ViennaMesh

Quick User Guide
v0.2.0

Institute for Microelectronics
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria

Developers:

Josef Weinbub
Johann Cervenka
Karl Rupp

Institute for Microelectronics
Technische Universität Wien
Gußhausstraße 27-29 / E360
A-1040 Vienna, Austria,Europe

Phone	+43-1-58801-36001
Fax	+43-1-58801-36099
Web	http://www.iue.tuwien.ac.at

Contents

1 Introduction

To provide applications with the utmost flexibility in the generation and adaptation of meshes, the generic and high-quality meshing library, ViennaMesh, has been developed. ViennaMesh provides a unified interface for various mesh related tools. These tools cover mesh generation, adaptation, and classification of multi-segmented (aka. multi-material) meshes and geometries for unstructured two- and three-dimensional meshes. The goal is to provide applications with an additional back-end layer for mesh generation, allowing to seamlessly exchange mesh tools, for example, mesh generation kernels.

2 Building

In general, ViennaMesh only supports Unix based systems and requires several libraries to be available:

- CMake [1]
- Boost [2]
- GMP [3]
- CGAL [4]

Typically `CMake`, `CGAL`, and `GMP` are provided by the distributions, for example, Ubuntu. However, CGAL maybe has to be installed manually.

An exemplary build script is provided within the root folder of ViennaMesh.

```
build.sh
```

Note that `Debug` and `Release` builds are available, by utilizing the appropriate `CMake` configuration parameter.

```
-D CMAKE_BUILD_TYPE=Release
-D CMAKE_BUILD_TYPE=Debug
```

The `CGAL` include and library paths have to be provided via the respective `CMake` configuration parameters.

```
-D CGAL_INC_DIR=$CGALINC
-D CGAL_LIB_DIR=$CGALLIB
```

Note that the build script expects the parameters `CGALINC` and `CGALLIB` to be provided by the environment. Certainly these paths can be entered statically within the build script.

After the build process is finished, an application executable is available within the build folder.

```
build/vmesh
```

This executable is capable of producing volume meshes of high quality of `bnd` and `hin` input files.

Example input files can be found in the input/ folder.

3 Usage

For examples how to utilize ViennaMesh, have a look at the source files of either the main application

```
src/vmesh.cpp
```

or the test applications.

```
tests/src/
```

However, in the following a short glimpse is provided on the capabilities of ViennaMesh.

3.1 Wrappers

A input datastructure wrapper approach is used to decouple the ViennaMesh internal algorithms from a specific input datastructure. For example, a wrapper for ViennaGrid datastructures is available.

```
typedef viennagrid::domain<viennagrid::config::line_2d>     domain_type;
domain_type domain;

typedef wrapper<viennagrid, domain_type>                    wrapper_type;
wrapper_type    wrapped_input_data(my_bnd_reader);
```

To support a different datastructure a corresponding wrapper has to be provided. Available wrappers can be found here:

```
viennamesh/wrapper
```

This wrapped input data can now be used as input for a functional meshing chain.

3.2 Functional Meshing Chain

ViennaMesh provides specific mesh related functionalities via C++-functors. These functors can be chained, meaning that the result of one functor can be the input of the next functor and so on. This is commonly known as a `fold` operation. This, naturally, enables the mapping of an arbitrary meshing flow into a C++ expression. Each functor alters the input and passes it on to its successor.

The functor types are generated by a tag-dispatched mechanism, as depicted in the following.

```
typedef mesh_generator<cervpt>::type            hull_generator_type;
typedef mesh_adaptor<orienter>::type            orient_adaptor_type;

hull_generator_type hull_generator;
orient_adaptor_type orient_adaptor;
```

These instantiated objects can now be utilized in the presented fold manner.

```
hull_generator(orient_adaptor(wrapped_input_data));
```

The result type of such a chain can be derived from the last applied functor type.

```
typedef hull_generator_type::result_type        result_type;
```

Hence, the result can be retrieved by

```
result_type result = hull_generator(orient_adaptor(wrapped_input_data));
```

3.3 Meta-Selection

ViennaMesh provides a meta-selection environment which determines the best suited mesh generator tool based on specific properties.

The required properties can be specified by a associative, compile-time container.

```
typedef make_map<
   cell_type, algorithm,
   simplex,   advancing_front
>::type               required_properties;
```

Note the associative relation between Line 2 and 3. The key is located above the related value entry. Hence, the required properties are that the mesh elements are of simplex type, for example, triangles or tetrahedrons. Additionally, an Advancing Front mesh generation algorithm should be used.

This property type can now be forwarded to the meta-selection utility which computes a corresponding mesh generator type.

```
typedef compute_mesh_generator<required_properties>::type
   computed_mesh_generator_type;
```

This type can immediately be used to instantiate the mesh generator functor, which again can be used in the already introduced functional meshing chain.

```
computed_mesh_generator_type  mesh_generator;
mesh_generator(wrapped_input_data);
```

4 Available Tools

As already mentioned, ViennaMesh meshing tools are provided via C++-functors. At the moment, we focus on unstructured mesh generation, adaptation, and classification methods based on simplicial mesh elements.

The following table provides an overview of the currently available tools.

Generation	INCREMENTAL DELAUNAY	ADVANCING FRONT	RAW
2D	1	0	0
32D	0	0	1
3D	1	1	0

Table 1: There is one two-dimensional mesh generator available (Triangle) and one hull mesh generator (CervPT). Note that CervPT is not based on a specific meshing algorithm, as it simply triangulates a given Polygon without any quality constraints. For three-dimensional volume meshing two meshing kernels are available, based on different meshing libraries (TetGen, Netgen).

Adaptation	ORIENTATION	QUALITY IMPROVEMENT	CELL NORMALS
2D	0	0	0
32D	1	1	1
3D	0	0	0

Table 2: At this stage of development the primary focus is on hull meshes. Therefore, the mesh adaptation tools are only available for these types of meshes, ultimately providing a statistical overview of the quality of the whole mesh.

Classification	CELL ASPECT RATIO
2D	0
32D	0
3D	1

Table 3: A mesh classification tool is available which analyzes the quality of each mesh element of a simplicial volume mesh.

5 License

ViennaMesh is published under the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) [5]. The complete license text can be found in the file

```
LICENSE
```

located in the root folder of the ViennaMesh package.

ViennaMesh ships several external libraries located in the external/ folder. These libraries are equipped with their own licenses, which are explicitly available within the respective root folders as a LICENSE file.

References

[1] “CMake.” [Online]. Available: http://www.cmake.org/

[2] “Boost C++ Libraries.” [Online]. Available: http://www.boost.org/

[3] “GMP - The GNU Multiple Precision Arithmetic Library.” [Online]. Available: http://gmplib.org/

[4] “CGAL - The Computational Geometry Algorithms Library.” [Online]. Available: http://www.cgal.org/

[5] “GNU Lesser General Public License.” [Online]. Available: http://www.gnu.org/licenses/lgpl.html