Introduction to CGAL

Constantinos Tsirogiannis

TU/Eindhoven
CGAL = Computational Geometry Algorithms Library:
CGAL = Computational Geometry Algorithms Library:

- A library of Geometric algorithms and data structures.
CGAL = Computational Geometry Algorithms Library:

- A library of Geometric algorithms and data structures.
- Written in C++
CGAL = Computational Geometry Algorithms Library:

- A library of Geometric algorithms and data structures.
- Written in C++
- Uses template programming
C++ = C + Object Oriented Programming
C++ = C + Object Oriented Programming

Traditional C code:
C++ = C + Object Oriented Programming

Traditional C code:

```c
int a=5;
int b=3;
add_function(a,b);
```
- **C++ code:**

  ```
  int a = 5;
  int b = 3;
  Example_class foo;
  foo.add(a, b);
  ```
Here follow few basic concepts of C++
Class:

```cpp
class Example
{
public:
    int add(int a, int b) { return a + b; }
    ...
private:
    int k;
};
```
Class:

class Example_class
{
    public:
        int add( int a, int b)
        {
            return a+b;
        }
    ...
    private:
        int k;
};
Structures: same as classes, yet everything is by default visible ("public")
Namespaces

- Namespace: Can be seen as a “big bag” containing declarations of class types and functions.
Namespaces

- Namespace: Can be seen as a “big bag” containing declarations of class types and functions.

- Example:

```cpp
namespace FOO
{
    class something{...};
    class another_class{...};
    struct whatever{...};
    int add(int a, int b){a+b;}
}
```
To access the type `something` from a namespace `FOO` try:

`FOO::something` or `typename FOO::something`
Namespace std

- To access the type *something* from a namespace FOO try:
  
  FOO::something or typename FOO::something

- Namespace std contains many helpful classes and functions of the Standard Library of C++.
Containers

- Nice and easy classes to insert and manipulate sets of objects.

Example: Vectors

```cpp
std::vector<int> integers;
integers.push_back(4);
integers.push_back(-2);
integers.push_back(9);
integers.push_back(3);
for (int i = 0; i < integers.size(); i++)
    std::cout << integers[i] << std::endl;
```
Containers

- Nice and easy classes to insert and manipulate sets of objects.
- Example: Vectors

```cpp
std::vector<int> integers;
integers.push_back(4);
integers.push_back(-2);
integers.push_back(9);
integers.push_back(3);
for( int i=0; i< integers.size(); i++ )
    std::cout << integers[i] << std::endl;
```
An iterator is a substitute of a pointer.
Iterators

- An iterator is a substitute of a pointer.
- Iterators are used to go through the elements of a container or some other kind of range.

Example:

```cpp
std::vector<int> integers;
integers.push_back(4);
integers.push_back(-2);
integers.push_back(9);
integers.push_back(3);
for (typename std::vector<int>::iterator it = integers.begin(); it < integers.end(); it++)
    std::cout << *it << std::endl;
```
An **iterator** is a substitute of a pointer.

Iterators are used to go through the elements of a container or some other kind of **range**.

Example:

```cpp
std::vector<int> integers;
integers.push_back(4);
integers.push_back(-2);
integers.push_back(9);
integers.push_back(3);
for (typename std::vector<int>::iterator it = integers.begin(); it < integers.end(); it++)
    std::cout << *it << std::endl;
```
In the beginning of your .cpp file put:

```cpp
#include <fstream>
```
In the beginning of your .cpp file put:

```cpp
#include <fstream>
```

To read a set of integers from a file:

```cpp
std::ifstream in('filename.txt');
std::istream_iterator<int> begin(in);
std::istream_iterator<int> end;
std::vector<int> integers;

integers.insert(integers.begin(), begin, end);
```
Example:

```cpp
std::ofstream os("filename.cout");
std::vector<int> integers;
integers.push_back(4);
integers.push_back(-2);
integers.push_back(9);
integers.push_back(3);
for( typename std::vector<int>::iterator it=integers.begin();
it< integers.end(); it++ )
os << *it << std::endl;
```
A template is a class or function that one can choose the types that it contains/manipulates.
A template is a class or function that one can choose the types that it contains/manipulates.

