universität innsbruck

VU C-C++: Libraries

VU Lecture Slides

Johannes Gerstmayer & Markus Walzthöni

uibk.ac.at

Overview

Content:

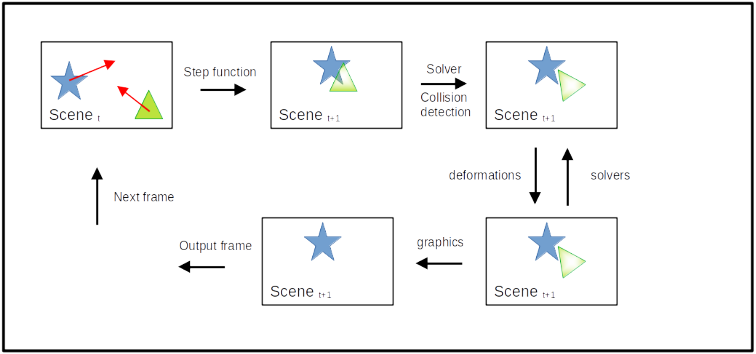
- What are simulations and from components are they made of?
- CGAL
- Eigen
- Boost

Simulation: Basic

Physically based simulations are used to animate behavious and phenomena based on physical properties.
Depending on the issue of simulation different approaches have to be used.

Simulation: Parts

- Scene consisting out of input models
- Input parameters:
  - timestep size
  - step function
  - solver
  - collision detection
  - ...



Simulation:

So we have a lot of the following to do:

- Matrix multiplications
- solving systems of euqations
- recompute shapes and forms
- optionally compute raytracing scene or other high level graphics

What do we need?

- Good container solutions
- Fast math libraries

Boost - What is it?

Boost:

- Collection of algorithms and datatypes
- Header only implementation
- peer-reviewed code based
- protable
- a lot of boost work is used in the C++ STL

Boost - Installation

Linux:

- `sudo apt install libboost-all-dev`
- download tar-ball or zip files from here, extract and install

Windows:

- download zip file from here, extract and install

Project:

- `git submodule add https://github.com/boostorg/boost extern/boost`
- `git submodule update -init -recursive`

Boost - What kind of libraries does it provide?

CMake:

```
project( my-project )
find_package(Boost
  [version] [EXACT]      # Minimum or EXACT version e.g. 1
  [REQUIRED]             # Fail with error if Boost is not
  [COMPONENTS <libs>...] # Boost libraries by their canoni
                         # e.g. "date_time" for "libboost_
  [OPTIONAL_COMPONENTS <libs>...]
                         # Optional Boost libraries by the
  )                      # e.g. "date_time" for "libboost_
```

Boost - What kind of libraries does it provide?

CMake:

```
# result variables

Boost_FOUND #    True if headers and requested libraries w
Boost_INCLUDE_DIRS #    Boost include directories.
Boost_LIBRARY_DIRS #    Link directories for Boost librar
Boost_LIBRARIES #    Boost component libraries to be link
Boost_VERSION # Boost version number in X.Y.Z format.
```

Boost - What kind of libraries does it provide?

Libraries of Boost:

- `Boost::Optional`
- `Boost::Any`
- `Boost::Shared_ptr`
- `Boost::String`
-
    - processing
    - splitting / concatinating
    - regular expressions (!)
- `Boost::Container & Multi-index Container`
- `Boost::Bimap`
- `Boost::Iterator`

Boost - What kind of libraries does it provide?

Libraries of Boost:

- Higher order programming features:
  - Lambdas
  - Delegates
  - Closures
- Compile time programming:
  - Branching
  - Recursion
  - SFINAE
  - Metaprogramming Library & Metafunctions
- lazy evaluations

Boost - What kind of libraries does it provide?

Libraries of Boost:

- Date and time
  - Chrono library
  - Dates, time, time points and durations
  - performance measuring
- Files, Directories:
  - paths incl. manipulation
  - traversing paths
  - querzing file system entries
  - performing operations on files
- IOStreams

Boost - What kind of libraries does it provide?

