- 2D, 3D, dD “rational” kernels
- 2D circular and 3D spherical kernels
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

\[
\text{midpoint}(a,b) = a + \frac{1}{2} \times (b-a)
\]
## 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>
Kernels and number types

Cartesian representation

Point

\[
\begin{align*}
    x &= \frac{hx}{hw} \\
    y &= \frac{hy}{hw}
\end{align*}
\]

Homogeneous representation

Point

\[
\begin{align*}
    hx \\
    hy \\
    hw
\end{align*}
\]

- ex: Intersection of two lines -

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

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

\[
(hx, hy, hw) = \frac{1}{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}} \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}
\]
Kernels and number types

Cartesian representation

Point
\[ x = \frac{hx}{hw}, \quad y = \frac{hy}{hw} \]

Homogeneous representation

Point
\[ hx, \quad hy, \quad hw \]

- ex: Intersection of two lines -

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

\[
(x, y) = \left( \begin{array}{c} b_1 \\ b_2 \\ a_1 \\ a_2 \\ b_1 \\ b_2 \end{array} \right)^{-1} \left( \begin{array}{c} c_1 \\ c_2 \\ a_1 \\ a_2 \\ b_1 \\ b_2 \end{array} \right)
\]

Field operations

Ring operations
The “rational” Kernels

CGAL::Cartesian< FieldType >
CGAL::Homogeneous< RingType >

→ Flexibility

typedef double NumberType;
typedef Cartesian< NumberType > Kernel;
typedef Kernel::Point_2 Point;
Arithmetic robustness issues

Rational Kernels:
Predicates = signs of polynomial expressions

Exact Geometric Computation
≠ exact arithmetics

Predicates evaluated exactly
Filtering Techniques (interval arithmetics, etc)
exact arithmetics only when needed

CGAL::Exact_predicates_inexact_constructions_kernel
Arithmetic 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); // squared radius 2
Arithmetic 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); // squared radius 2
assert( C.has_on_boundary(q) );

OK    if NT gives exact sqrt
assertion violation otherwise
The circular/spherical kernels

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

Exact computations on algebraic numbers of degree 2
= roots of polynomials of degree 2

Algebraic methods reduce comparisons to
computations of signs of polynomial expressions
Application of the 2D circular kernel

Computation of arrangements of 2D circular arcs and line segments

Pedro M.M. de Castro, Master internship
Application of the 3D spherical kernel

Computation of arrangements of 3D spheres

Sébastien Loriot, PhD thesis
2D, 3D Triangulations in CGAL

1 Introduction
   • The CGAL Open Source Project
   • Contents of CGAL
   • The CGAL Kernels

2 2D, 3D Triangulations in CGAL
   • Introduction
   • Functionalities
   • Representation
   • Robustness
   • Software Design
Introduction

- The CGAL Open Source Project
- Contents of CGAL
- The CGAL Kernels

2D, 3D Triangulations in CGAL

- Introduction
  - Functionalities
  - Representation
  - Robustness
  - Software Design
Simplicial complex

= set $\mathcal{K}$ of $0, 1, 2, \ldots d$-faces such that

- each face is a simplex
- $\sigma \in \mathcal{K}, \tau \leq \sigma \Rightarrow \tau \in \mathcal{K}$
- $\sigma, \sigma' \in \mathcal{K} \Rightarrow \sigma \cap \sigma' \leq \sigma, \sigma'$
Various triangulations

2D, 3D, dD Basic triangulations
2D, 3D, dD Delaunay triangulations
2D, 3D, dD Regular triangulations
Basic and Delaunay triangulations

(figures in 2D)

Basic triangulations: incremental construction
Delaunay triangulations: empty sphere property
Introduction
- The CGAL Open Source Project
- Contents of CGAL
- The CGAL Kernels

2D, 3D Triangulations in CGAL
- Introduction
- Functionalities
- Representation
- Robustness
- Software Design
General functionalities

- Traversal of a 2D (3D) triangulation
  - passing from a face (cell) to its neighbors
  - iterators to visit all faces (cells) of a triangulation
  - circulators (iterators) to visit all faces (cells) incident to a vertex
  - circulators to visit all cells around an edge
General functionalities

- Traversal of a 2D (3D) triangulation
  - passing from a face (cell) to its neighbors
  - iterators to visit all faces (cells) of a triangulation
  - circulators (iterators) to visit all faces (cells)
    incident to a vertex
  - circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips
General functionalities

- Traversal of a 2D (3D) triangulation
  - passing from a face (cell) to its neighbors
  - iterators to visit all faces (cells) of a triangulation
  - circulators (iterators) to visit all faces (cells) incident to a vertex
  - circulators to visit all cells around an edge