Example: A function that computes a power of a given integer.

```c
int power (int base, int exponent)
{
    int res=1;
    for( int i=1; i<=exponent; i++ )
        res = res * base;
    return res;
}
```
Example: A function that computes a power of a given object of type Type.

```cpp
template <class Type>
Type power (Type base, int exponent)
{
    Type res=1;
    for( int i=1; i<=exponent; i++ )
        res = res * base;
    return res;
}
```
CGAL provides implementations of:
CGAL provides implementations of:

- Triangulations.
CGAL provides implementations of:

- Triangulations.
- Voronoi diagrams.
CGAL provides implementations of:

- Triangulations.
- Voronoi diagrams.
- Arrangements.
CGAL provides implementations of:

- Triangulations.
- Voronoi diagrams.
- Arrangements.
- Kinetic Data Structures.
CGAL provides implementations of:

- Triangulations.
- Voronoi diagrams.
- Arrangements.
- Kinetic Data Structures.
- . . .
The concepts provided by CGAL are categorized as:

- Geometric Objects
- Geometric Predicates/Constructions
- Geometric Algorithms/Data-Structures
The concepts provided by CGAL are categorized as:

- Geometric Objects
CGAL Concepts

The concepts provided by CGAL are categorized as:

- Geometric Objects
- Geometric Predicates/Constructions
The concepts provided by CGAL are categorized as:

- Geometric Objects
- Geometric Predicates/Constructions
- Geometric Algorithms/Data-Structures
Geometric objects and predicates appear inside a kernel.
- Geometric objects and predicates appear inside a kernel.
- A kernel is more or less a large class the encapsulates objects and predicates that fall in the same general category.
Geometric objects and predicates appear inside a kernel.

A kernel is more or less a large class the encapsulates objects and predicates that fall in the same general category.

```cpp
struct Cartesian_kernel
{
    class Point {...};
    class Segment {...};
    class Triangle {...};
    ...
    class Do_intersect_predicate {...};
};
```
A kernel is in fact a template. The template parameter is a number type.
A kernel is in fact a template. The template parameter is a number type.

template <typename Num_type>
struct Cartesian_kernel
{
    class Point<Num_type> {...};
    class Segment<Num_type> {...};
    class Triangle<Num_type> {...};
    ...
    class Do_intersect_pred<Num_type> {...};
};
CGAL provides many different number types:

- **Built in**: int, double.
CGAL Number types

CGAL provides many different number types:

- **Built in**: int, double.
- **Arbitrary precision rationals**: Gmpq, MP_Float.
CGAL provides many different number types:

- **Built in:** `int`, `double`.
- **Arbitrary precision rationals:** `Gmpq`, `MP_Float`.
- **Algebraic numbers:** `Root_of_2<Type>`.
Kernels

- You can use kernels with fixed number types such as Cartesian<double>.
Kernels

• Or you can use predefined “Filtered Kernels” that provide exact predicates, and even exact constructions, if desired.

• Filtered Kernels use less computationally intensive number types as default, and fall back to exact computation when required.

• Exact_predicates_exact_constructions_kernel

• Exact_predicates_inexact_constructions_kernel
• The main algorithms and data-structures appear outside the kernels.
The main algorithms and data-structures appear outside the kernels.

They are also templates.
The main algorithms and data-structures appear outside the kernels.

They are also templates.

Example:

```cpp
template <typename Kernel>
struct Triangulation_2
{
    typedef typename Kernel::Point Point;
    typedef typename Kernel::Segment Segment;
    typedef typename Kernel::Triangle Triangle;
    ...
};
```
Let’s have a close look on the example program.
#include <CGAL/basic.h>
#include <CGAL/intersections.h>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>

#include <qapplication.h>
#include <qmainwindow.h>

#include <CGAL/IO.Qt_widget.h>
#include <CGAL/IO.Qt_widget_standard_toolbar.h>
typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point;
typedef Kernel::Segment_2 Segment;
typedef Kernel::Line_2 Line;
Point a, b, c, d;
Segment s;
Line l;
CGAL::Object o;
class My_window : public QMainWindow {
    Q_OBJECT
public:
    My_window()
    {
        ............... //private slots
    }

private slots: //functions
void redraw_win()
{
    ............... //private
}

private: //members
    CGAL::Qt_widget *widget;
    CGAL::Qt_widget_standard_toolbar *std_toolbar;
};
My_window()
{
    widget = new CGAL::Qt_widget(this);
    setCentralWidget(widget);
    resize(3,3);
    widget->show();
    widget->set_window(-1, 2, -1, 2);

    a = Point(0,0);
    b = Point(1,0);
    c = Point(1,1);
    d = Point(0,1);

    s = Segment(a,c);
    l = Line(b,d);

    o = CGAL::intersection(s, l);

