OpenMesh Toolkit for mesh analysis

Documentation

Author: Rostislav Hulik ([rosta.hulik@gmail.com](mailto:rosta.hulik@gmail.com))

Contents

[Description of the OMToolkit 3](#_Toc354993296)

[Installation 4](#_Toc354993297)

[Prerequisities 4](#_Toc354993298)

[Compilation 4](#_Toc354993299)

[OMToolkit library api 5](#_Toc354993300)

[OpenSceneGraph Geometries 5](#_Toc354993301)

[Writer and REader 5](#_Toc354993302)

[Property Visualiser 6](#_Toc354993303)

[Local projections method 6](#_Toc354993304)

[Curvature analyser 6](#_Toc354993305)

[Local Projections smoothing 7](#_Toc354993306)

[Local Projections SIFT descriptors 7](#_Toc354993307)

[Types and logs 7](#_Toc354993308)

[Modules 9](#_Toc354993309)

[OMLoadFile 9](#_Toc354993310)

[OMSaveFile 9](#_Toc354993311)

[OMViewer 10](#_Toc354993312)

[OMDecimateMesh 10](#_Toc354993313)

[OMComputeMatrices 11](#_Toc354993314)

[OMFilterMesh 12](#_Toc354993315)

[OMSmoothMesh 13](#_Toc354993316)

[OMSubdivideMesh 14](#_Toc354993317)

[OMCurvature 14](#_Toc354993318)

[OMDetectEdges 15](#_Toc354993319)

[OMComputeDescriptors 15](#_Toc354993320)

[Examples of use 17](#_Toc354993321)

[References 18](#_Toc354993322)

# Description of the OMToolkit

This framework is based on the OpenMesh [1] library, a well-known mesh processing framework. OpenMesh provides great low level support and several processing algorithms, such as decimation, subdivision etc., but it doesn’t consider any analysis tools (curvature computation etc).

Main goal of this project is to extend OpenMesh to provide these functions. Another purpose is to implement the Local Projections analysis method, which was published in several papers. [7, 8]. Moreover, we added a strong module support for sequential batch processing possibilities. Each module uses MDSTk [2] library interface to provide multiplatform input and output channel (pipe, file, stdin etc) and processes only elementary operations (see Implemented Modules) specified by command line arguments.

In this document, the installation and overview of the OMToolkit is provided. For more precise API documentation, please use the Doxygen html files.

This project was created to support my dissertation thesis and is meant to be used for academic purposes only. Although it has open BUT license, the large optimizations must be done for industrial use.

# Installation

## Prerequisities

OMToolkit uses CMake for creating a VS project file / makefile. There is however some libraries OMToolkit is dependent on.

* OpenMesh version 2.2
  + Since OMToolkit is extension of OpenMesh, you must have OpenMesh compiled and installed on your system.
  + CMake has preconfigured names of libraries with “d” as suffix in case of debug version (ATTENTION, older versions of OpenMesh compiler names debug and release versions without any difference, so it has to be renamed manually)
* **MDSTk version 1.0.2**
  + Medical data segmentation toolkit has to be installed also. This is because of need some base classes for deriving modules, logging classes, mathematical background etc.
  + It is only necessary basic compilation of MDSTk without any extensions, naming conventions of compiled files are default.
* **Open CV version 2.2.0**
  + The Open CV library capabilities are used for some Local projections matrix computations.
  + Naming conventions for debug/release are default.
* **Open CL version 1.0**
  + The Open CL library is used for experimental HW computation of Local projections.
  + The headers used are the Khronos Open CL headers version 1.0
* **OpenSceneGraph version 3.0**
  + OSG library is necessary only if user wants to compile viewer module and osg::geometry derived classes in OMLibrary. This can be set up in CMake by choosing BUILD\_MODULES\_OSG.
  + Naming conventions for debug/release are default.

## Compilation

Compilation itself is simple. It is necessary to configure CMake as usual, there is not much options. The most important setting is OMToolkit\_XXX paths to libraries. User must set them to path with installed libraries. If BUILD\_MODULES\_OSG is not checked, OSG path is not necessary to be valid.

However, CMake global settings are under construction for now, so please be patient, if there are some problems.

# OMToolkit library api

The library provides additional functions to OpenMesh. In fact, almost all functions provided in this library are used in modules described in section **Modules**. In this section, there is only a brief introduction on library functions, for detailed usage, please refer to Doxygen documentation.