- Point location query

- Insertion, removal, flips

- is_valid
  checks local validity (sufficient in practice)
  not global validity
Traversing a 3D triangulation

**Iterators**

- `All_cells_iterator`
- `All_faces_iterator`
- `All_edges_iterator`
- `All_vertices_iterator`

**Circulators**

- `Cell_circulator`: cells incident to an edge
- `Facet_circulator`: facets incident to an edge

C++ code snippet:
```cpp
All_vertices_iterator vit;
for (vit = T.all_vertices_begin(); vit != T.all_vertices_end(); ++vit)
    ...
```
Traversals of a 3D triangulation

Around a vertex

incident cells and facets, adjacent vertices

template <class OutputIterator>
OutputIterator
    t.incident_cells
    (Vertex_handle v, OutputIterator cells)
Point location, insertion, removal

basic triangulation:

Delaunay triangulation:
Additional functionalities for Delaunay triangulations

- Nearest neighbor queries
- Voronoi diagram
2D, 3D Triangulations in CGAL — Representation

1 Introduction
   • The CGAL Open Source Project
   • Contents of CGAL
   • The CGAL Kernels

2 2D, 3D Triangulations in CGAL
   • Introduction
   • Functionalities
   • Representation
   • Robustness
   • Software Design
The main algorithm

Incremental algorithm

- fully dynamic (point insertion, vertex removal)
- any dimension
- easier to implement
- efficient in practice
- ...
Needs

Walking in a triangulation

Access to
- vertices of a simplex
- neighbors of a simplex

in constant time
2D - Representation based on faces

Vertex
- Face_handle v_face

Face
- Vertex_handle vertex[3]
- Face_handle neighbor[3]
2D - Representation based on faces

Vertex

- Face_handle v_face

Face

- Vertex_handle vertex[3]
- Face_handle neighbor[3]

Edges are implicit: std::pair< f, i >
where f = one of the two incident faces.

From one face to another

n = f → neighbor(i)
j = n → index(f)

more efficient than half-edges
2D, 3D Triangulations in CGAL

Geometry vs. combinatorics

Each finite vertex stores a point
Arithmetic robustness

see above

Benchmarks

2.3 GHz, 16 GByte workstation

3.9 (Release mode)
Arithmetic robustness

see above

Benchmarks

2.3 GHz, 16 GByte workstation

3.9 (Release mode)

Delaunay triangulation - 10 Mpoints

<table>
<thead>
<tr>
<th>Kernel</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian &lt; double</td>
<td>9.7 sec</td>
<td>75 sec</td>
</tr>
<tr>
<td>Exact_predicates_inexact_constructions_kernel</td>
<td>10.6 sec</td>
<td>82 sec</td>
</tr>
</tbody>
</table>
Arithmetic robustness

see above

Benchmarks

2.3 GHz, 16 GByte workstation

Delaunay triangulation - 10 Mpoints

<table>
<thead>
<tr>
<th>Kernel</th>
<th>2D</th>
<th>3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian &lt; double &gt;</td>
<td>10.6 sec</td>
<td>82 sec</td>
</tr>
<tr>
<td>Exact_predicates_inexact_constructions_kernel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

may loop (or crash)!
Robustness

Pictures by Pierre Alliez
1 Introduction
   - The CGAL Open Source Project
   - Contents of CGAL
   - The CGAL Kernels

2 2D, 3D Triangulations in CGAL
   - Introduction
   - Functionalities
   - Representation
   - Robustness
   - Software Design
Traits class

Triangulation_2<Traits, TDS>

**Geometric traits classes provide:**

Geometric objects + predicates + constructors

**Flexibility:**

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

Generic algorithms

Delaunay_Triangulation_2< Traits, TDS >

Traits parameter provides:
• Point
• orientation test, in_circle test
Traits class

2D Kernel used as traits class

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

- 2D points: coordinates \((x, y)\)
- orientation, in_circle
Traits class

Changing the traits class

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Projection_traits_xy_3<K> Traits;
typedef CGAL::Delaunay_triangulation_2<Traits> Terrain;

- 3D points: coordinates \((x, y, z)\)
- orientation, in \_circle:
  - on \(x\) and \(y\) coordinates only
Layers

Triangulation\_3< Traits, TDS >

Triangulation
Geometry
location

Data Structure
Combinatorics
insertion

Geometric information
Additional information

Triangulation_data_structure_3< Vb, Cb > ;
Vb and Cb have default values.
The base level
Concepts `VertexBase` and `CellBase`.

Provide
- Point + access function + setting
- incidence and adjacency relations (access and setting)

Several models, parameterised by the `traits` class.
demos

web site www.cgal.org