Libraries of Boost:

- Concurrent tasks and parallel execution
- Networkprogramming (Network Socket IO)

Eigen - What is it?

What is Eigen3?

- C++ library with matrix operations implemented
- Easy to install
- Easy to use interface
- Already delivered with architecture optimized code
- Helper functions for dense and spare matrices, arrays and vectors
- "Header only" implementation

Eigen - Basics?

```cpp
#include <Eigen/Dense>

int main()
{
  Eigen::Matrix<double, 10, 10> A;
  A.setZero();
  A(9, 0) = 1.234;
  std::cout << A << std::endl;
  return 0;
}
```

This is similar to `double A[10][10]`

Eigen - Dynamic size

```
int n = 64;
int m = 65;
Eigen::Matrix<
    double,
    Eigen::Dynamic,
    Eigen::Dynamic > A(n, m);
A.resize(20, 20);
std::cout << "Size␣is";
std::cout << A.rows() << "x"
  << A.cols() << std::endl;
```

This is similar to a 2D version of `std::vector<double>`

Eigen - Convenience Typedefs

```
Eigen::Matrix3d = Eigen::Matrix<double, 3, 3>
Eigen::Matrix3i = Eigen::Matrix<int, 3, 3>
Eigen::MatrixXd = Eigen::Matrix<double, Eigen::Dynam
Eigen::VectorXd = Eigen::Matrix<double, Eigen::Dynam
Eigen::RowVectorXd = Eigen::Matrix<double, 1, Eigen::
etc.
```

Eigen - Matrix arithmetics

```
Eigen::MatrixXd A(5, 10);
Eigen::MatrixXd B(10, 2);
Eigen::VectorXd vec(10);
Eigen::MatrixXd C = A * B;
Eigen::VectorXd w = A * vec;
```

Also dot and cross products for vectors, transpose, and usual scalar arithmetic +-*/

Eigen - Coefficientwise matrix arithmetics

```
Eigen::Matrix3d A, B;
A.array() = 2.0; // set all values to 2.0
A.array() = B.array().sin(); // set each element of A
the same element in B
Eigen::Array3d W;
W = W * A; // Error - cannot mix Array with Matrix
```

Eigen - Interface for `std::vector<>` or similar

It is easy to "overlay" existing memory with an Eigen Array or Matrix:

```
std::vector<double> a(1000);
Eigen::Map<Eigen::Matrix<double, 100, 10>> a_eigen(a.
a_eigen(10, 0) = 1.0;
Eigen::Map<Eigen::MatrixXd> a2_eigen(a.data(), 10, 10
```

Eigen - Important remarks

- True optimization only with `-O2` upwards
- eigen.tuxfamily.org DOC
- Boost DOCS
- Linear Algebra Tutorial

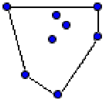
CGAL

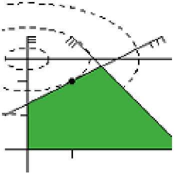
THX to IGS for the next slides!

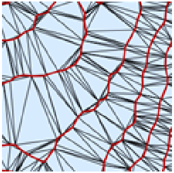
What is CGAL?

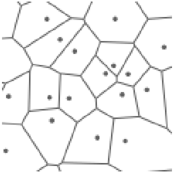
- A reliable geometric algorithms in the form of an open source C++ library
- The project started in October 1996, collaboration of 7 European Institutions
- Robust and generic
- Computational Geometry

