Geometric Computing: Introduction to CGAL

Panos Giannopoulos, Dror Atariah
AG TI

WS 2012/13
!! Register in Campus Management !!
Outline

What you need to know

What is the course about?

Course structure

Let’s start!

Exercise 1
You need to know (or learn quickly)

- **C++** (templates, STL iterators, containers, generic algorithms, etc.)

  ```
  template< class InputIterator, class T >
  InputIterator find( InputIterator first, InputIterator last, const T& value );
  ```

  or

  ```
  std::vector<int> v;
  ...
  for(std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    ...
  }
  ```

- **Algorithms** (run-time analysis, $\mathcal{O}$-notation, basic algorithmic techniques and data structures, etc.)

- **Basic geometry**
This course is about

- Computing with geometric objects (points, lines, segments, curves, planes, etc.)
- Implementing and using geometric algorithms and data structures (efficiently and correctly)
- Using the Computational Geometry Algorithms Library (CGAL) (together with STL, Boost, Qt, etc.)
  - Basic packages: Convex Hull, Arrangements, Triangulations, Voronoi Diagrams, etc.
- Having Fun!
This course is about (for example) intersection point, Arrangement cell, and query point.
This course is about (for example)

- Bounding Volumes
- Triangulations
- Surface Reconstruction
Course structure

- Lectures: 19, 26 Oct., 2 Nov., then every second week
- Be present in the lectures!
- Bring your laptop
- One concept, package, algorithm per lecture (approximately). Then we’ll work on exercises
- Exercises will gradually become more involved. By the end of Nov. - beginning Dec. each team should choose some project
- Make teams of 1-3 people
- Suggestion of projects also welcome!
- You’ll be graded for the project
Let’s start!

(Many thanks to Efi Fogel from TAU, IL, for sharing the next slides)
Geometric Computing: The Goal

(Re)design and implement geometric algorithms and data structures that are at once certified and efficient in practice.
Geometric Computing: The Assumptions

- Input data is in general position
  - Degenerate input, e.g., three curves intersecting at a common point, is precluded.
- Computational model: the real RAM
  - Operations on real numbers yield accurate results.
- Each basic operation on a small (constant-size) set of simple objects takes unit time.
Geometric Computing: The Problems

These assumptions often do not hold in practice

- Degenerate input is commonplace in practical applications.
- Numerical errors are inevitable while using standard computer arithmetic.
- Naive use of floating-point arithmetic causes geometric programs to:
  - Crash after invariant violation
  - Enter an infinite loop
  - Produce wrong output
- There is a gap between Geometry in theory and Geometry with floating-point arithmetic.
  - Standard cs-theory asymptotic performance measures many times poor predictors of practical performance.
Geometry in Theory

\[
\text{orientation}(p, q, r) = \text{sign} \left( \det \begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix} \right) = \begin{cases} < 0 & \text{cw}(p, q, r) \\ = 0 & \text{colinear}(p, q, r) \\ > 0 & \text{ccw}(p, q, r) \end{cases}
\]

\[
= \text{sign}((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x))
\]

\[
\text{ccw}(p, q, s) \cap \text{ccw}(s, q, r) \cap \text{ccw}(p, s, r) \Rightarrow \text{ccw}(p, q, r)
\]
Geometry in Practice: Trouble with Double

orientation\((p, q, r) = \text{sign}\((q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)\))

256 × 256 pixel image

Positive  Zero  Negative
The Naive Solution: Exact Multi-Precision Arithmetic

- Implemented for several number types:
  - Integers, rational (e.g., GMP, CORE, and LEDA)
  - Even algebraic numbers (e.g., CORE and LEDA)
  - No solution for transcendental numbers!

- Exact up to memory limit.
- Slow running time.
The Efficient Solution: Exact Geometric Computation

Ensure that the control flow in the implementation corresponds to the control flow with exact arithmetic. [Yap04]

- Evaluate predicate instantiated with limited precision.
- If uncertain ⇒ evaluate predicate instantiated with multiple precision.

orientation(p, q, r) < 0 = 0 > 0
Floating-Point Arithmetic

- A double float $f$ uses 64 bits
  - 1 bit for the sign $s$.
  - 52 bits for the mantissa $m = m_1 \ldots m_{52}$.
  - 11 bits for the exponent $e = e_1 \ldots e_{52}$.

- $f = -1^s \cdot (1 + \sum_{1 \leq i \leq 52} m_i 2^{-i}) \cdot 2^{e-2013}$, if $0 < e < 2^{11} - 1$

...
Geometric-Computing Bibliography

Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee K. Yap.
Classroom Examples of Robustness Problems in Geometric Computations.

Chee K. Yap.
Robust geometric computation.

