Computational Geometry Algorithms Library

http://parasol.tamu.edu
• Find the closest emergency phone on campus?
• How I can go from HRBB to Reed arena?

http://parasol.tamu.edu/CampusNavigator/
• Find the closest emergency phone on campus?
• How I can go from HRBB to Reed arena?
• I want go to HEB, but I don't have car, what I need to do?
• Is the marked point inside of the polygon?
Computational Geometry

• A branch of computer science devoted to the study of algorithms which can be stated in terms of geometry.[1]

• Systematic study of algorithms and data structures for geometric objects, with a focus on exact algorithms that are asymptotically fast.[2]

Applications of Computational Geometry

- Computer Graphics
- Computer-Aided Design and Manufacturing
- Robotics
- Geographic Information Systems
- Integrated Circuit design
- Computer-Aided Engineering
Convex Hull
Convex Hull
Convex Hull
Triangulation
Triangulation
Delaunay Triangulation
Delaunay Triangulation
Delaunay Triangulation
Voronoi diagram
Voronoi diagram
Voronoi diagram
CGAL (Computational Geometry Algorithms Library)

- The goal of CGAL Open Source project, is to provide easy access to efficient and reliable geometric algorithms in the form of C++ libraries.

- Offers data structures and algorithms, all of these operate on geometric objects like points and segments, and perform geometric tests on them.

http://www.cgal.org/
Structure of CGAL

**Basic Library**
Algorithm and Data Structures

**Kernel**
Geometric Objects
Geometric Operations

**Core Library**
Configurations, assertions, ...

**Support Library**
Visualization
File
I/O
NumberTypes
Generators
...
CGAL Kernel

- Elementary geometric objects
- Elementary computation of them

- Primitives 2D, 3D, dD
  - Point, Vector, Triangle, Iso_Rectangle, Circle, ...

- Predicates
  - Comparison, Orientation, InSphere, ...

- Constructions
  - Intersection, square distance, ...
Kernel and NumberTypes

- Cartesian representation
  \[ Point: (x,y) \]
  
  \[
  \text{CGAL::Cartesian<FT>}
  \]
  \[
  \text{double, Quotient<Gmpz>, leda_real}
  \]

- Homogeneous representation
  \[ Point: (hx,hy,hw) \]
  
  \[
  \text{CGAL::Homogeneous<RT>}
  \]
  \[
  \text{int, Gmpz, double}
  \]

\[
\text{typedef double NumberType;}
\]
\[
\text{typedef Cartesian< NumberType > Kernel;}
\]
\[
\text{typedef Kernel::Point_2 Point;}
\]
Kernel and NumberTypes

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);
assert( C.has_on_boundary(q) );

Assertion if NT doesn't gives an exact sqrt

CGAL Predicates & Constructions

**Predicates**

Orientation

in_circle

**Constructions**

Intersection

circumcenter

---

CGAL Basic Library

Bounding Volumes
Polyhedral Surface
Boolean Operations

Triangulations
Voronoi Diagrams
Mesh Generation

Subdivision
Simplification
Parametrisation
Streamlines

Ridge Detection
Neighbor Search
Kinetic Datastructures

Lower Envelope
Arrangement
Intersection Detection
Minkowski Sum

PCA
Polytope distance
QP Solver
CGAL Basic Library

- Convex Hull (5 alg. In 2D and 3 alg. In 3D)
- 2D/3D Triangle/Tetrahedron based data-structure
- 2D/3D Delaunay triangulation
- 2D/3D Regular Triangulation
- 2D Voronoi Diagram
- 2D Meshes

CGAL Basic Library

- Polyhedra data-structure
  - Half-edge
  - Polyhedral surface
  - 2D Nef polygons
  - 3D Nef polyhedra

- Geometric Optimizations
  - Smallest enclosing circle and ellipse in 2D
  - Smallest enclosing sphere in dD
  - Largest empty rectangle
  - ...

CGAL Basic Library

- Line segments or polylines
- Search structures
  - Range-tree, Segment-tree, kD-tree
  - Window query
  - Approximate nearest neighbors
  - ...

Installation prerequisites

- Compilers
  - GNU g++ 4.0, 4.1, 4.2, 4.3
  - MS Visual C++ 8.0, 9.0
Prerequisites

Optional

- GMP+MPFR and LEDA: Exact arithmetic

- Visualization
  - Trolltech's cross-platform GUI toolkits Qt3 or Qt4
  - Geomview
Installation

- The CGAL library can be downloaded from http://www.cgal.org/download.html

- Unpack source code:
  
  `tar xzf CGAL-3.8.tar.gz`

- Windows installer: CGAL-3.8-Setup.exe

- CGAL manual must be downloaded separately from http://www.cgal.org/download.html
Directories

- **config**: configuration files for install script
- **cmake/modules**: modules for finding and using libraries
- **demo**: demo programs (most of them need QT, geomview or other third-party products)
- **doc_html**: Installation documentation (HTML)
- **examples**: example programs
- **Include**: header files
- **scripts**: some useful scripts (e.g. for creating CMakeLists.txt files)
Configure and Build CGAL