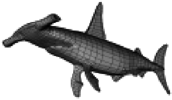
Packages

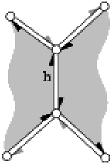


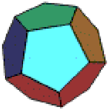




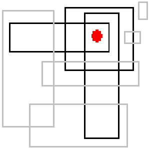


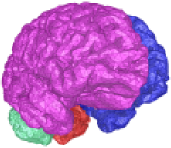


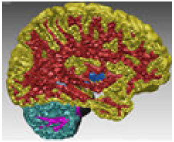


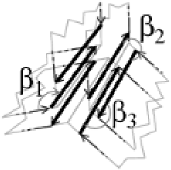


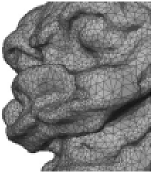
Packages

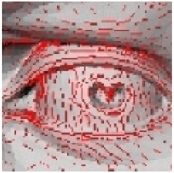


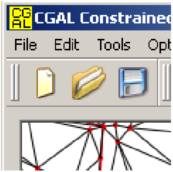


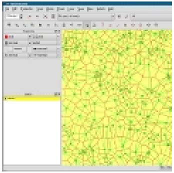




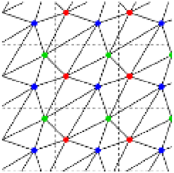
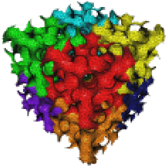
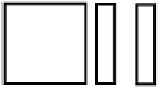
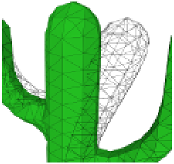
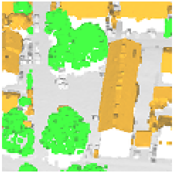
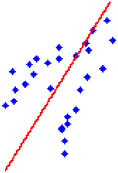
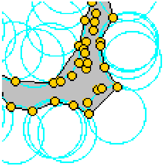
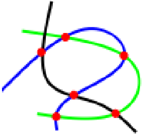




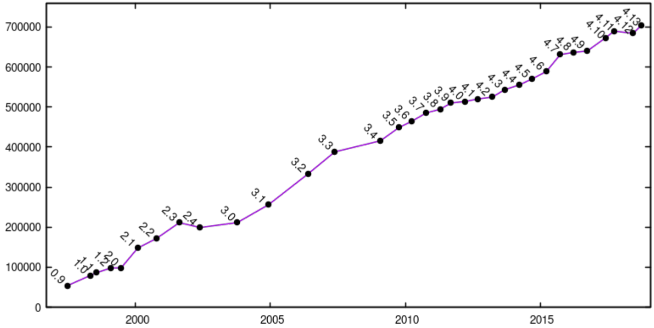




Packages

CGAL Release timeline



Getting Started

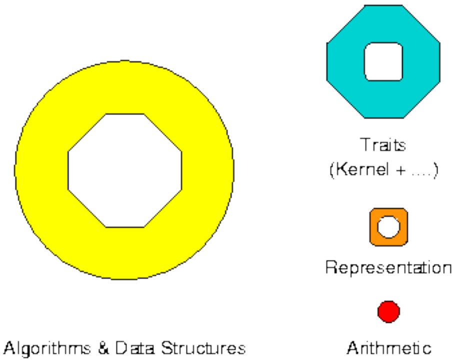
- https://www.cgal.org/
- https://github.com/CGAL/cgal
- Cross-platform: Window, Linux, Mac OS
- Binaries or source

```
# Debian or Linux Mint
sudo apt-get install libcgal-dev # install the CGAL library
sudo apt-get install libcgal-demo # install the CGAL demos
```

Third Party Libraries

- Standard Template Library (STL)
- Boost
- GMP and MPFR
- zlib
- OpenGL
- Qt5

Kernel, Traits and Concepts



Kernels

```
//#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
//#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>

#include <CGAL/Simple_cartesian.h>

typedef CGAL::Simple_cartesian<double>                  Kernel;
typedef Kernel::Point_3                                 Point_3;
typedef Kernel::Vector_3                                Vector_3;
typedef Kernel::Plane_3                                 Plane_3;
typedef Kernel::Triangle_3                              Triangle_3;
```

CGAL Hello World!

```
#include <iostream>
#include <CGAL/Simple_cartesian.h>

typedef CGAL::Simple_cartesian<double> Kernel;
typedef Kernel::Point_2 Point_2;
typedef Kernel::Segment_2 Segment_2;

int main()
{
  Point_2 p(1,1), q(10,10);

  std::cout << "p = " << p << std::endl;
  std::cout << "q = " << q.x() << " " << q.y() << std::endl;

  std::cout << "sqdist(p,q) = " << CGAL::squared_distance(p,q) << std::endl;

  std::cout << "midpoint(p,q) = " << CGAL::midpoint(p,q) << std::endl;
  return 0;
}
```

