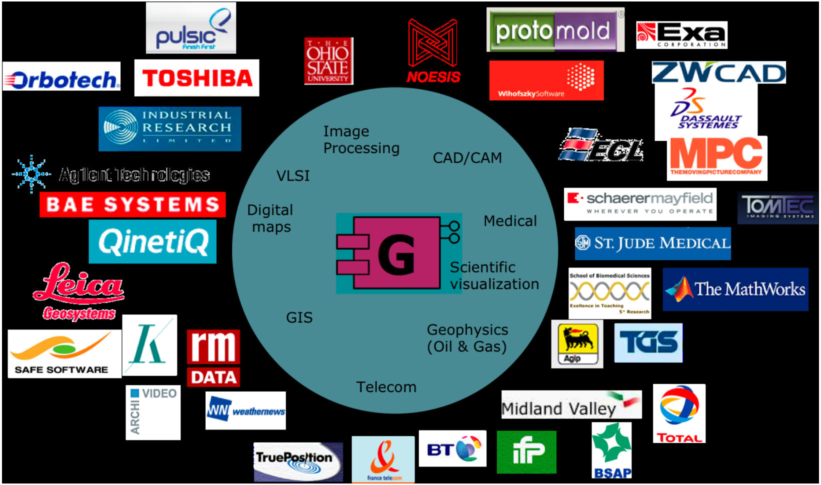


Users

Long list of identified users
(see web site)

More non-identified users. . .

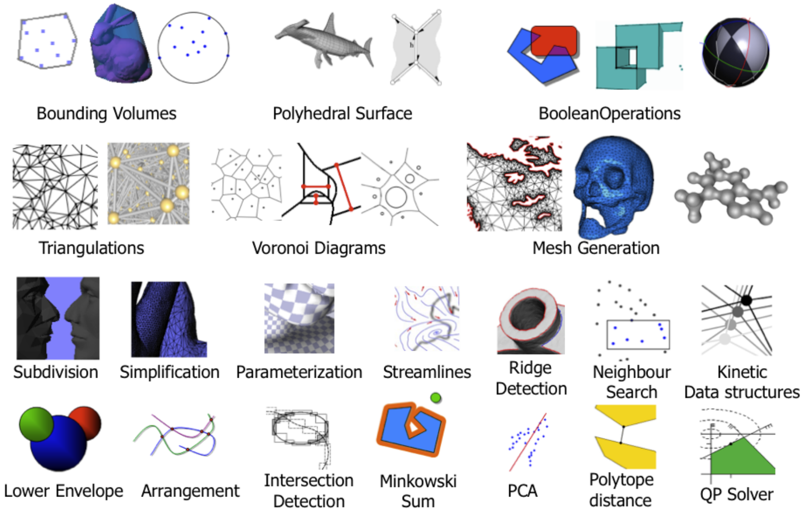
Customers of GEOMETRY FACTORY



Part II

Contents of CGAL

Contents



Bounding Volumes
Polyhedral Surface
BooleanOperations
Triangulations
Voronoi Diagrams
Mesh Generation
Subdivision
Simplification
Parameterization
Streamlines
Ridge Detection
Neighbour Search
Kinetic Data structures
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)

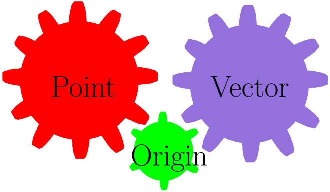
In the kernels

- Elementary geometric objects
- Elementary computations on them

Primitives	Predicates	Constructions
2D, 3D, dD	• comparison	• intersection
• Point	• Orientation	• squared distance
• Vector	• InSphere	...
• Triangle	...	
• Circle		
...		

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)

Kernels and number types

Cartesian representation

$$\text{Point} \left| \begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array} \right.$$

Homogeneous representation

$$\text{Point} \left| \begin{array}{l} hx \\ hy \\ hw \end{array} \right.$$

Kernels and number types

Cartesian representation

$$\text{Point} \left| \begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array} \right.$$

Homogeneous representation

$$\text{Point} \left| \begin{array}{l} hx \\ hy \\ hw \end{array} \right.$$

- ex: Intersection of two lines -

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

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

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right)$$

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Kernels and number types

Cartesian representation

$$\text{Point}\ \left|\ \begin{array}{l} x = \frac{hx}{hw} \\ y = \frac{hy}{hw} \end{array}\right.$$

Homogeneous representation

$$\text{Point}\ \left|\ \begin{array}{l} hx \\ hy \\ hw \end{array}\right.$$

- ex: Intersection of two lines -

$$\left\{ \begin{array}{l} a_1 x + b_1 y + c_1 = 0 \\ a_2 x + b_2 y + c_2 = 0 \end{array}\right.$$

$$\left\{ \begin{array}{l} a_1 hx + b_1 hy + c_1 hw = 0 \\ a_2 hx + b_2 hy + c_2 hw = 0 \end{array}\right.$$

