Tutorial I: Programming with CGAL

Lutz Kettner

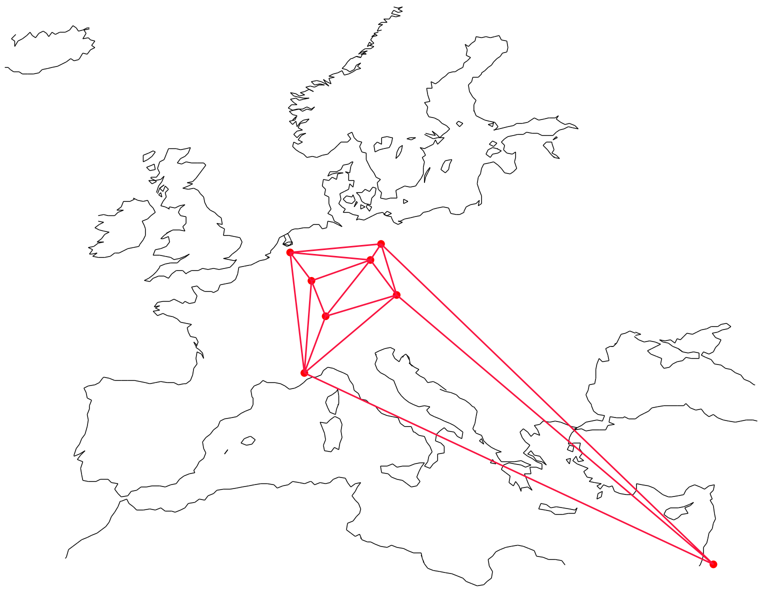
ETH Zürich, Switzerland

Rel. 2.0 (June 99): http://www.cs.uu.nl/CGAL

Contents

Computational Geometry Algorithms Library

ESPRIT IV LTR 21957 (CGAL)
and 28155 (GALIA)



MPI Saarbrücken
Universität des Saarlandes
ETH Zürich
Freie Universität Berlin
INRIA Sophia Antipolis
Martin-Luther-Universität
Halle-Wittenberg
Tel-Aviv University
Utrecht University
RISC Linz

Project Goal

make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications.

Deliverables and Platforms: R2.0 June 99

- Library: C++, source code delivered (templates) (100 KLOC)
- Systems: UNIX (SUN/SGI/Linux/...)
- Compiler: SGI Mips(Pro) CC 7.2.1,
  GNU g++ 2.8.1, CYGNUS egcs 1.1
- Supports: STL, LEDA, GMP, GeomView, Inventor, VRML, . . .
- Manuals: Printed (LaTeX) or Online (HTML)
- Getting Started: Manual + demo programs

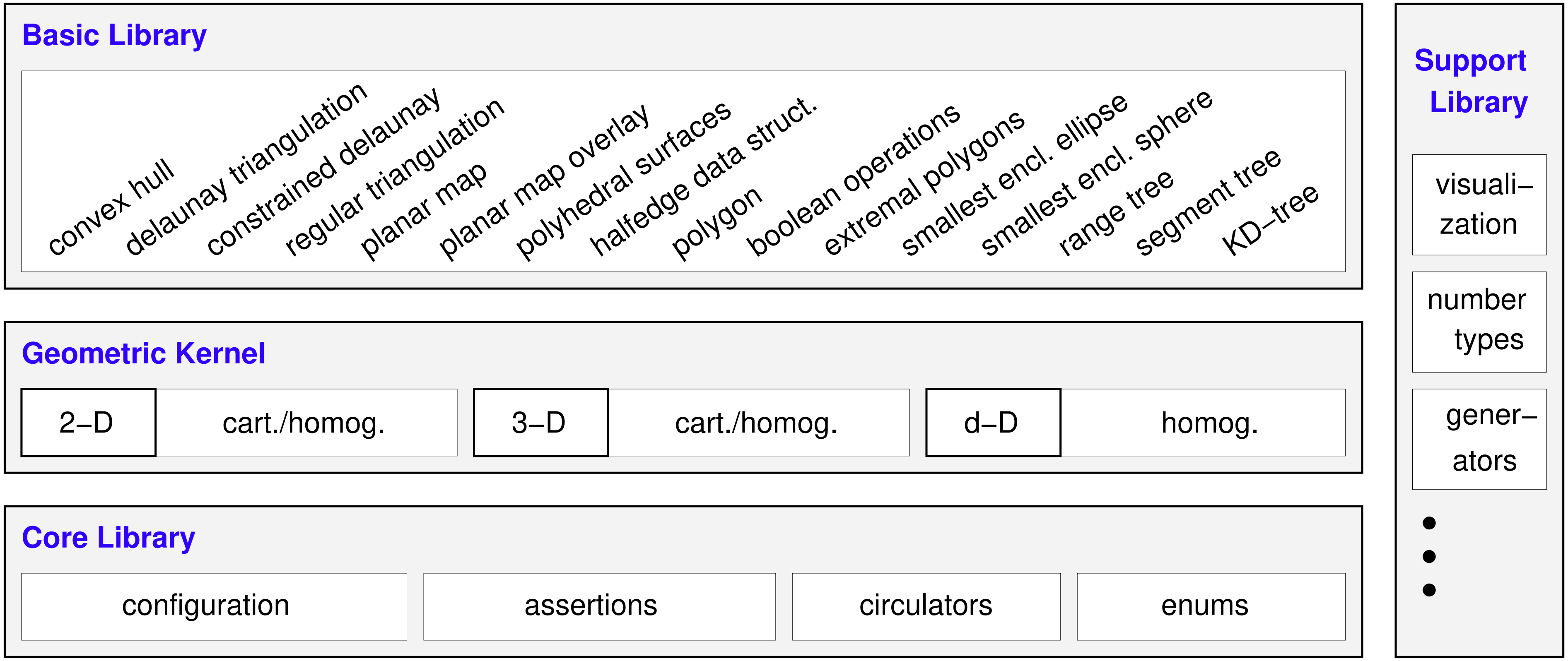
Changes from Release 1.2 to 2.0

- Standard header files: `<cstdlib>`, `<vector>`
- Standard namespace: `std::list`
- CGAL in its own namespace: `CGAL::`
- Unchanged functionality
- SunPro CC no longer supported

Future Plans

- New functionality
- Support of MS Visual C++

The Structure of CGAL



Challenges for a Library in CG

- **Exact arithmetic versus floating-point**

  CGAL
  - exact arithmetic, exact predicates
  - filtered predicates
  - robust floating point specializations