Kurt Mehlhorn and Stefan Näher.
Generic Programming Paradigm

**Definition (Generic Programming)**

A discipline that consists of the gradual lifting of concrete algorithms abstracting over details, while retaining the algorithm semantics and efficiency.

[MS88]
Generic Programming Paradigm

Definition (Generic Programming)
A discipline that consists of the gradual lifting of concrete algorithms abstracting over details, while retaining the algorithm semantics and efficiency.

[MS88]

Translation:
- You do not want to write the same algorithm again and again!
Generic Programming Paradigm

**Definition (Generic Programming)**

A discipline that consists of the gradual lifting of concrete algorithms abstracting over details, while retaining the algorithm semantics and efficiency.

[MS88]

Translation:

- You do not want to write the same algorithm again and again!
- You even want to make it independent from the used types.

See also: [http://en.wikipedia.org/wiki/Generic_programming](http://en.wikipedia.org/wiki/Generic_programming)
Terms and Definitions

Class Template  A specification for generating (instantiating) classes based on parameters.

Function Template  A specification for generating (instantiating) functions based on parameters.

Template Parameter

Specialization  A particular instantiation from a template for a given set of template parameters.
Generic Programming Dictionary

**Concept** A set of requirements that a class must fulfill.

**Model** A class that fulfills the requirements of a concept.

**Traits** Models that describe behaviors.

**Refinement** An extension of the requirements of another concept.

**Generalization** A reduction of the requirements of another concept.
Some Generic Programming Libraries

**STL** The C++ Standard Template Library.

**Boost** A large set of portable and high quality C++ libraries that work well with, and are in the same spirit as, the C++ STL.

**LEDA** The Library of efficient data types and algorithms.

**CGAL** The computational geometry algorithms and data structures library.
STL Components

**Container** A class template, an instance of which stores collection of objects.

**Iterator** Generalization of pointers; an object that points to another object.

**Algorithm**

**Function Object (Functor)** A computer programming construct invoked as though it were an ordinary function.

**Adaptor** A type that transforms the interface of other types.

**Allocator** An objects for allocating space.
template< class InputIterator, class T >
InputIterator find( InputIterator first, InputIterator last, const T& value );

or

std::vector<int> v;
...
for(std::vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
  ...
}
Generic Algorithms

- A generic algorithm has 2 parts:
  - The actual instructions that describe the steps of the algorithm.
  - A set of requirements that specify which properties its argument types
A Trivial Example: `swap()`

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

- When a function call is compiled the function template is instantiated.
- The template parameter `T` is substituted with a data type.
- The data type must have
  1. a copy constructor, and
  2. an assignment operator.
A Trivial Example: `swap()`

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

- When a function call is compiled the function template is instantiated.
- The template parameter `T` is substituted with a data type.
- The data type must have
  1. a copy constructor, and
  2. an assignment operator.

In formal words:
- `T` is a model of the concept `CopyConstructible`.
- `T` is a model of the concept `Assignable`.
A Trivial Example: `swap()`

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

- When a function call is compiled the function template is instantiated.
- The template parameter `T` is substituted with a data type.
- The data type must have
  1. a copy constructor, and
  2. an assignment operator.

In formal words:
- `T` is a model of the concept `CopyConstructible`.
- `T` is a model of the concept `Assignable`.

The `int` data type is a model of the 2 concepts.

```cpp
int a = 2, b = 4; std::swap(a, b);
```
Concept

A concept is a set of requirements divided into four categories:
Concept

A concept is a set of requirements divided into four categories:

**Associated Types** — auxiliary types, for example

- `Point_2` — a type that represents a two-dimensional point.
Concept

A concept is a set of requirements divided into four categories:

**Associated Types** — auxiliary types, for example
  - `Point_2` — a type that represents a two-dimensional point.

**Valid Expressions** — C++ expressions that must compile successfully, for example
  - `p = q`, where `p` and `q` are objects of type `Point_2`. 
Concept

A concept is a set of requirements divided into four categories:

**Associated Types** — auxiliary types, for example
- \texttt{Point\_2} — a type that represents a two-dimensional point.

**Valid Expressions** — \texttt{C++} expressions that must compile successfully, for example
- \( p = q \), where \( p \) and \( q \) are objects of type \texttt{Point\_2}.

**Runtime Characteristics** — characteristics of the variables involved in the valid expressions that apply during the variables’ lifespans,
- pre/post-conditions.
Concept

A concept is a set of requirements divided into four categories:

**Associated Types** — auxiliary types, for example
- \texttt{Point\_2} — a type that represents a two-dimensional point.

**Valid Expressions** — C++ expressions that must compile successfully, for example
- \( p = q \), where \( p \) and \( q \) are objects of type \texttt{Point\_2}.