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \right)$$

$$(hx, hy, hw) = \left(\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}, -\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}, \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \right)$$

Field operations

Ring operations

C++ Templates

`CGAL::`**Cartesian**< **FT** > (`CGAL::`**Simple_Cartesian**)

`CGAL::`**Homogeneous**< **RT** > (`CGAL::`**Simple_Homogen.**)

C++ Templates

`CGAL::`**Cartesian**< **FT** > (`CGAL::`**Simple_Cartesian**)
`CGAL::`**Homogeneous**< **RT** > (`CGAL::`**Simple_Homogen.**)

Cartesian Kernels : Field type

- `double`
- `Quotient<Gmpz>`
- `leda_real`

Homogeneous Kernels : Ring type

- `int`
- `Gmpz`
- `double`

C++ Templates

CGAL::**Cartesian**< **FT** > (CGAL::**Simple_Cartesian**)
CGAL::**Homogeneous**< **RT** > (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;
```

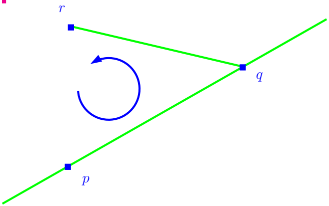
Numerical robustness issues

```
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



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

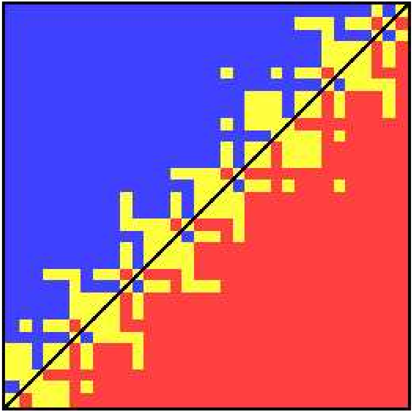
Numerical robustness issues

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, $= 0$, < 0



$\longrightarrow$ **inconsistencies** 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

$\neq$
exact arithmetics

More number types

- Detailed hierarchy of
algebraic and **arithmetic** **concepts** and **classes**

The circular/spherical kernels

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**

```
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: **RT** or **FT** ring or field type $(+, -, \times, /)$

For circles: `Root_of_2`< **RT** > $(<, =, >)$

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

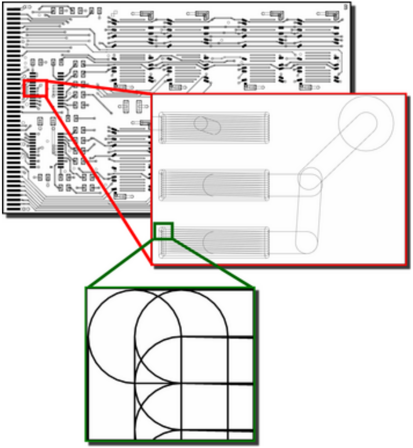
Polynomial representation of `Root_of_2`< **RT** > :

3 coefficients **RT** + 1 boolean

Sturm sequences, resultants, Descartes' rule,...
reduce **comparisons** to
computations of **signs of polynomial expressions**

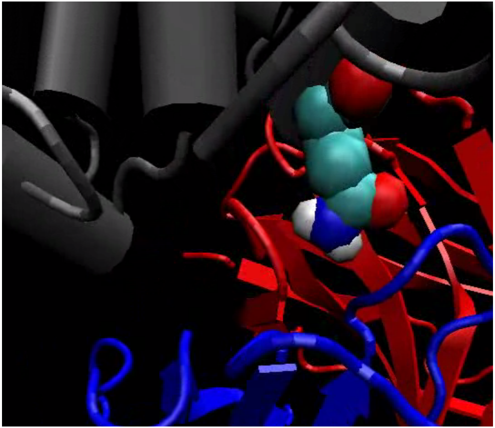
Application

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

```
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>
...
```

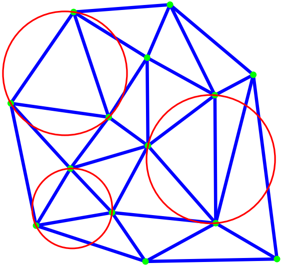
Geometric traits classes provide:

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

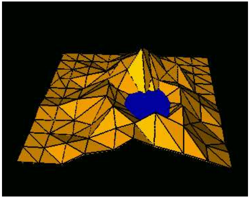
```
typedef
CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2< K > Delaunay;
```

Playing with traits classes

Delaunay Triangulation

- 3D points: coordinates $(\mathbf{x}, \mathbf{y}, \mathbf{z})$
- 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;
```



More flexibility

The user can add information in vertices and cells

...

To know more

CGAL

www.cgal.org