- **Degeneracy handling**

  CGAL
  - explicit treatment

- **Inherent complexity of many efficient solutions**

  CGAL
  - generic components, flexibility

Geometric Kernel

Geometric Kernel

- Points, Predicates, and Exactness
- Number Types
- Cartesian Representation
- Homogeneous Representation

Points, Predicates, and Exactness

```
#include "tutorial.h"
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>

int main() {
    Point p( 1.0, 0.0);
    Point q( 1.3, 1.7);
    Point r( 2.2, 6.8);
    switch ( CGAL::orientation( p, q, r)) {
        case CGAL::LEFTTURN:   std::cout << "Left turn.\n";  break;
        case CGAL::RIGHTTURN:  std::cout << "Right turn.\n"; break;
        case CGAL::COLLINEAR:  std::cout << "Collinear.\n";  break;
    }
    return 0;
}
```

Number Types

- Builtin: `double`, `float`, `int`, `long`, ...
- CGAL: `Filtered_exact`, `Interval_nt`, ...
- LEDA: `leda_integer`, `leda_rational`, `leda_real`, ...
- Gmpz: `CGAL::Gmpz`
- others are easy to integrate

Coordinate Representations

- Cartesian $p = (x, y)$: `CGAL::Cartesian<Field_type>`
- Homogeneous $p = (\frac{x}{w}, \frac{y}{w})$: `CGAL::Homogeneous<Ring_type>`

Cartesian with double

```
#include <CGAL/Cartesian.h>
#include <CGAL/Point_2.h>




int main() {
    CGAL::Point_2< CGAL::Cartesian<double> >  p( 0.1, 0.2);
    return 0;
}
```

Cartesian with double

```
#include <CGAL/Cartesian.h>
#include <CGAL/Point_2.h>


typedef CGAL::Cartesian<double>          Rep;
typedef CGAL::Point_2<Rep>               Point;

int main() {
    Point  p( 0.1, 0.2);
    return 0;
}
```

Cartesian with Filtered_exact and leda_real

```
#include <CGAL/Cartesian.h>
#include <CGAL/Arithmetic_filter.h>
#include <CGAL/leda_real.h>
#include <CGAL/Point_2.h>

typedef CGAL::Filtered_exact<double, leda_real> NT;
typedef CGAL::Cartesian<NT>                     Rep;
typedef CGAL::Point_2<Rep>                      Point;

int main() {
    Point  p( 0.1, 0.2);
    return 0;
}
```

Exact Orientation Test

```
#include <CGAL/Homogeneous.h>
#include <CGAL/Point_2.h>
#include <CGAL/predicates_on_points_2.h>
#include <iostream>
typedef CGAL::Homogeneous<long>          Rep;
typedef CGAL::Point_2<Rep>               Point;
int main() {
    Point p( 10,  0, 10);
    Point q( 13, 17, 10);
    Point r( 22, 68, 10);
    switch ( CGAL::orientation( p, q, r)) {
        case CGAL::LEFTTURN:   std::cout << "Left turn.\n";  break;
        case CGAL::RIGHTTURN:  std::cout << "Right turn.\n"; break;
        case CGAL::COLLINEAR:  std::cout << "Collinear.\n";  break;
    }
}
```

Generic Programming and the Basic Library

Generic Programming and the Basic Library

- Templates in C++
- Generic Function with Iterators
- Random Points as Input Iterators
- Delaunay Triangulation
- Convex Hull
- Circulators

Templates in C++

Function Template

```
template <class T> void swap( T& a, T& b) {
    T tmp = a; a = b; b = tmp;
}
```

```
int i = 5;
int j = 7;
swap( i, j);
```

Class Template

```
template <class T>  class list {
    void push_back( const T& x);
    typedef  ...  iterator;
    iterator begin();
    iterator end();
};
```

```
list<int> ls;
ls.push_back(5);
```

Generic Function with Iterators

```
template <class InputIterator, class OutputIterator>
OutputIterator copy( InputIterator  first,
                     InputIterator  last,
                     OutputIterator result) {
    while (first != last)
        *result++ = *first++;
    return result;
}
```

```
int a1[100];
int a2[100];
copy( a1, a1+100, a2);
```

```
list<int> ls1;
list<int> ls2;
copy( ls1.begin(), ls1.end(),
      back_inserter(ls2));
```