Single file CMakelists

```
# Created by the script cgal_create_cmake_script
# This is the CMake script for compiling a CGAL application.

project( Hello_World_Examples )
cmake_minimum_required(VERSION 2.8.10)

find_package(CGAL QUIET)

if ( CGAL_FOUND )
  include( ${CGAL_USE_FILE} )
  include( CGAL_CreateSingleSourceCGALProgram )
  include_directories (BEFORE "../../include")

  create_single_source_cgal_program( "points_and_segment.cpp" )
else()
    message(STATUS "This program requires the CGAL library, and will not be compiled.")
endif()
```

Multiple files CMakelists

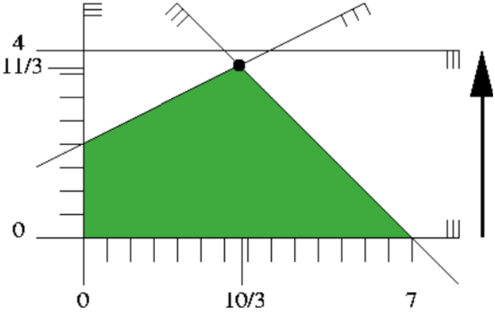
```
project( Kernel_23_Examples )

cmake_minimum_required(VERSION 2.8.10)

find_package(CGAL QUIET)

if ( CGAL_FOUND )

  include( ${CGAL_USE_FILE} )

  include( CGAL_CreateSingleSourceCGALProgram )

  include_directories (BEFORE "../../include")

  # create a target per cppfile
  file(GLOB cppfiles RELATIVE ${CMAKE_CURRENT_SOURCE_DIR} ${CMAKE_CURRENT_SOURCE_DIR}/*.cpp)
  foreach(cppfile ${cppfiles})
    create_single_source_cgal_program( "${cppfile}" )
  endforeach()

else()

    message(STATUS "This program requires the CGAL library, and will not be compiled.")

endif()
```

Convex Hull

```
#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;
typedef std::vector<Point_2> Points;
int main()
{
  Points points, result;
  points.push_back(Point_2(0,0));
  points.push_back(Point_2(10,0));
  points.push_back(Point_2(10,10));
  points.push_back(Point_2(6,5));
  points.push_back(Point_2(4,1));
  CGAL::convex_hull_2( points.begin(), points.end(), std::back_inserter(result) );
  std::cout << result.size() << " points on the convex hull" << std::endl;
    for(int i = 0; i < result.size(); i++){
    std::cout << result[i] << std::endl;}
  return 0;
}
```

Linear Programming

$$
\begin{array}{llcl}
\text{minimize} & -32y + 64 & & \\
\text{subject to} & x + y & \leq & 7 \\
 & -x + 2y & \leq & 4 \\
 & x & \geq & 0 \\
 & y & \geq & 0 \\
 & y & \leq & 4
\end{array}
$$



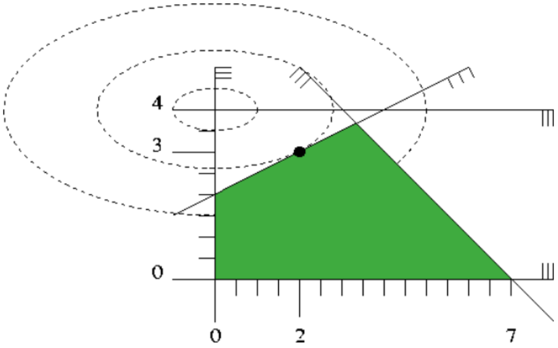
Linear Programming solver

```
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
// choose exact integral type
#ifdef CGAL_USE_GMP
#include <CGAL/Gmpz.h>
typedef CGAL::Gmpz ET;
#else
#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;
#endif
// program and solution types
typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;
```