## OpenSceneGraph Geometries

This section consists three main classes which are used for conversion into open scene graph. All classes are inherited from OSG, so they can be manipulated like standard geometries.

* OMGeometry
  + Class is used for converting loaded module into OSG geometry, so it can be easily attached as OSG drawable.
  + Specified in constructor parameters, vertices, faces, colors and normals can be added for viewing. Also, their binding is possible to change.
* OMDirectionsGeometry
  + This class visualizes vector informations attached as vector property of vertex or face.
  + A constructor searches given model for property handle and visualizes a vector from at face/vertex.
  + It is also possible to specify length of a vector as percent his original length.
* OMLegendGeometry
  + Geometry visualizes a legend for color information.
  + It shows a color bar computed using OMVisualiser::getColor function.
  + It also shows boundary values specified in constructor.
  + It’s necessary to add it into OSG group using its addToGroup function due to specific settings of HUD.

For specific usage of these three classes, please refer to OMViewer module.

## Writer and REader

These two classes are heavily rewrited OpenMesh::Reader and OpenMesh::Writer. They are rewrited due to necessity of MDSTk channel serialization. In fact, MDSTk channel is a stream, but it is not compatible with std::stream, so several methods and structures must be modified. For normal user, there is not necessary to look under the hood of these classes.

To serialize your own property (specific stricture), it is necessary to write your own methods for (de)serialization – see [www.openmesh.org](http://www.openmesh.org) documentation. There is no change in this routine.

* OMWriterExt
  + Class derived from OM writer. It uses a binary OM format to serialize a mesh with all properties into a specific stream/file/channel.
  + User defined properties must be set as persistent to be copied into stream.
* OMReaderExt
  + Class derived from OM reader. It uses a binary OM format to deserialize a mesh with all properties from a specific stream/file/channel.
  + User defined properties are deserialized only if they are contained in destination mesh, standard properties also.
* OMFormatExt
  + Class containing extended OM format methods. There is no need to change this class.
* OMStoreRestore
  + Extension of binary serializer from OpenMesh. Like OMFormatExt, there is no need to do any changes in this file.
* OMIO
  + A couple of methods for writing/reading from files/streams/channels.
  + There are simple methods for accessing all writer/reader modules and they should be used.

For specific usage of these three classes, see OMLoadFile or OMSaveFile modules or Doxygen documentation.

## Property Visualiser

This chapter consists of only one class – **OMVisualiser**. This class is written for understandably visualizing scalar information of vectors / colors. By calling ComputeColors method, you can easily convert a value given by property handle into color scale.

There is possibility to omit some values – this can be specified as percentual value. This value specifies number of per cent of highest/lowest values, which will be omitted from visualizing. This prevents this scale from degrading, when there are several too high values.

Computed information is saved as vertex/face color (unsigned char 4D vector). In this instance, there is also saved minimum and maximum (without omitted values)

## Local projections method

The library also consists of the tool for local projecting of the mesh. This is the implementation of algorithm presented in [7, 8].

The method is implemented by the class **OMProjector**. It projects the neighborhood of each vertex and saves it into the mesh structure. The matrices can be sent by the pipes to other modules.

## Curvature analyser

Class **OMCurvature** approximates several curvatures on a triangular mesh. Each call of calcXCurvature() computes specified curvature at each vertex of triangular mesh. If there is a direction defined, it also adds a vector of this direction (unitary). Algorithms are described in [4] and [5].

For anticipating any misunderstandings, a brief explication is provided:

* calcGaussCurvature
  + Function calculates and saves Gauss curvature to a mesh.
  + Curvature value is saved in curvature magnitude property in ModuleMesh (see OMTypes).
  + There is no Gauss curvature direction defined – curvature vector is set to (0,0,0)
  + Gauss curvature is defined as GC = minC \* maxC
* calcMeanCurvature
  + Function calculates and saves mean curvature.
  + As vector information, a normal to mean curvature direction is saved – this vector should be perpendicular to a vertex normal and mean curvature direction.
  + Mean curvature is defined as MC = (minC + maxC)/2
* calcMinCurvature
  + Calculates minimal curvature and its direction.
* calcMaxCurvature
  + Calculates maximal curvature and its direction.

For the Loacal projections computed matrices, the OMMatrixCurvature class is available. The method **Compute** provides the computation of minimum, maximum, mean and Gauss curvature using the LPM.