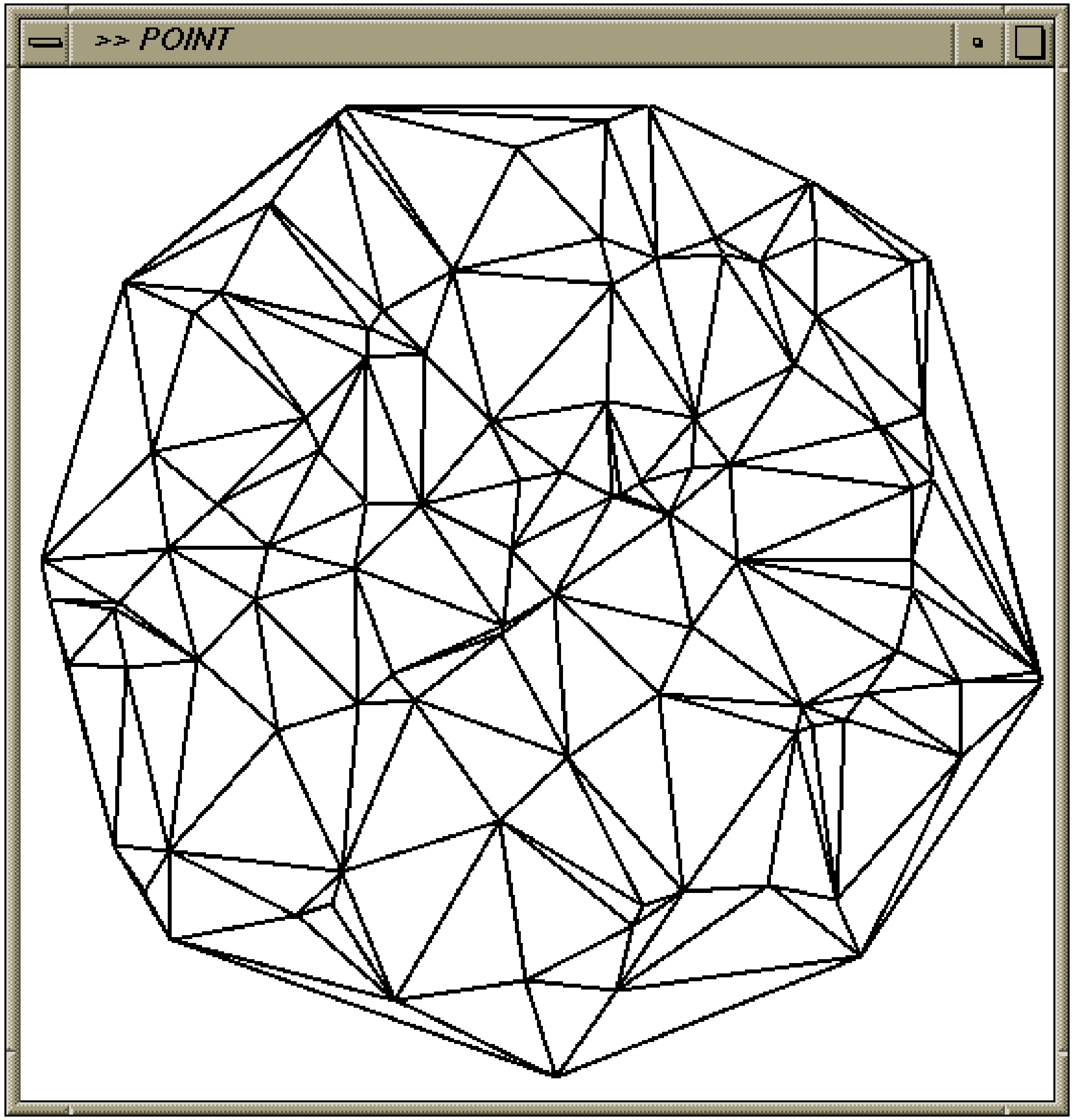
Random Points as Input Iterators

Iterator categories:

(input | output) ← forward ← bidirectional ← random access

```
typedef CGAL::Cartesian<double>                Rep;
typedef CGAL::Point_2<Rep>                     Point;
typedef CGAL::Random_points_in_disc_2<Point> Random_points_in_disc;

int main () {
    CGAL::Random            rnd(1);
    Random_points_in_disc   rnd_pts( 1.0, rnd);
    CGAL::copy_n( rnd_pts, 8, ostream_iterator<Point>(cout, "\n"));
    return 0;
}
```

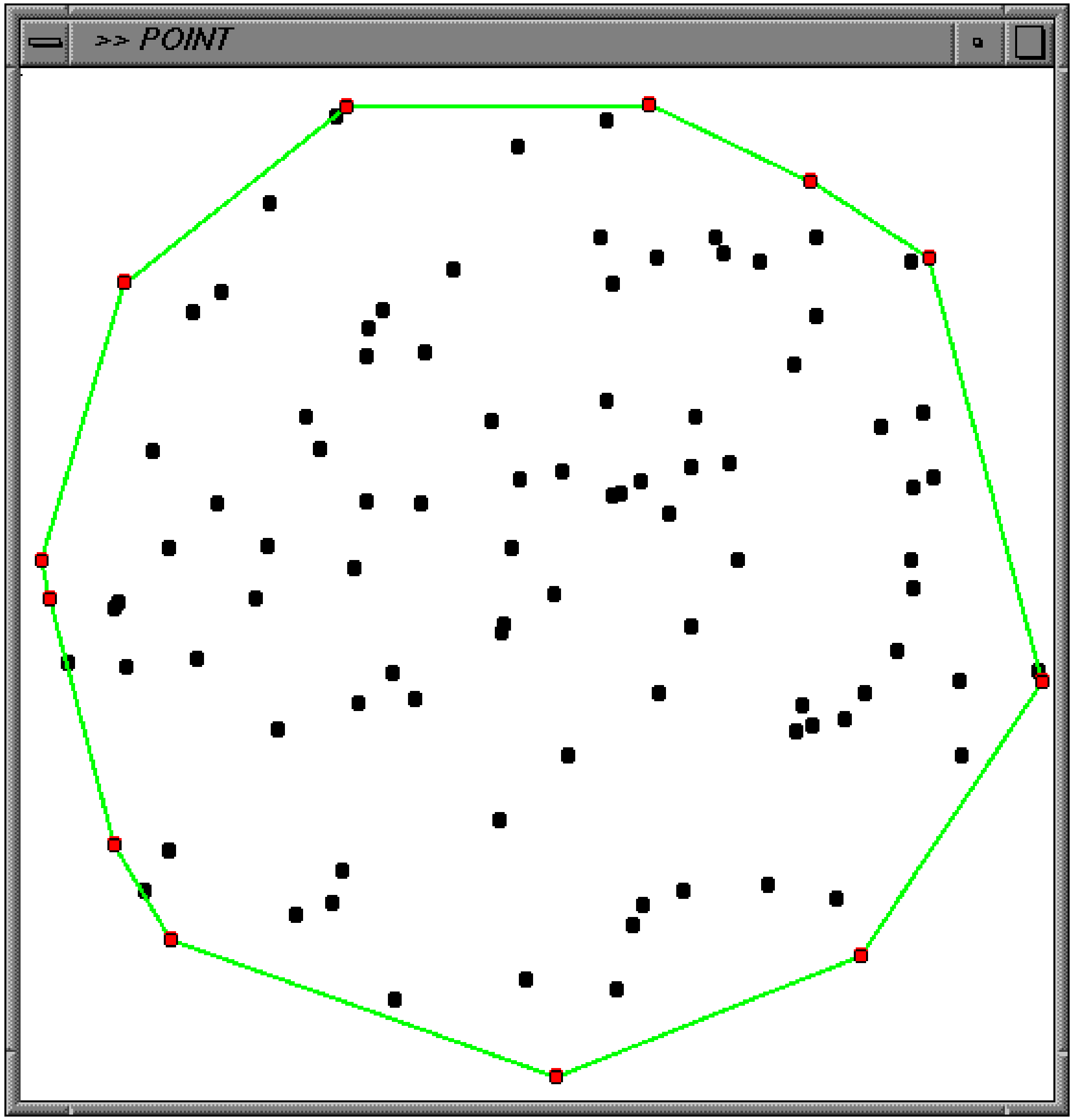
Delaunay Triangulation – Image



Delaunay Triangulation

```
int main () {
    CGAL::Random                                  rnd(1);   // random points
    CGAL::Random_points_in_disc_2<Point>  rnd_pts( 1.0, rnd);

    Delaunay_triangulation  dt;                             // triangulation
    CGAL::copy_n( rnd_pts, 100, std::back_inserter( dt));

    leda_window* window = CGAL::create_and_display_demo_window();
    *window << dt;                                          // window output
    Point  p;
    *window >> p;
    delete window;
    return 0;
}
```