Linear Programming solver

```
int main() {
  // by default, we have a nonnegative LP with Ax <= b
  Program lp (CGAL::SMALLER, true, 0, false, 0);
  // now set the non-default entries
  const int X = 0;
  const int Y = 1;
  lp.set_a(X, 0,  1); lp.set_a(Y, 0, 1); lp.set_b(0, 7);  //  x + y  <= 7
  lp.set_a(X, 1, -1); lp.set_a(Y, 1, 2); lp.set_b(1, 4);  // -x + 2y <= 4
  lp.set_u(Y, true, 4);                                   //       y <= 4
  lp.set_c(Y, -32);                                       // -32y
  lp.set_c0(64);                                          // +64
  // solve the program, using ET as the exact type
  Solution s = CGAL::solve_linear_program(lp, ET());
  assert (s.solves_linear_program(lp));

  // output solution
  std::cout << s;
  return 0;
}
```

Quadratic Programming Problem



$$\begin{array}{llcl}
\text{minimize} & x^2 + 4(y-4)^2 & & \\
\text{subject to} & x + y & \leq & 7 \\
& -x + 2y & \leq & 4 \\
& x & \geq & 0 \\
& y & \geq & 0 \\
& y & \leq & 4
\end{array}$$

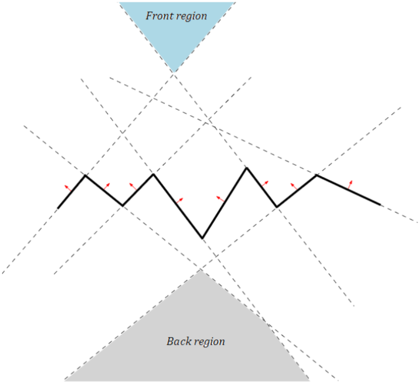
Quadratic Programming solver

```
#include <iostream>
#include <cassert>
#include <CGAL/basic.h>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
// choose exact integral type
#ifdef CGAL_USE_GMP
#include <CGAL/Gmpz.h>
typedef CGAL::Gmpz ET;
#else
#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;
#endif
// program and solution types
typedef CGAL::Quadratic_program<int> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;
```

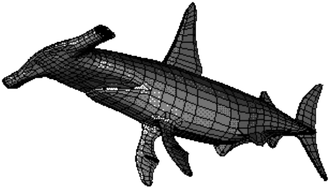
Quadratic Programming solver

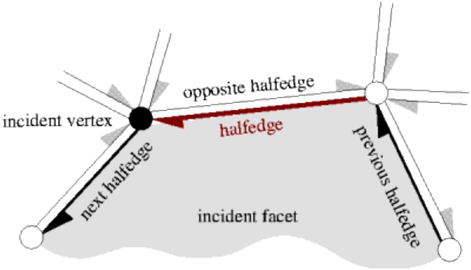
```
int main() {
  // by default, we have a nonnegative QP with Ax <= b
  Program qp (CGAL::SMALLER, true, 0, false, 0);
  // now set the non-default entries:
  const int X = 0;
  const int Y = 1;
  qp.set_a(X, 0,  1); qp.set_a(Y, 0, 1); qp.set_b(0, 7);  //  x + y  <= 7
  qp.set_a(X, 1, -1); qp.set_a(Y, 1, 2); qp.set_b(1, 4);  // -x + 2y <= 4
  qp.set_u(Y, true, 4);                                   //       y <= 4
  qp.set_d(X, X, 2); qp.set_d (Y, Y, 8); // !!specify 2D!!    x^2 + 4 y^2
  qp.set_c(Y, -32);                                       // -32y
  qp.set_c0(64);                                          // +64
  // solve the program, using ET as the exact type
  Solution s = CGAL::solve_quadratic_program(qp, ET());
  assert (s.solves_quadratic_program(qp));
  // output solution
  std::cout << s;
  return 0;
}
```

