OI-OPPA. European Social Fund
Prague & EU: We invest in your future.

CGAL[1] Genericity

Philosophy of CGAL structure

Petr Kubizňák

CTU, FEL, A4M39VG

2012-12-06

[1] Computational Geometry Algorithms Library

Summary

Motivation

1. Want to finally understand CGAL better?
   1. Traits
   2. Kernel
2. Want to know the practices used for object modelling?
3. Love abstract programming?

About Seagull

Figure: Seagull. [stanleyseagull]

About CGAL

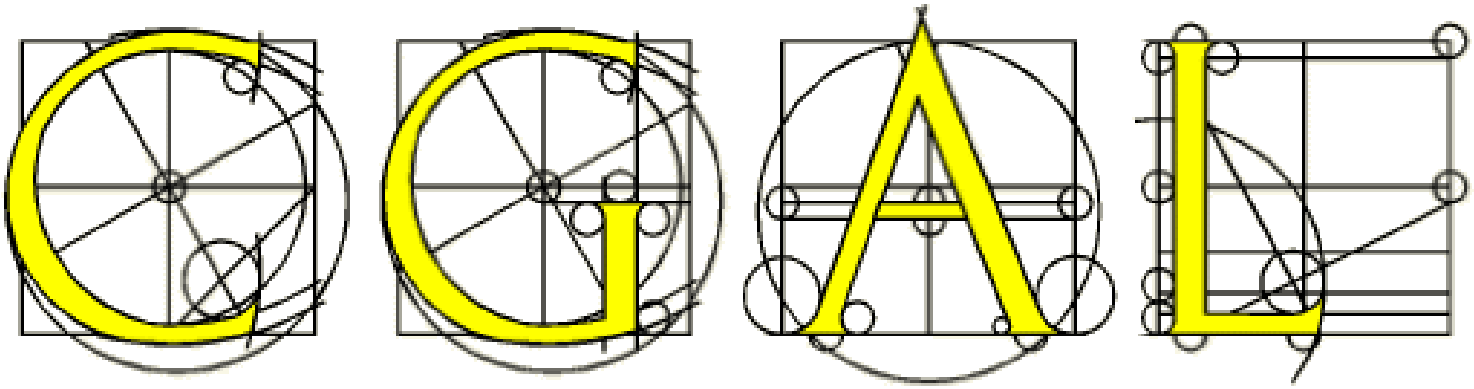
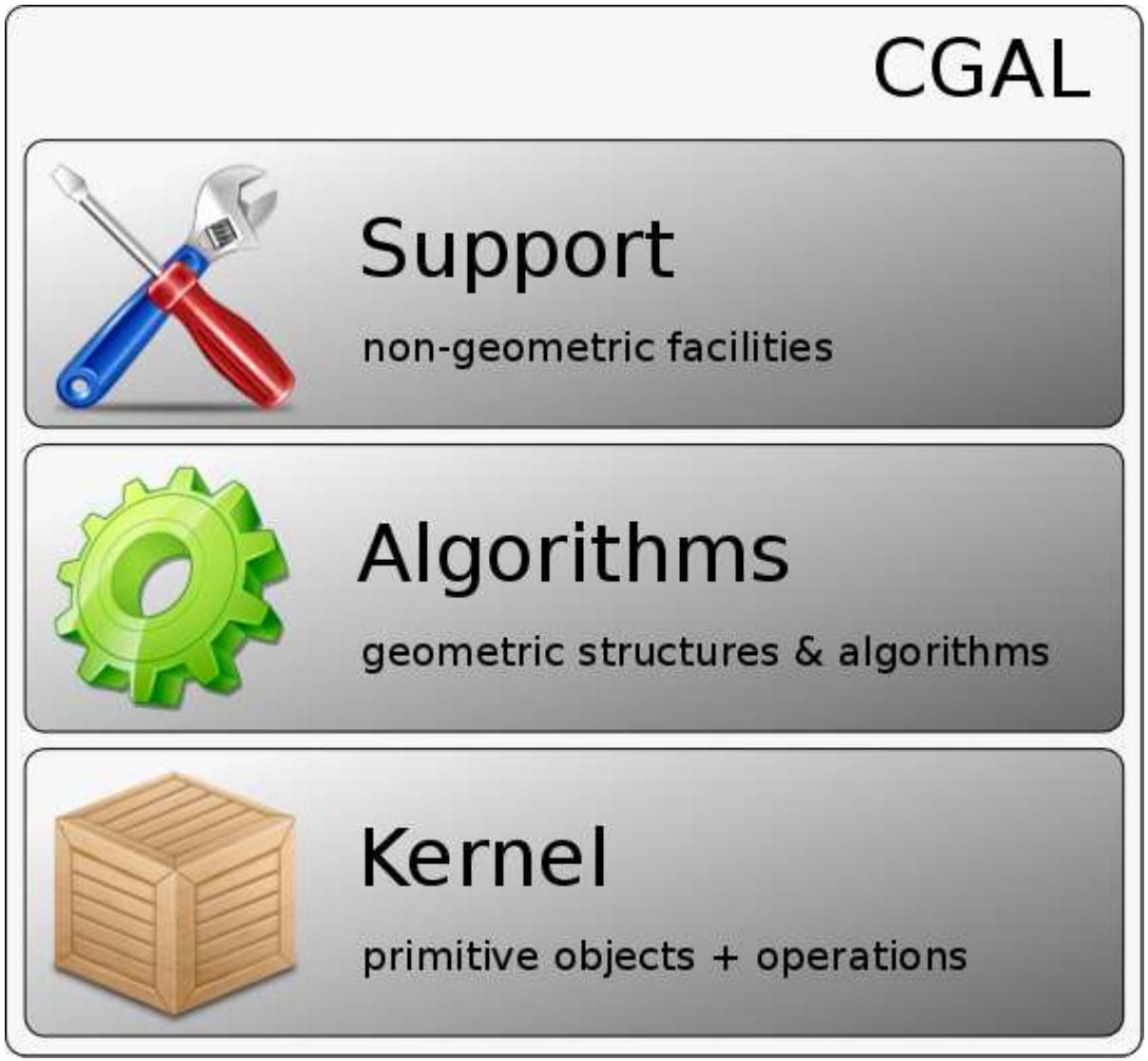