Convex Hull Algorithm – Image



Convex Hull Algorithm

```
int main () {
    CGAL::Random                               rnd(1);  // random points
    CGAL::Random_points_in_disc_2<Point>  rnd_pts( 1.0, rnd);
    std::list<Point>                           pts;
    CGAL::copy_n( rnd_pts, 100, std::back_inserter( pts));

    Polygon  ch;                                        // convex hull
    CGAL::convex_hull_points_2( pts.begin(), pts.end(),
                                std::back_inserter(ch));

    leda_window* window = CGAL::create_and_display_demo_window();
    Window_iterator wout( *window);                     // window output
    std::copy( pts.begin(), pts.end(), wout);
    *window << CGAL::GREEN << ch << CGAL::RED;
    std::copy( ch.vertices_begin(), ch.vertices_end(), wout);
    // ...
```

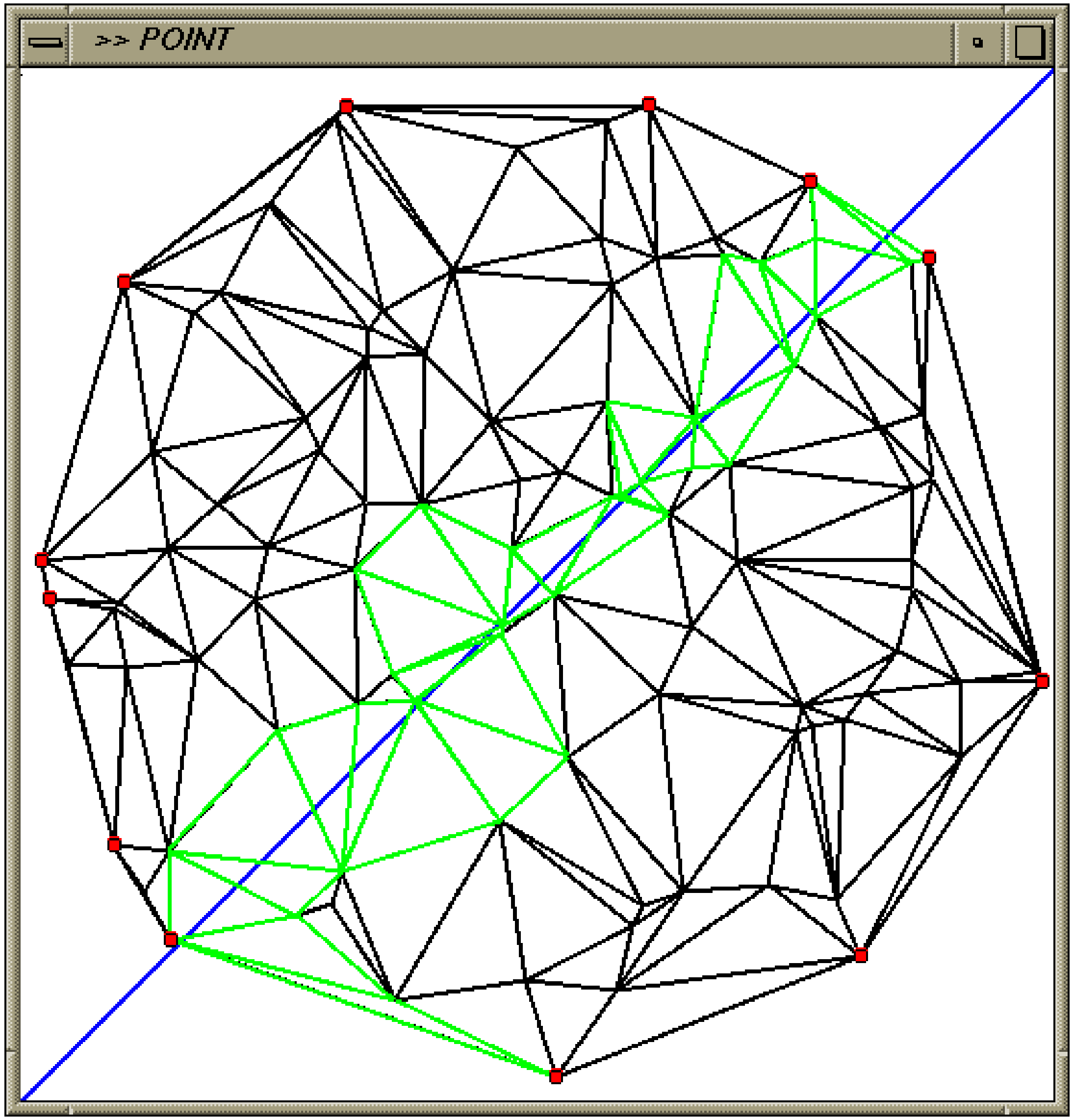
Circulators

Similar to iterators, but made for circular structures:

- Cyclic behavior of `operator++`.
- `do ...while()`-loop instead of `while()`-loop

```
template <class Circulator, class T>
bool contains( Circulator c, Circulator d, const T& value) {
    if (c != NULL) {
        do {
            if (*c == value)
                return true;
        } while (++c != d);
    }
    return false;
}
```

Circulators in Triangulations – Image



Circulators in Triangulations

```
Delaunay_triangulation  dt; leda_window* window; // ...

Delaunay_triangulation::Vertex_circulator vc =
    dt.incident_vertices( dt.infinite_vertex());
Delaunay_triangulation::Vertex_circulator vc_start = vc;
do {
    *window << vc->point();
} while ( ++vc != vc_start);

Delaunay_triangulation::Line_face_circulator lfc =
    dt.line_walk( Point( 0, 0), Point( 1, 1));
Delaunay_triangulation::Line_face_circulator lfc_start = lfc;
do {
    if ( ! dt.is_infinite( lfc))
        *window << dt.triangle( lfc);
} while ( ++lfc != lfc_start);
```