**Runtime Characteristics** — characteristics of the variables involved in the valid expressions that apply during the variables’ lifespans,
- pre/post-conditions.

**Complexity Guarantees** — maximum limits on the computing resources consumed by the valid expressions.
Generic Programming & the **STL** Bibliography

- **Andrei Alexandrescu.**
  *Modern C++ Design: Generic Programming And Design Patterns Applied.*
  Addison-Wesley, Boston, MA, USA, 2001.

- **Matthew H. Austern.**
  *Generic Programming and the STL.*
  Addison-Wesley, Boston, MA, USA, 1999.

- **Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides.**
  *Design Patterns — Elements of Reusable Object-Oriented Software.*
  Addison-Wesley, Boston, MA, USA, 1995.

- **David R. Musser, Gillmer J. Derge, and Atul Saini.**
  *STL tutorial and reference guide: C++ programming with the standard template library.*

- **David Vandevoorde and Nicolai M. Josuttis.**
  *C++ Templates: The Complete Guide,*
  Addison-Wesley, Boston, MA, USA, 2002.

- **Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine.**
  *The Boost Graph Library,*
  Addison-Wesley, Boston, MA, USA, 2002.

- **David A. Musser and Alexander A. Stepanov.**
  Generic programming.

- **The planned new standard for the C++ programming language.**
“Make the large body of geometric algorithms developed in the field of computational geometry available for industrial applications”

*CGAL* Project Proposal, 1996
Some of CGAL Content

Bounding Volumes
Polyhedral Surfaces
Boolean Operations
Triangulations
Voronoi Diagrams
Mesh Generation
Subdivision
Simplification
Parametrisation
Streamlines
Ridge Detection
Neighbor Search
Kinetic Data Structures

Envelopes
Arrangements
Intersection Detection
Minkowski Sums
PCA
Polytope Distance
QP Solver

FU Berlin, Geometric Computing: Introduction to CGAL, WS 2012/13
● Written in C++
● Follows the *generic programming* paradigm
● Development started in 1995
● Active European sites:
  1. INRIA Sophia Antipolis
  2. MPII Saarbrücken
  3. Tel Aviv University
  4. ETH Zürich (Plageo)
  5. University of Crete and FO.R.T.H.
  6. INRIA Nancy
  7. Université Claude Bernard de Lyon
  8. ENS Paris
  9. University of Eindhoven
  10. University of California, San Francisco
  11. University of Athens
900,000 lines of C++ code
10,000 downloads per year not including Linux distributions
3,500 manual pages
3,000 subscribers to cgal-announce list
1,000 subscribers to cgal-discuss list
120 packages
60 commercial users
25 active developers
6 months release cycle
7 Google’s page rank for cgal.org.com
2 licenses: Open Source and commercial
- Reliability
  - Explicitly handles degeneracies
  - Follows the Exact Geometric Computation (EGC) paradigm

- Flexibility
  - Is an open library
  - Depends on other libraries (e.g., Boost, Gmp, Mpfr, Qt, & Core)
  - Has a modular structure, e.g., geometry and topology are separated
  - Is adaptable to user code
  - Is extensible, e.g., data structures can be extended

- Ease of Use
  - Has didactic and exhaustive Manuals
  - Follows standard concepts (e.g., C++ and STL)
  - Characterizes with a smooth learning-curve

- Efficiency
  - Adheres to the generic-programming paradigm
    - Polymorphism is resolved at compile time
CGAL Structure

Basic Library

Algorithms and Data Structures
e.g., Triangulations, Surfaces, and Arrangements

Kernel

Elementary geometric objects
Elementary geometric computations on them

Support Library

Configurations, Assertions,...

Visualization

Files
I/O
Number Types
Generators
...
CGAL Kernel Concept

- Geometric objects of constant size.
- Geometric operations on object of constant size.

<table>
<thead>
<tr>
<th>Primitives 2D, 3D, dD</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicates</td>
</tr>
<tr>
<td>point</td>
<td>comparison</td>
</tr>
<tr>
<td></td>
<td>orientation</td>
</tr>
<tr>
<td>vector</td>
<td></td>
</tr>
<tr>
<td>triangle</td>
<td>containment</td>
</tr>
<tr>
<td>iso rectangle</td>
<td></td>
</tr>
<tr>
<td>circle</td>
<td></td>
</tr>
<tr>
<td>. . .</td>
<td></td>
</tr>
</tbody>
</table>
**CGAL Kernel Affine Geometry**

- `point - origin` → `vector`
- `point - point` → `vector`
- `point + vector` → `point`