Figure: CGAL logo. [cgal]

- C++ library of geometric *data structures* and *algorithms*, e.g.:
  - points, lines, planes, polygons; trees; ...
  - distance, orientation, interpolation, convex hull, ...
- Stress on *adaptability*, *extensibility* and *efficiency*

CGAL Structure



Support Facilities

- STL Extensions
- Handles, Ranges and Circulators
- Geometric Object Generators
- Profiling Tools
- IO Streams
- Visualization Tools (BGL[2], Qt, ...)
- ...

[2] Boost Graph Library

Algorithms

- Universal implementations of geometric algorithms, e.g. convex hull computation algorithms
- Interface provided by traits classes

```
1  #include <CGAL/ch_graham_andrew.h>
2
3  template <class InputIterator, class OutputIterator, class Traits>
4  OutputIterator ch_graham_andrew(InputIterator first,
5                                  InputIterator beyond,
6                                  OutputIterator result,
7                                  Traits ch_traits = Default_traits)
```

Traits

Wikipedia: "traits class is a class template used to associate state and/or behaviour to a compile-time entity, typically a data type or a constant, without modifying the existing entity" [wikipedia]

- Template overcoming problems of inheritance
- Form interface between data and algorithms
- *"Define the interface between the data structure or algorithm and the primitives they use"* [cgal]
- *"Classes are composed from a set of traits by specifying glue code that connects the traits together"* [scharli]

Example: ConvexHullTraits_2 I

- *Concept* defining the primitives (objects and predicates) that the convex hull algorithms use
- Types:
  - ConvexHullTraits_2::Point_2
  - ConvexHullTraits_2::Equal_2
  - ConvexHullTraits_2::Less_xy_2
  - ConvexHullTraits_2::Less_yx_2
  - ConvexHullTraits_2::Left_turn_2
  - ConvexHullTraits_2::Less_signed_distance_to_line_2
  - ConvexHullTraits_2::Less_rotate_ccw_2
- Operations:
  - Equal_2 `traits.equal_2_object ()`
  - Less_xy_2 `traits.less_xy_2_object ()`
  - ...