CGAL::Object

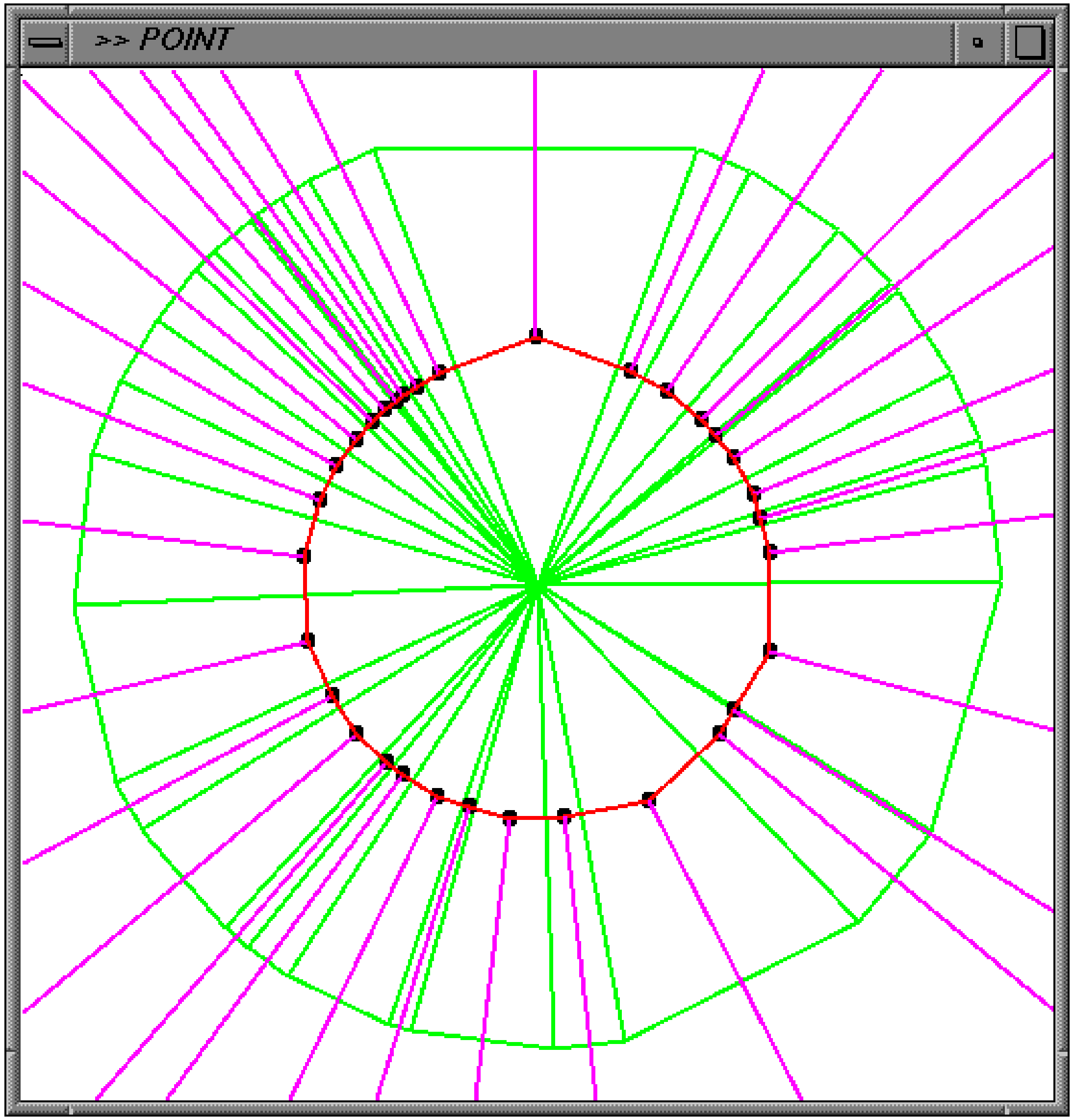
CGAL::Object

- Intersection of two Segments
- Voronoi Diagram

Intersection of two Segments

```
#include <CGAL/intersections.h>

int main() {
    Segment s( Point(1,1), Point(1,5));
    Segment t( Point(1,3), Point(1,8));
    if ( CGAL::do_intersect( s, t)) {
        CGAL::Object result = CGAL::intersection( s, t);
        Point   pt;
        Segment seg;
        if (CGAL::assign( pt, result))
            std::cout << "intsct. point = " << pt << std::endl;
        else if (CGAL::assign( seg, result))
            std::cout << "intsct. segment = " << seg << std::endl;
    } else
        std::cout << "no intersection" << std::endl;
}
```

Voronoi Diagram – Image



Voronoi Diagram

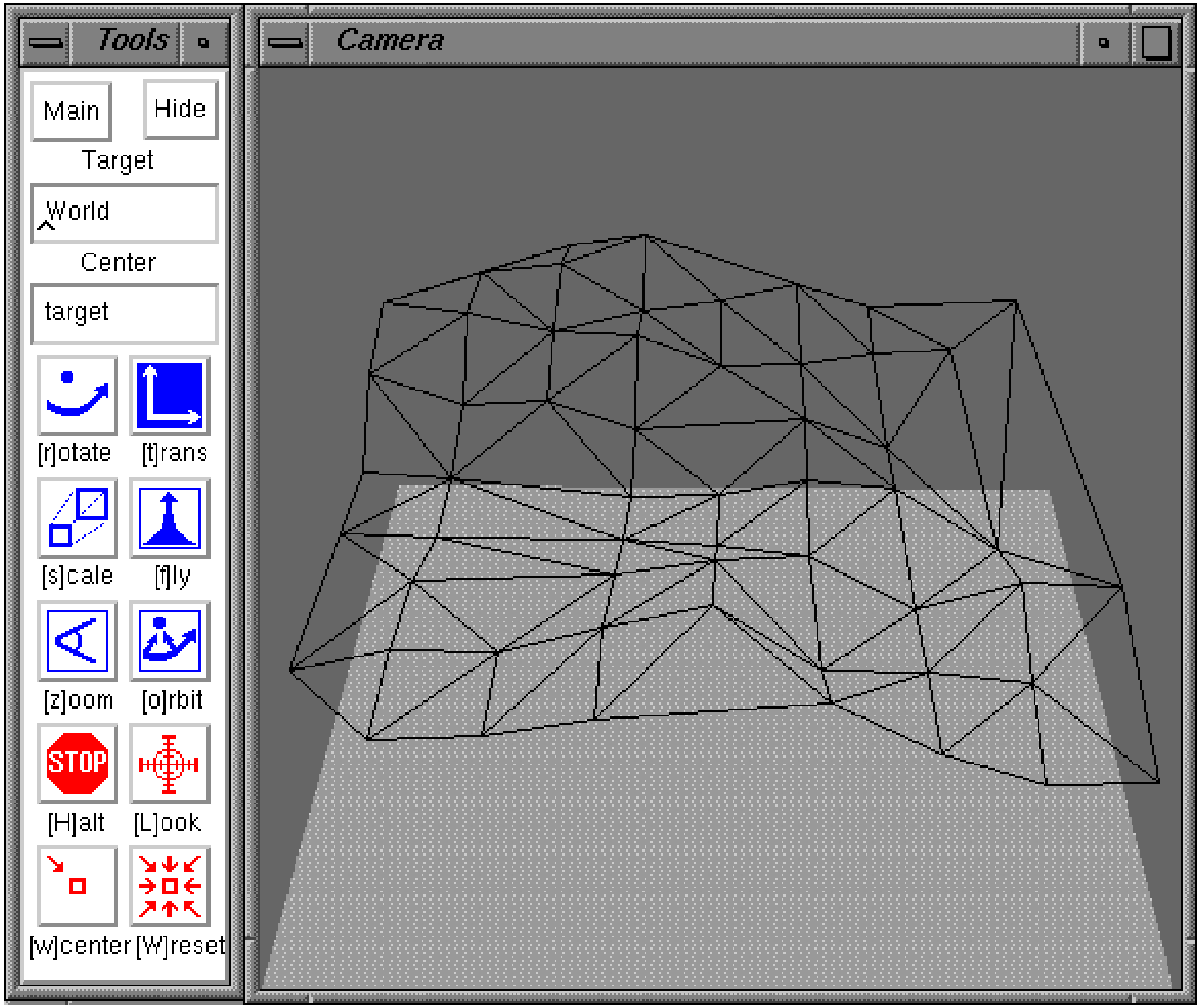
```
Delaunay_triangulation  dt; leda_window* window; // ...

Delaunay_triangulation::Edge_iterator e = dt.edges_begin();
while ( e != dt.edges_end()) {
    CGAL::Object o = dt.dual(e);
    Delaunay_triangulation::Segment seg;
    Delaunay_triangulation::Ray     ray;
    if (CGAL::assign( seg, o))
        *window << CGAL::RED << seg;
    else if (CGAL::assign( ray, o))
        *window << CGAL::VIOLET << ray;
    ++e;
}
```