Example of usage



3D Polyhedral Surface





Polyhedron

```
#include <CGAL/Simple_cartesian.h>
#include <CGAL/HalfedgeDS_vector.h>
#include <CGAL/Polyhedron_3.h>
#include <iostream>
typedef CGAL::Simple_cartesian<double>                  Kernel;
typedef Kernel::Point_3                                 Point_3;
typedef CGAL::Polyhedron_3< Kernel,
                           CGAL::Polyhedron_items_3,
                           CGAL::HalfedgeDS_vector>    Polyhedron;
int main() {
    Point_3 p( 1.0, 0.0, 0.0); Point_3 q( 0.0, 1.0, 0.0);
    Point_3 r( 0.0, 0.0, 1.0); Point_3 s( 0.0, 0.0, 0.0);
    Polyhedron P;    // alternative constructor: Polyhedron P(4,12,4);
    P.make_tetrahedron( p, q, r, s);
    CGAL::set_ascii_mode( std::cout);
    std::copy( P.points_begin(), P.points_end(),
           std::ostream_iterator<Point_3>( std::cout, "\n"));
    return 0;
}
```

Normal Vectors

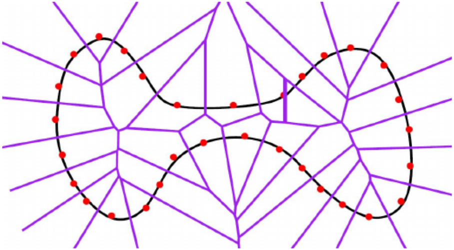
```
struct Normal_vector {
    template <class Facet>
    typename Facet::Plane_3 operator()( Facet& f) {
        typename Facet::Halfedge_handle h = f.halfedge();
        // Facet::Plane_3 is the normal vector type. We assume the
        // CGAL Kernel here and use its global functions.
        return CGAL::cross_product(
          h->next()->vertex()->point() - h->vertex()->point(),
          h->next()->next()->vertex()->point() - h->next()->vertex()->point());
    }
};
```

Plane Equation - Camel Demo

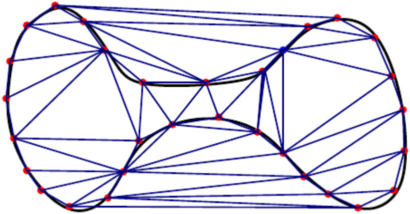
```
struct Plane_equation {
    template <class Facet>
    typename Facet::Plane_3 operator()( Facet& f) {
        typename Facet::Halfedge_handle h = f.halfedge();
        typedef typename Facet::Plane_3  Plane;
        return Plane( h->vertex()->point(),
                      h->next()->vertex()->point(),
                      h->next()->next()->vertex()->point());
    }
};
Point_3 p( 1, 0, 0);
Point_3 q( 0, 1, 0);
Point_3 r( 0, 0, 1);
Point_3 s( 0, 0, 0);
Polyhedron P;
P.make_tetrahedron( p, q, r, s);
std::transform( P.facets_begin(), P.facets_end(), P.planes_begin(),
                Plane_equation());
```

Voronoi Diagram

$$Vor(\mathbf{p}) = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \forall \mathbf{q} \in P,\ (\|\mathbf{x} - \mathbf{p}\| \leq \|\mathbf{x} - \mathbf{q}\|) \right\}$$



Delaunay Triangulation

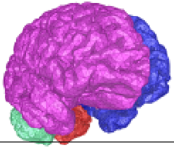


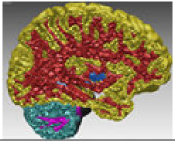
Delaunay and points generator

```
#include <CGAL/Delaunay_triangulation_3.h>
typedef CGAL::Exact_predicates_inexact_constructions_kernel     K;
typedef K::Point_3                                              Point_3;
typedef CGAL::Delaunay_triangulation_3<K>                       Delaunay;
typedef Delaunay::Vertex_handle                                 Vertex_handle;
typedef CGAL::Surface_mesh<Point_3>                             Surface_mesh;

int main()
{
  CGAL::Random_points_in_sphere_3<Point_3> gen(100.0);
  std::list<Point_3>   points;

  // generate 250 points randomly in a sphere of radius 100.0
  // and insert them into the triangulation
  CGAL::cpp11::copy_n(gen, 250, std::back_inserter(points) );
  Delaunay T;
  T.insert(points.begin(), points.end());
```