**Illegal**

- `point + point` ← Illegal

- `midpoint(a, b) = a + 1/2 \times (b - a)`
**CGAL Kernel Classification**

- **Dimension**: 2, 3, arbitrary
- **Number types**:
  - Ring: +, −, ×
  - Euclidean ring (adds integer division and gcd) (e.g., CGAL::Gmpz).
  - Field: +, −, ×, / (e.g., CGAL::Quotient(CGAL::Gmpz)).
  - Exact sign evaluation for expressions with roots (Field_with_sqr).
- **Coordinate representation**
  - Cartesian — requires a field number type or Euclidean ring if no constructions are performed.
  - Homogeneous — requires Euclidean ring.
- **Reference counting**
  - Exact, Filtered
**CGAL Kernels and Number Types**

<table>
<thead>
<tr>
<th>Cartesian representation</th>
<th>Homogeneous representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>$x = \frac{hx}{hw}$</td>
</tr>
<tr>
<td></td>
<td>$y = \frac{hy}{hw}$</td>
</tr>
<tr>
<td></td>
<td>point</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Intersection of two lines**

\[
\begin{align*}
\left\{\begin{array}{l}
a_1x + b_1y + c_1 = 0 \\
a_2x + b_2y + c_2 = 0
\end{array}\right.
\end{align*}
\]

\[
\begin{align*}
\left\{\begin{array}{l}
a_1hx + b_1hy + c_1hw = 0 \\
a_2hx + b_2hy + c_2hw = 0
\end{array}\right.
\end{align*}
\]

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

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

**Field operations**

**Ring operations**
Example: Kernels $\langle$NumberType$\rangle$

- **Cartesian** $\langle$FieldNumberType$\rangle$
  - `typedef CGAL::Cartesian<Gmpq> Kernel;`
  - `typedef CGAL::Simple_cartesian<double> Kernel;`
  - ★ No reference-counting, inexact instantiation

- **Homogeneous** $\langle$RingNumberType$\rangle$
  - `typedef CGAL::Homogeneous<Core::BigInt> Kernel;`

- d-dimensional **Cartesian** $\_d$ and **Homogeneous** $\_d$

- **Types + Operations**
  - `Kernel::Point_2`, `Kernel::Segment_3`
  - `Kernel::Less_xy_2`, `Kernel::Construct_bisector_3`
CGAL Numerical Issues

```cpp
typedef CGAL::Cartesian<NT> Kernel;
NT sqrt2 = sqrt(NT(2));

Kernel::Point_2 p(0,0), q(sqrt2, sqrt2);
Kernel::Circle_2 C(p,4);
assert(C.has_on_boundary(q));
```

- OK if NT supports exact sqrt.
- **Assertion violation** otherwise.
Computing the Intersection

typedef Kernel::Line_2 Line_2;

int main() {
    Kernel kernel;
    Point_2 p(1,1), q(2,3), r(-12,19);
    Line_2 l1(p,q), l2(r,p);
    if (do_intersect(l1, l2)) {
        CGAL::Object obj = CGAL::intersection(l1, l2);
        if (const Point_2* point = object_cast<Point_2>(&obj)) {
            /* do something with point */
        } else if (const Segment_2* segment = object_cast<Segment_2>(&obj)) {
            /* do something with segment */
        }
    }
    return 0;
}
CGAL Basic Library

- Generic data structures are parameterized with Traits
  - Separates algorithms and data structures from the geometric kernel.
- Generic algorithms are parameterized with iterator ranges
  - Decouples the algorithm from the data structure.
CGAL Bibliography

A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr.
On the design of 

On the design of CGAL a computational geometry algorithms library.

A. Fabri and S. Pion.
A generic lazy evaluation scheme for exact geometric computations.
In 2nd Library-Centric Software Design Workshop, 2006.

M. H. Overmars.
Designing the computational geometry algorithms library CGAL.

The CGAL Project.

Efi Fogel, Ron Wein, and Dan Halperin.
CGAL Arrangements and Their Applications, A Step-by-Step Guide.
Exercise 1

1. Install CGAL: www.cgal.org
   (Installation tips also on the course’s website)

2. Read CGAL’s manual: Chapters 1 (Introduction), 2 (Preliminaries), 3 (Installation), 5 (Number Types), 11 (2D and 3D Geometry Kernel)

3. For the following use the Cartesian kernel with number type double. Use only geometric primitives from the kernel. Do not use any kernel operation (predicate or construction).
   ▶ Write a function that takes two segments in the plane as arguments and returns: 1 if the segments intersect, 0 if they don’t.
   ▶ Write a function that takes two segments in the plane as arguments and computes their intersection (if it exists).
   ▶ Write a function that takes a point and a segment in the plane as arguments and returns 1 if the point lies on the segment and 0 otherwise.
   ▶ Test your functions.