Traits Classes in the Basic Library

Traits Classes in the Basic Library

- Triangulation of Terrains: xy-Projection of 3d-Points
- Independence from the Geometric Kernel
- Lexicographic Sorting with a Function Object
- LEDA Kernel and CGAL Convex Hull

Triangulation of Terrains – Image



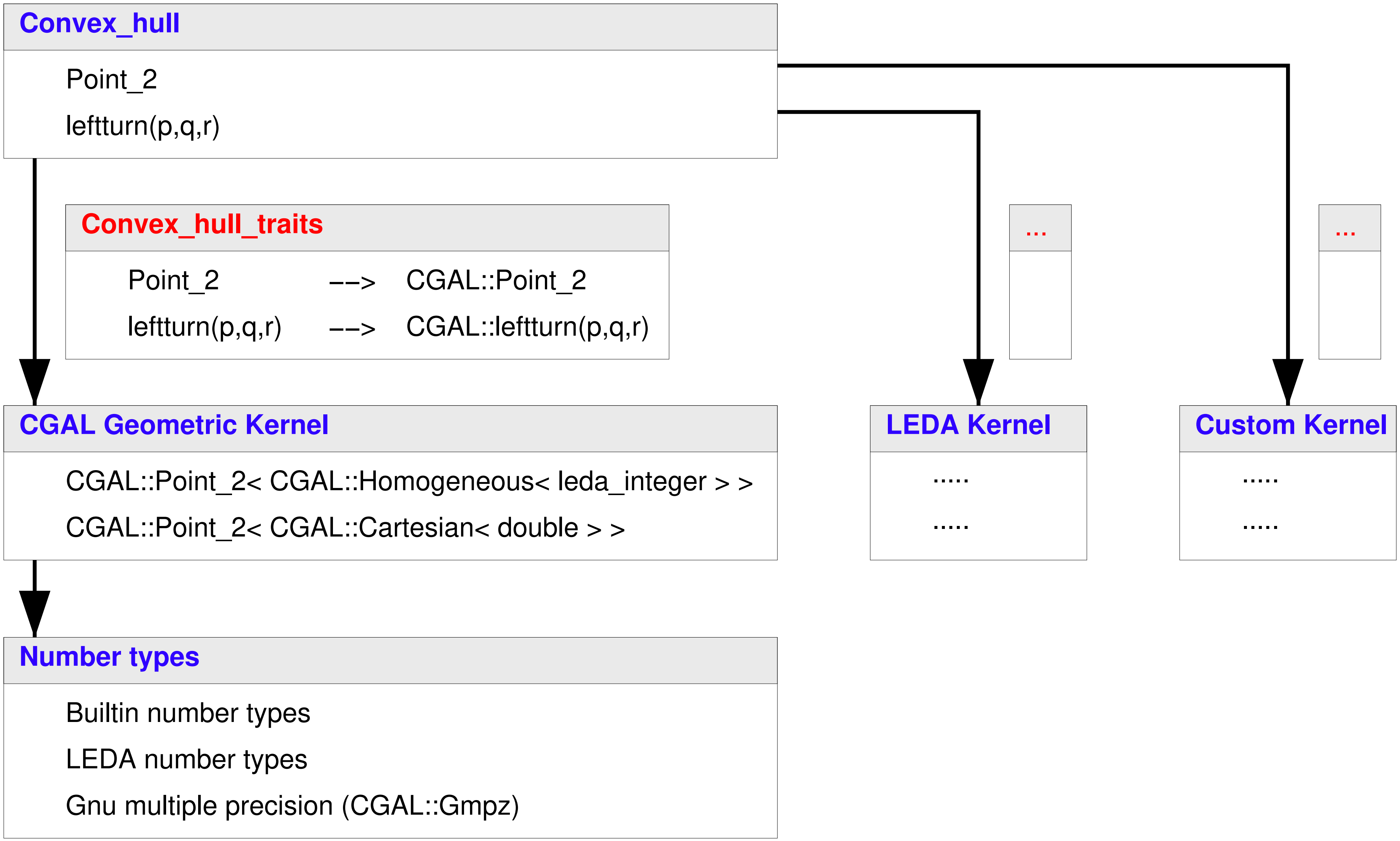
Declaration of the 2d-Triangulation

```
typedef CGAL::Triangulation_euclidean_traits_2<TutorialR>    Traits;
typedef CGAL::Triangulation_vertex_base_2<Traits>            Vb;
typedef CGAL::Triangulation_face_base_2<Traits>              Fb;
typedef CGAL::Triangulation_default_data_structure_2<Traits,Vb,Fb> Tds;
typedef CGAL::Delaunay_triangulation_2<Traits,Tds>           Delaunay;

int main () {
    Delaunay  d;
    copy( istream_iterator<Point>(cin), istream_iterator<Point>(),
          back_inserter(d));
    // ... window output
}
```