- Using cmake GUI
  
  cd CGAL-3.8
  
  cmake-gui .
  
  make

- Using cmake command line
  
  cd CGAL-3.8
  
  cmake .
  
  make
Installing CGAL

- After configure and build
  
  make install

- By default: /usr/local

- Change CMAKE_INSTALL_PREFIX to the preferred location to install
Compile examples

- Compile the demos and examples
  
  cd examples/Core
  cmake .
  make

- To generate verbose makefile (show executed command) use
  
  cmake -DCMAKE_VERBOSE_MAKEFILE=ON .
Simple example

```cpp
#include <iostream>
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/convex_hull_2.h>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;

typedef K::Point_2 Point_2;

int main() {
    Point_2 points[5] = { Point_2(0,0), Point_2(10,0), Point_2(10,10),
                          Point_2(6,5), Point_2(4,1) };

    std::vector<Point_2> result;

    CGAL::convex_hull_2( points, points+5, std::back_inserter(result) );

    std::cout << result.size() << " points on the convex hull\n";

    return 0;
}
```
Compile simple example

- Without cmake:
  ```
g++ -frounding-math -O3 -o simple_example -lCGAL simple_example.cpp
  ```

- Using cmake:
  ```
  Create CmakeLists.txt file using CGAL-3.8/scripts/cgal_create_cmake_script
cmake .
make simple_example
  ```
Timing your experiments

- Dedicated machine, minimal background processes
- Measure “wallclock” time
  - ctime
  - gettimeofday
  - PAPI counters
  - ...
- Measure within clock resolution
- Average multiple executions
- Choose appropriate input size range
- Use optimization level -O3

https://parasol.tamu.edu/people/amato/Courses/620/cgal/simple-cgal-demo.tgz
Determine number of iterations

Confidence interval

- \( m \): number of samples (iterations)
- \( \bar{x} \): sample mean
- \( s \): sample standard deviation
- \( r \): tolerance/accuracy (e.g., 0.05)
- \( z \): confidence coefficient (95% is 1.96)

- **Conditional** (require compute mean, \( s \) and \( m \) on each iteration):

  While \( \frac{z s}{\sqrt{m}} \geq \bar{x} r \) continue iterating

- **Precomputed** (require compute a set of initial samples):

  \[
  m = \left( \frac{z s}{\bar{x} r} \right)^2
  \]
Timing your experiments

// Computes the difference between two times. Result in seconds
double diff_time(timeval t0, timeval t1) {
    return (t1.tv_sec-t0.tv_sec) + (double(t1.tv_usec-t0.tv_usec)/1000000);
}

int main(int argc, char const **argv) {
    // Initialize set of points
    ....
    timeval t0,t1;
    double elapse = 0;
    for(int i=0; i < num_iter; ++i) {
        gettimeofday(&t0,NULL);
        CGAL::convex_hull_2( points.begin(), points.end(), std::back_inserter(result) );
        gettimeofday(&t1,NULL);
        elapse += diff_time(t0,t1);
    }
    std::cout << result.size() << " points on the convex hull in " << (elapse/num_iter) << " sec" << std::endl;
Finding Big-O constant

A function $T(n)$ is $O(F(n))$ if for some constant $c$ and for all values of $n$ greater than some value $n_0$

$$T(n) \leq c \times F(n), \forall n > n_0$$

Where $T(n)$ is the exact complexity (execution time) for the input size $n$

Experimentally determining $c$ and $n_0$

- Let $F(n)$ be the asymptotic complexity and let $T(n)$ be the measured time of your algorithm
- Run experiments for a wide range of values of $n$ and plot $T(n)/F(n)$ versus $n$. The value of $n$ where $T(n)/F(n)$ levels off is $n_0$ and the $y$ value that it converges to is the constant $c$
Finding Big-O constant

Example: computing ratio of expected time $F(n)$ and measured time $T(n)$

<table>
<thead>
<tr>
<th>n</th>
<th>time (sec)</th>
<th>$T(n)/F(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1.84E-006</td>
<td>1.8435E-008</td>
</tr>
<tr>
<td>1000</td>
<td>8.38E-006</td>
<td>8.3787E-009</td>
</tr>
<tr>
<td>10000</td>
<td>7.51E-005</td>
<td>7.5094E-009</td>
</tr>
<tr>
<td>100000</td>
<td>0.000736748</td>
<td>7.36748E-009</td>
</tr>
<tr>
<td>1000000</td>
<td>0.00757946</td>
<td>7.57946E-009</td>
</tr>
</tbody>
</table>
Finding Big-O constant

Example: graphically computing $c$ and $n_0$
More information

- CSCE-620 Computational Geometry
  http://parasol.tamu.edu/people/amato/Courses/620/

- CGAL
  - Manuals: http://www.cgal.org/Manual/
  - Videos: http://www.cgal.org/videos.html