## Local Projections smoothing

The algorithm for smoothing using the local projections method is also implemented in OMToolkit. The class **OMSmoother** implements this algorithm, however this method is in the experimental state only.

## Local Projections SIFT descriptors

The toolkit provides the mesh matching capabilities by the implementation of the SIFT algorithm computed from projected matrices. This is implemented in the class **OMMatrixDescriptors**.

The class computes the feature points, for now based on the pre-computed curvature. Then, it has the functions for the computations of the SIFT features from these feature points, using the projected matrices and the Open CV SIFT algorithm. The output is saved in the output file.

## Types and logs

Classes discussed in this chapter are auxiliary classes used in OMToolkit. They are primarily targeted to help creating additional modules.

* ErrorLog
  + This class provides deprecated logging functions. In fact, this class was used for debugging, now, it is replaced by MDSTk log functions.
* OMTypes
  + This namespace is necessary for communication between modules, since reader reads only properties included in destination mesh. So it has to exist a mutual class ModuleMesh, which contains all used properties (curvature etc…).
  + If user wants to add a new property, which is considered to persist in communication between modules, it must be added in ModuleMesh and set as “persistent” (see curvature property for details)
  + In namespace OMTypes, types used in all OMToolkit library are defined. So if user wants to use one of the specific data types, it is enough to include this header file.

# Modules

## OMLoadFile

### Overview

This is a basic module for loading a mesh from file. It uses standard OpenMesh functions for loading a file, so it’s dependent on implemented read methods for each file. For now, following formats are supported:

* OFF – ASCII and binary
* OBJ – ASCII only
* STL – ASCII and binary
* OM – binary only
* PLY – ASCII and binary

File format is determined from file extension, so the only parameter of this module is file specification (module expects format and extension match of course). After file reading, a mesh is sent into output channel.

### Parameters

Command line parameters for this module are following:

* -file filename
  + This parameter specifies an input file
  + It is obligatory to specify a file

Example of startup: **omLoadFile –file intput.stl**

## OMSaveFile

### Overview

This module is very similar to OMLoadFile, except it executes the contrary – it saves a mesh into a file. For format support, please see OMLoadFile. File format is also determined from given file name and this module saves the files with default OpenMesh parameters (formats which support binary mode are saved as binary, the other as ASCII).

Module waits for connecting input channel (pipe, stdin) and reads a mesh from it. Then, the mesh is written into output file.

### Parameters

Command line parameters for this module are following:

* -file filename
  + This parameter specifies an output file
  + Obligatory parameter

Example of startup: **omSaveFile – file output.stl**

## OMViewer

### Overview

For use of this module, you must configure CMake for compile modules with OpenSceneGraph support. Module expects a file from input channel, then loads this mesh into OSG geometry and fires up an OpenSceneGraph viewer. There is no output (it should be used as the end of executing pipeline). This module can visualize specific data from given mesh as color information (with a legend shown at the top of the screen), also it can visualize specific vectors.

### Parameters

All parameters are voluntary:

* -visualise what
  + Specifies, what scalar information should be visualized.
  + For now, there is only one possibility – curvature
  + Default value is none
* -directions ofWhat
  + Specifies, what vector information should be visualized.
  + Default value is also none.
  + There is three possibilities (face normals, vertex normals, vertex curvature directions)
* -omit
  + Number of highest/lowest values per cent, which will be omitted from visualize (see chapter Property visualiser)

ATTENTION: If there is need to visualize any information, it must be computed in advance (specific module must be started before viewer). Only normals are computed in viewer.

Example of startup: **omViewer –visualise curvature –directions fnormals < input.om**

Default setting of this module is visualize model without any information omitting 2% of extrema values.

## OMDecimateMesh

### Overview

This module provides support for decimation of meshes. It reads an input channel, processes specified decimation and writes a result into output channel. For now, the decimation methods are these implemented in OpenMesh. Module can be executed without any command line parameter (see default setting).

Each decimation method can be set up as binary or not. Binary means, that only stopping criterion is desired number of vertices/collapses. Non binary method adds a criterion, which can be for example minimal error etc. If this algorithm did not reach desired vertex/collapse count but there is no edge which can be collapsed according to this criterion, it stops.