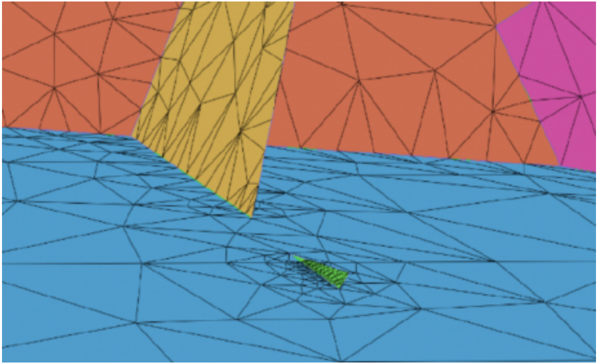
CGAL with SOFA

```xml
<?xml version="1.0"?>
<Node name="root" gravity="0 0 0" dt="1"  >
    <RequiredPlugin pluginName="CGALPlugin"/>
    <RequiredPlugin pluginName="image"/>

    <ImageContainer name="image" template="ImageUC" filename="data/image/image-cube.inr"/>

    <MeshGenerationFromImage template="Vec3d" name="generator" printLog="true" drawTetras="true"
    image="@image.image" transform="@image.transform"
    cellSize="0.5" facetAngle="30" facetSize="1" cellRatio="3" facetApproximation="1" ordering="0"
    label="1 2 3" labelCellSize="0.2 0.5 0.1" labelCellData="100 200 300"/>

    <Mesh name="volume" points="@generator.outputPoints"    tetras="@generator.outputTetras"/>
    <VTKExporter name="exporter" filename="data/output.vtu"  XMLformat="1" edges="0" tetras="1"
    listening="true" exportAtBegin="true" cellsDataFields="generator.outputCellData" overwrite="true"/>
</Node>
```



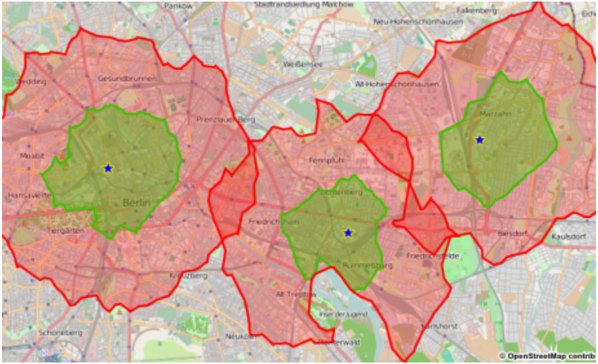


Geometry Factor: Fracture Mesh Generation

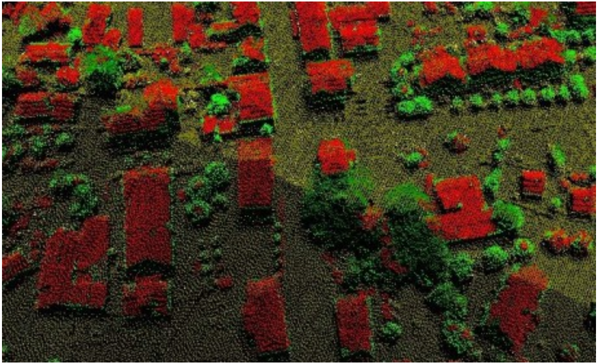


Geometry Factory: Surface Meshes from Aerial Images

Geometry Factory: Alpha Shapes for Computing Catchment Areas



Geometry Factory: Constrained Delaunay Triangulations for Photogrammetry



References

- IGS - Obergurgl
- Eigen, Boost and CGAL dev teams + content
- Chris Richardson (Camebridge)
- Boost DOC
- 500 page Boost book

Thank you for your attention!

Johannes Gerstmayer & Markus Walzthöni
uibk.ac.at