Example: ConvexHullTraits_2 II

- Models:
  - `CGAL::Convex_hull_traits_2<R>`
    - model of ConvexHullTraits_2
    - `#include <CGAL/convex_hull_traits_2.h>`
  - `CGAL::Convex_hull_constructive_traits_2<R>`
    - model of ConvexHullTraits_2
    - `#include <CGAL/convex_hull_constructive_traits_2.h>`
  - `CGAL::Projection_traits_xy_3<K>`
    - model of TriangulationTraits_2
    - model of DelaunayTriangulationTraits_2
    - model of ConstrainedTriangulationTraits_2
    - model of PolygonTraits_2
    - model of ConvexHullTraits_2
    - `#include <CGAL/Projection_traits_xy_3.h>`

Kernel I

- *Concept* defining constant-size non-modifiable geometric primitive objects and operations on these objects
- *Types:*
  - Kernel::FT
  - Kernel::RT
  - Kernel::Boolean
  - Kernel::Angle
- *Access:*
  - Kernel::Cartesian_const_iterator_2
- *Geometric Objects:*
  - Kernel::Point_2
  - Kernel::Vector_2
  - Kernel::Triangle_2

Kernel II

- *Constructions:*
  - Kernel::Construct_point_2
  - Kernel::Construct_vector_2
  - Kernel::Construct_triangle_2
- *Constructions with unknown return type:*
  - Kernel::Intersect_2
- *Predicates:*
  - Kernel::Angle_2
  - Kernel::Equal_2
  - Kernel::Less_xy_2

Kernel III

- *Models:*
  - `CGAL::Cartesian<FieldNumberType>`
  - `CGAL::Homogeneous<RingNumberType>`
  - `CGAL::Simple_cartesian<FieldNumberType>`
  - `CGAL::Simple_homogeneous<RingNumberType>`
  - `CGAL::Filtered_kernel<CK>`
  - `CGAL::Exact_predicates_exact_constructions_kernel`
  - `CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt`
  - `CGAL::Exact_predicates_inexact_constructions_kernel`

Kernel Families

- Cartesian Kernels
  - Cartesian coordinates x, y
  - Operate on FieldNumberType[3]
  - `CGAL::Cartesian<FieldNumberType>`
  - `CGAL::Simple_cartesian<FieldNumberType>`
- Homogeneous Kernels
  - Homogenenous coordinates hx, hy, hw
  - Operate on RingNumberType[4], *where possible*
  - `CGAL::Homogeneous<RingNumberType>`
  - `CGAL::Simple_homogeneous<RingNumberType>`

[3] +, -, *, / ; e.g. `double`

[4] +, -, * ; e.g. `int`

Kernel Object Example: Point_2<Kernel>

- *Model* of the Kernel::Point_2 concept
- *Constructors:*
  - `Point_2<Kernel> p (Kernel::FT x, Kernel::FT y);`
  - `Point_2<Kernel> p (Kernel::RT hx, Kernel::RT hy, Kernel::RT hw = RT(1));`
- *Access functions:*
  - `Kernel::FT p.x() const;`
  - `Kernel::RT p.hx() const;`
- *Tests:*
  - `bool p.operator== (q) const;`
  - `bool operator< (p, q);`