Triangulation of Terrains

```
typedef CGAL::Point_3<TutorialR>                                 Pt;
typedef CGAL::Triangulation_euclidean_traits_xy_3<TutorialR> Traits;
typedef CGAL::Triangulation_vertex_base_2<Traits>                Vb;
typedef CGAL::Triangulation_face_base_2<Traits>                  Fb;
typedef CGAL::Triangulation_default_data_structure_2<Traits,Vb,Fb> Tds;
typedef CGAL::Delaunay_triangulation_2<Traits,Tds>               Delaunay;

int main () {
    Delaunay  d;
    copy( istream_iterator<Pt>(cin), istream_iterator<Pt>(),
          back_inserter(d));
    CGAL::Geomview_stream geomview( CGAL::Bbox_3( -1,-1,-1, 1,1,1));
    for ( Delaunay::Edge_iterator e = d.edges_begin();
          e != d.edges_end(); ++e)
        geomview << d.segment(e);
}
```

Independence from the Geometric Kernel



Function Objects: Lexicographic Sorting

```
#include "tutorial.h"
#include <vector>

struct LexComp {
    bool operator()( const Point& p, const Point& q) const {
        return q.x() < q.x() || (q.x() == q.x() && q.y() < q.y());
    }
};

int main() {
    std::vector<Point> points;
    // ...
    std::sort( points.begin(), points.end(), LexComp());
    return 0;
}
```

LEDA Kernel and CGAL Convex Hull

```
typedef leda_rat_point   Point;
struct Traits {
  typedef Point  Point_2;
  struct  Less_xy {
    bool operator()( const Point& p, const Point& q) {
        return (p.xcoord() <  q.xcoord()) || (
               (p.xcoord() == q.xcoord()) && (p.ycoord() < q.ycoord()));
  }};
  struct  Leftturn {
    bool operator()( const Point& p, const Point& q, const Point& r) {
        return left_turn( p, q, r); // leda_left_turn

  }};
  Less_xy  get_less_xy_object()  const { return Less_xy();  }
  Leftturn get_leftturn_object() const { return Leftturn(); }
};
```

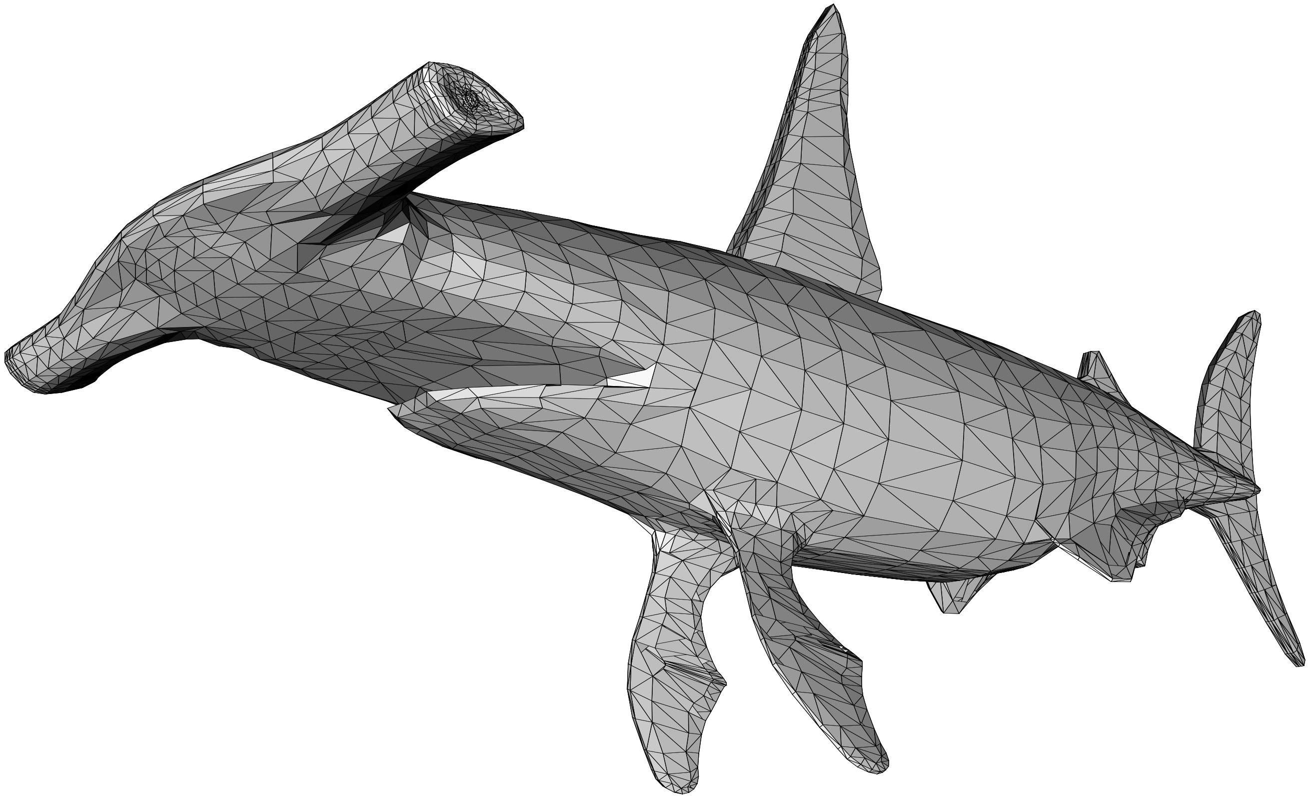
LEDA Kernel and CGAL Convex Hull (2)

```
int main() {
    std::list<Point>  pts;  // ...
    std::list<Point>  ch;
    CGAL::ch_graham_andrew( pts.begin(), pts.end(),
                            std::back_inserter(ch), Traits());
    // ...
}
```

Polyhedral Surfaces

Polyhedral Surfaces

- Declaring and File IO
- Triangulating Convex Facets



Declaring and File IO

```
#include <CGAL/Halfedge_data_structure_polyhedron_default_3.h>
#include <CGAL/Polyhedron_default_traits_3.h>
#include <CGAL/Polyhedron_3.h>
#include <CGAL/IO/Polyhedron_iostream.h>

typedef CGAL::Cartesian<double>                               Rep;
typedef CGAL::Polyhedron_default_traits_3<Rep>                Traits;
typedef CGAL::Halfedge_data_structure_polyhedron_default_3<Rep> HDS;
typedef CGAL::Polyhedron_3<Traits,HDS>                        Polyhedron;

int main() {
    Polyhedron P;
    std::cin >> P;
    // ...
    std::cout << P;
}
```