    //How to attach the standard toolbar
    std_toolbar = new CGAL::Qt_widget_standard_toolbar(widget, this, "Standard Toolbar");

    connect(widget, SIGNAL(redraw_on_back()),
            this, SLOT(redraw_win()));
}
void redraw_win()
{
    *widget << a << b << c << d << CGAL::BLUE << s << l << CGAL::RED;
    if (const Point *ipoint = CGAL::object_cast<Point>(&o)) {
        // handle the point intersection case with *ipoint.
        *widget << *ipoint;
    } else if (const Segment *iseg = CGAL::object_cast<Segment>(&o)) {
        // handle the segment intersection case with *iseg.
        *widget << *iseg;
    } else {
        // handle the no intersection case.
    }
}
```c
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    My_window W();
    app.setMainWidget( &W );
    W.show();
    W.setCaption("Using the Standard Toolbar");
    return app.exec();
}
```
Version of the program using new framework:
#include <CGAL/basic.h>
#include <CGAL/intersections.h>
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/number_utils.h>

#include <iostream>
#include <boost/format.hpp>
#include <QtGui>
#include <CGAL/Qt/GraphicsViewNavigation.h>
typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef Kernel::Point_2 Point;
typedef Kernel::Segment_2 Segment;
typedef Kernel::Line_2 Line;
```cpp
int main(int argc, char **argv)
{
    QApplication app(argc, argv);

    QGraphicsScene scene;

    Point a(0,0), b(1,0), c(1,1), d(0,1);
    Segment s(a,c);
    Line l(b,d);

    CGAL::Object o = CGAL::intersection(s,l);

    scene.setSceneRect(-1, -1, 2, 2);

    scene.addEllipse(CGAL::to_double(a.x()) - 0.01, CGAL::to_double(a.y()) - 0.01, 0.02, 0.02);
    scene.addEllipse(CGAL::to_double(b.x()) - 0.01, CGAL::to_double(b.y()) - 0.01, 0.02, 0.02);
    scene.addEllipse(CGAL::to_double(c.x()) - 0.01, CGAL::to_double(c.y()) - 0.01, 0.02, 0.02);
    scene.addEllipse(CGAL::to_double(d.x()) - 0.01, CGAL::to_double(d.y()) - 0.01, 0.02, 0.02);

    scene.addLine(CGAL::to_double(s.source().x()), CGAL::to_double(s.source().y()),
                  CGAL::to_double(s.target().x()), CGAL::to_double(s.target().y()), QPen(Qt::blue));

    scene.addLine(-2, CGAL::to_double(l.y_at_x(-2)),
                  3, CGAL::to_double(l.y_at_x(3)), QPen(Qt::blue));

    if (const Point *ipoint = CGAL::object_cast<Point>(&o)) {
        // handle the point intersection case with *ipoint.
        scene.addEllipse(CGAL::to_double(ipoint->x()) - 0.01, CGAL::to_double(ipoint->y()) - 0.01, 0.02, 0.02, QPen(),
                         QBrush(Qt::red));
    } else if (const Segment *iseg = CGAL::object_cast<Segment>(&o)) {
        // handle the segment intersection case with *iseg.
        scene.addLine(CGAL::to_double(iseg->source().x()), CGAL::to_double(iseg->source().y()),
                      CGAL::to_double(iseg->target().x()), CGAL::to_double(iseg->target().y()), QPen(Qt::red));
    } else {
        // handle the no intersection case.
    }

    QGraphicsView* view = new QGraphicsView(&scene);
    CGAL::Qt::GraphicsViewNavigation navigation;
    view->installEventFilter(&navigation);
    view->viewport()->installEventFilter(&navigation);
    view->setRenderHint(QPainter::Antialiasing);

    view->show();
    return app.exec();
}
```
Arrangement DCEL

#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Arr_segment_traits_2.h>
#include <CGAL/Arrangement_2.h>
#include <CGAL/Arr_default_dcel.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef CGAL::Arr_segment_traits_2<Kernel> Traits_2;
typedef CGAL::Arr_default_dcel<Traits_2> ArrangementDcel;

Installing CGAL

- Prerequisites: cmake, boost, GMP, MPFR, Qt3 and/or Qt4.
- cmake-gui
- make
- sudo make install
Tips on using CGAL

• Use predefined filtered kernels: These define your geometric objects and predicates.

• Documentation can be a hit or a miss, when in doubt, look at example code, which is provided in the CGAL directory.
Further Reference

- http://www.cgal.org/mailing_list.html