### Parameters

All parameters are voluntary, but following settings can be specified:

* -method decimationMethod
  + This argument specifies used decimation method
  + It is based on implemented decimation methods in OpenMesh
    - Quadric error metric (value: quadric)
    - Maximum roundness of triangle metric (value roundness)
  + Each method has specific arguments
* -minR num
  + This argument is Maximum roundness metric specific
  + Specifies minimal roundness of triangle (sets non binary mode on)
  + Values [0..1]
* -maxE num
  + This argument is Quadric error metric specific
  + Specifies maximal error for collapse (set non binary mode on)
  + Values [0..inf]
* -maxND num
  + This is non method specific argument
  + Specifies maximum normal deviation before/after a collapse
* -indS
  + This switch-like flag specifies using of independent sets for decimation
  + After each collapse, a neighborhood of result is locked and it won’t be decimated no more
* -vertices no
  + Final number of vertices.
  + This is a main stopping criterion for mesh smoothing
* -collapses no
  + Final number of collapses
  + This is also stopping criterion for mesh smoothing, but is should not be used in combination with vertices

Default settings of this module are Quadric error decimation to a 50% of vertices.

## OMComputeMatrices

### Overview

The module computes matrices on the mesh (executes the local projections algorithm [7, 8]). The matrices are computed for each vertex and its neighborhood.

### Parameters

All parameters are voluntary, but following settings can be specified:

* -size size
  + Specifies matrix size in mesh space. This means the double precision number of the length of matrix.
* -resolution res
  + Specifies square matrix dimension in one direction.
  + Must be odd number (for filtration purposes)
* Relative true/false
  + Use this argument if you want to specify matrix size relatively to the median of edge lengths.
* -xdir dir
  + Specifies vector, which will be used to align matrix X direction.
  + It can be set to none, or the curvature.

## OMFilterMesh

### Overview

The module filters the pre-computed local projections matrices by the arbitrary filter matrix (convolution). The result (filtered valu efor the center of each matrix) is saved in the feature vector of each vertex.

### Parameters

* -weight
  + Includes matrix weighting - matrix of the same size multiplying each element.
* -inputW
  + Necessary argument if weighting is present - specifies file with filter kernel.
* -filter
  + Includes matrix convolution into list of filters to be computed.
* -inputF
  + Necessary argument if filter is present - specifies file with filter kernel.
* -min
  + Includes minimal element computation into list of filters to be computed.
* -max
  + Includes maximal element computation into list of filters to be computed.
* -mean
  + Includes mean computation into list of filters to be computed.
* -median
  + Includes median computation into list of filters to be computed.
* -distance
  + Includes abs(min, max) computation into list of filters to be computed.
* -variance
  + Includes variance computation into list of filters to be computed.

ATTENTION: The matrices must be computed by the ComputeMatrices module!

## OMSmoothMesh

### Overview

Module smoothes an input mesh. It uses also methods implemented in OpenMesh (Jacobi – Laplace smoothing) and the Local Projections based approach.

### Parameters

All parameters are voluntary, but following settings can be specified:

* -method methodname
  + Method used for smoothing a mesh
    - Jacobi – Laplace (value: Jacobi)
    - Local projections method
* -iterations no
  + Number of smoothing iterations
  + Should be a number > 0
* -continuity arg
  + Specifies a continuity which must be preserved during smoothing
  + Three possibilities (C0, C1, C2)
* -smooth arg
  + Specifies a component to be smoothed.
  + There is three options:
    - normal: Smoothing will process normal directions
    - tangential: Smoothing will process tangential directions
    - both: Smoothing will process both, normal and tangential directions

Default settings of this module are one pass of Jacobi-Laplace smoothing algorithm with C1 continuity and affecting both, normal and tangential directions.

ATTENTION: If the matrix methods are selected, the matrices must be computed by the ComputeMatrices module!

## OMSubdivideMesh

### Overview

This module executes a fraction of methods implemented in OpenMesh, especially sqrt(3} subdivision algorithm and Loop algorithm. In Openmesh, there are big possibilities to set up a subdivision algorithm (adaptive subdivision) nevertheless, module implements only uniform methods (n iterations of all faces subdivision).

### Parameters

All parameters are voluntary, but following settings can be specified:

* -method methodname
  + Parameter specifies used subdivision method
  + In all cases, there are uniform subdivision methods
  + Values:
    - Sqrt(3) subdivision method (value sqrt3)
    - Loop subdivision method (value loop)
* -iterations no
  + Specifies number of iterations of subdivision
  + Should be a number > 0

Default settings is one pass of sqrt(3) algorithm.