Triangulating Facets

```
struct Triangulate {
  Polyhedron* P;
  Triangulate( Polyhedron& pol) : P(&pol) {}
  typedef Polyhedron::Facet Facet;
  void operator()( Facet& f) {
    while ( CGAL::circulator_size( f.facet_begin ()) > 3)
      P->split_facet( f.halfedge(), f.halfedge()->next()->next());
  }
};
```

```
std::for_each( P.facets_begin(), P.facets_end(), Triangulate(P));
```

Curve Reconstruction: crust

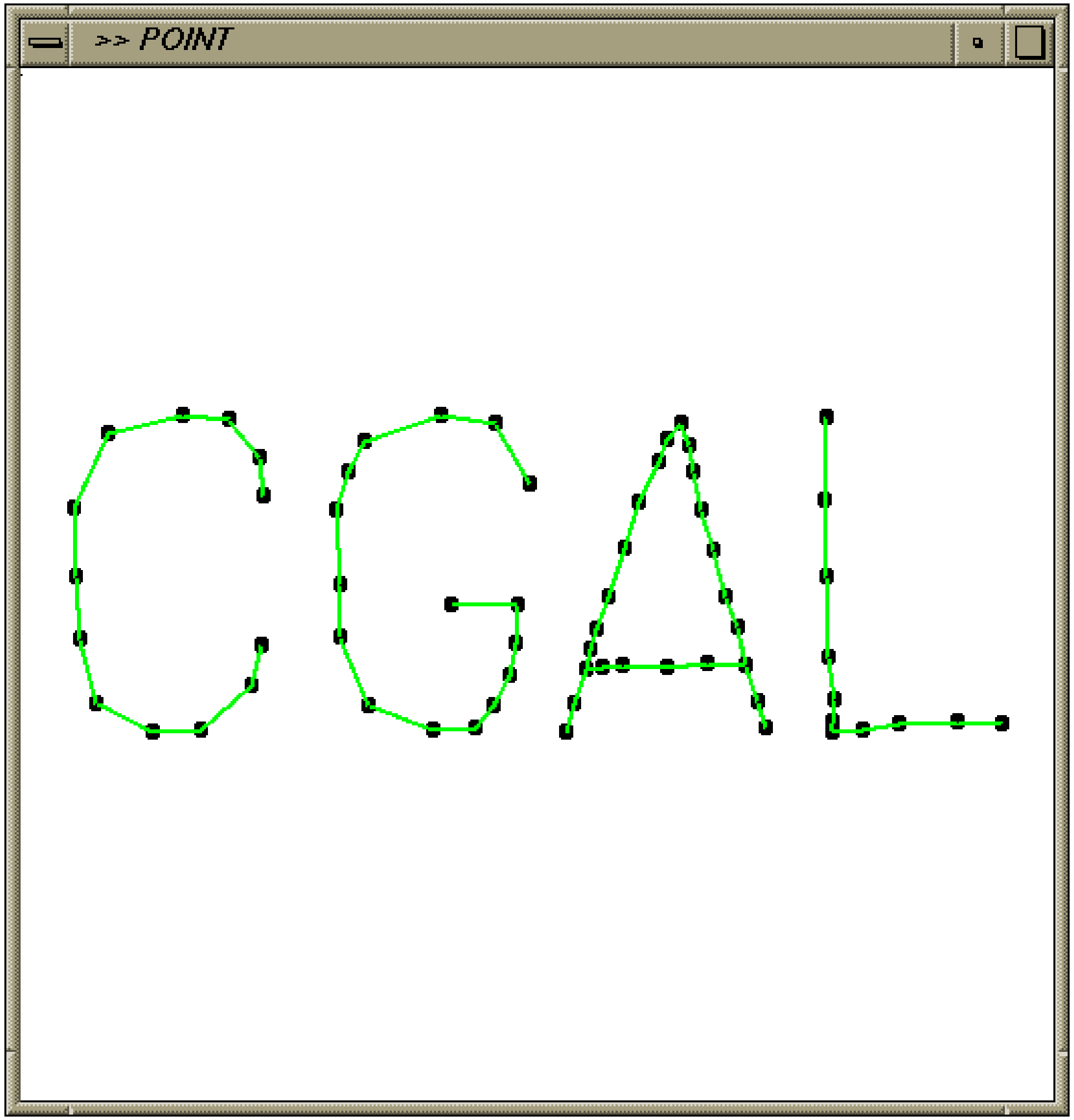
Curve Reconstruction: crust

Input: a set P of points in the plane

Output: a set S of segments with endpoints in P

1. Construct the Voronoi diagram *VD* of P.
2. Construct the Delaunay triangulation *DT* of $P \cup V$, where V are the vertices of *VD*.
3. $S := \forall$ edges of *DT* with endpoints in P.

Crust – Image



Crust: Declaring Own Point Type

```
struct Tmp_point : public Point {
    static bool init;
    bool delaunay;
    Tmp_point() : delaunay( init) {}
    Tmp_point( const Point& p) : Point(p), delaunay( init) {}
};

struct Traits : public EucliTraits {
    typedef Tmp_point Point;
};

typedef CGAL::Triangulation_vertex_base_2<Traits>                    Vb;
typedef CGAL::Triangulation_face_base_2<Traits>                      Fb;
typedef CGAL::Triangulation_default_data_structure_2<Traits,Vb,Fb> Tds;
typedef CGAL::Delaunay_triangulation_2<Traits,Tds>              Delaunay;
```

Crust: Computation

```
template <class InputIterator, class OutputIterator>
void crust( InputIterator first, InputIterator last,
            OutputIterator out) {
    Delaunay dt;
    Tmp_point::init = true;
    dt.insert( first, last);
    vector<Point> voronoi_vertices;
    voronoi_vertices.reserve( dt.number_of_faces());
    Delaunay::Face_iterator f = dt.faces_begin();
    while ( f != dt.faces_end()) {
        voronoi_vertices.push_back( dt.dual( f));
        ++f;
    }
    Tmp_point::init = false;
    dt.insert( voronoi_vertices.begin(), voronoi_vertices.end());
```

Crust: Selection

```
    Delaunay::Edge_iterator e = dt.edges_begin();
    while ( e != dt.edges_end()) {
        Delaunay::Face_handle face = (*e).first;
        int index = (*e).second;
        if ( face->vertex( dt.cw(  index))->point().delaunay &&
             face->vertex( dt.ccw( index))->point().delaunay)
            *out++ = dt.segment(e);
        ++e;
    }
}
```

Other Examples

- Midpoint of two Points
- Affine Transformation of Polyhedra

Release 2.0 available (June 99)

http://www.cs.uu.nl/CGAL