Usage

```
1  #ifndef ORIENTATIONWIDGET_H
2  #define ORIENTATIONWIDGET_H
3
4  typedef CGAL::Cartesian<CGAL::Gmpfr>    Kernel;
5  typedef CGAL::Point_2<Kernel>           Point_2;
6  ...
```

```
1  #include "orientationwidget.h"
2
3  void OrientationWidget::paintEvent(QPaintEvent * paintEvent) {
4      QPainter painter(this);
5      ....
6      Point_2 q(12, 12);
7      Point_2 r(24, 24);
8      float u = pow(2.0,_epsilonExponent);
9      ...
10 }
```

Questions

Figure: [quomodo]

Traits Example: The Problem [myers]

```
1  class streambuf {
2      ...
3      int sgetc();  // return the next character, or EOF.
4      int sgetn(char*, int N);  // get N characters.
5  };
6
7  template <class charT, class intT>
8  class basic_streambuf {
9      ...
10     intT sgetc();
11     int sgetn(charT*, int N);
12 };
```

Traits Example: The Solution [myers] I

```
1  template <class charT> struct ios_char_traits { };
2
3  struct ios_char_traits<char> {
4      typedef char char_type;
5      typedef int  int_type;
6      static inline int_type eof() { return EOF; }
7  };
8
9  template <class charT>
10 class basic_streambuf {
11     public:
12       typedef ios_char_traits<charT> traits_type;
13       typedef traits_type::int_type int_type;
14       int_type eof() { return traits_type::eof(); }
15       ...
16       int_type sgetc();
17       int sgetn(charT*, int N);
18 };
```

Traits Example: The Solution [myers] II

```
1  struct ios_char_traits<char> {
2      typedef char char_type;
3      typedef int  int_type;
4      static inline int_type eof() { return EOF; }
5  };
6
7  struct ios_char_traits<wchar_t> {
8      typedef wchar_t char_type;
9      typedef wint_t  int_type;
10     static inline int_type eof() { return WEOF; }
11 };
```

```
1  int main(int argc, char *argv[]) {
2      ios_char_traits<char>     iosAscii;
3      ios_char_traits<wchar_t>  iosUTF;
4      ...
5  }
```

Polymorphic Return Values [cgal] I

```
1  {
2      typedef Cartesian<double>  K;
3      typedef K::Point_2         Point_2;
4      typedef K::Segment_2       Segment_2;
5
6      Segment_2 segment_1, segment_2;
7
8      std::cin >> segment_1 >> segment_2;
9
10     Object obj = intersection(segment_1, segment_2);
11
12     if (const Point_2 *point = object_cast<Point_2>(&obj)) {
13         /* do something with *point */
14     } else if (const Segment_2 *segment = object_cast<Segment_2>(&obj)) {
15         /* do something with *segment*/
16     }
17
18     /*  there was no intersection */
19 }
```

Polymorphic Return Values [cgal] II

```
1  template < class Kernel >
2  Object  intersection(Segment_2<Kernel> s1, Segment_2<Kernel> s2) {
3      if (/* intersection in a point */ ) {
4         Point_2<Kernel> p = ... ;
5         return make_object(p);
6      } else if (/* intersection in a segment */ ) {
7         Segment_2<Kernel> s = ... ;
8         return make_object(s);
9      }
10     return Object();
11 }
```

Sources


CGAL, *Computational Geometry Algorithms Library*
URL: http://www.cgal.org/


Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, Andrew P. Black, *Traits: Composable Units of Behaviour*
URL: http://link.springer.com/chapter/10.1007%2F978-3-540-45070-2_12


Wikipedia, *Traits class*
URL: http://en.wikipedia.org/wiki/Traits_class


Nathan C. Myers, *Traits: a new and useful template technique*
URL: http://www.cantrip.org/traits.html


Cathy Mazur, *Habitat of a seagull*
URL: http://stanleyseagull.blogspot.cz/2011/08/habitat-of-seagull.html


Le Groupe De Ouf, *Au fil du temps*
URL:
http://club.quomodo.com/groupedeouf/l_histoire_du_groupe/au_fil_du_temps

OI-OPPA. European Social Fund
Prague & EU: We invest in your future.