## OMCurvature

### Overview

Module computes curvature information on vertices. A model derived from ModuleMesh must be used, because it contains curvature and curvature magnitude vertex properties. Specified by parameter, four types of curvature can be computed:

* Minimal curvature and its direction
* Maximal curvature and its direction
* Mean curvature and its normal vector (vector perpendicular to vertex normal and mean curvature direction) – it is possible to compute mean curvature direction vector then.
* Gauss curvature (there is no direction for Gauss curvature of course)

All computations can be done by the analytical approach, or the Local projections approach.

### Parameters

Module has simple parameters, which describes what information should be computed:

* -curvature name
  + Type of curvature to be computed
  + There is eight possibilities (mean, gauss, min, max, matrix mean, matrix min, matrix max and matrix gauss)

Default settings is mean curvature with its normal.

ATTENTION: If the matrix methods are selected, the matrices must be computed by the ComputeMatrices module!

## OMDetectEdges

### Overview

The module provides the experimental computation of the mesh edges.

### Parameters

All parameters are voluntary, but following settings can be specified:

* -size size
  + Specifies matrix size in mesh space. This means the double precision number of the length of matrix.
* -resolution res
  + Specifies square matrix dimension in one direction.
  + Must be odd number (for filtration purposes)
* Relative true/false
  + Use this argument if you want to specify matrix size relatively to the median of edge lengths.
* -xdir dir
  + Specifies vector, which will be used to align matrix X direction.
  + It can be set to none, or the curvature.

## OMComputeDescriptors

### Overview

The module provides the mesh descriptor computing capabilities. The feature points and their descriptors (SIFT) are computed and exported into the file.

### Parameters

* -count
  + Specifies the count of feature points.
* -size
  + The size of SIFT matrices relative to the median of the edge lengths.
* -resolution
  + Specifies square matrix dimension in one direction (for SIFT descriptor computation).
* -diameter
  + The SIFT feature point diameter.
* -filename
  + Specifies the output file name with features.

ATTENTION, the matrices must be computed by the OMComputeMatrices module.

### Output file format

Th output file is produce to get compatibility with the other mesh descripting algorithms. The output file is the .txt file, where each row corresponds to each feature vertex and its descriptor. The row format follows:

ID SIFT\_1 SIFT\_2 … SIFT\_128

Where ID is the ID of vertex and SIFT\_XXX are the SIFT vector values. The values are separated by spaces.

# Examples of use

This section provides several possibilities to use of the models provided. The sections is only for demonstrational purposes.

* Computation and visualization of mean value
  + omloadfile -file input.stl | omcomputematrices -size 2.0 -resolution 5 -relative | omfiltermesh -mean | omviewer -visualise vattributes -component 0
* Computation and visualization of feature points and descriptors
  + omloadfile -file input.stl | omcomputematrices -size 1 -resolution 5 -relative | omcurvature -curvature gauss | omcomputedescriptors| omviewer -visualise curvature
* Mesh smoothing using the local projections method
  + omloadfile -file input.stl | omcomputematrices -size 1 -resolution 5 -relative | omcurvature –curvature mean | omsmoothmesh –method local | omsavefile –file output.stl

# References

[1] L. Kobbelt. OpenMesh website. [www.openmesh.org](http://www.openmesh.org). Accessed 12th November 2010

[2] Medical Data Segmentation Toolkit. [mdstk.sourceforge.net](http://mdstk.sourceforge.net). Accessed 12th November 2010

[3] OpenSceneGraph. [www.openscenegraph.org](http://www.openscenegraph.org). Accessed 12th November 2010

[5] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. Technical Report RC-19860, IBM Research, December 1994. (also in Proceedings ICCV’95).

[6] M. Desbrun, M. Meyer, P. Schröder, and A. Barr. Discrete differential geometry operators in 2d and 3d. [www.cs.caltech.edu/~mmeyer](http://www.cs.caltech.edu/~mmeyer). Submitted, 2000.

[7] Hulík, R., Kršek, P.: Local Projections Method and Curvature Approximation on 3D Polygonal Models, In: Communications proceedings of WSCG, Plzen, CZ, UPGM FIT VUT, 2012, s. 223-230, ISBN 978-80-86943-79-4

[8] Hulík, R.: 3D Polygonal Model Curvature Approximation Based on Local Projections Method, In: Sborník prací konference a soutěže Student EEICT 2012, Brno, CZ, UPGM FIT VUT, 2012, s. 436-440, ISBN 978-80-214-4462-1