# ViennaMesh

v1.0
User Manual

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

Developers:

Franz Stimpfl
René Heinzl
Philipp Schwaha

Current Maintainers:

Johann Cervenka
Josef Weinbub

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

Due to increasing computational power with a vast performance even in desktop systems, new possibilities for multiphysics simulations arise, which intensify the tension placed on mesh generation. On the one hand, this results in an increased requirement for the robustness of the algorithms employed, while on the other hand, it also forces the use of high performance methodologies in order to reduce meshing times. The quality of the mesh is not only critically important to the quality of the calculated results: failure to properly control the meshing process can also jeopardize or even completely prevent simulation. Since meshing is the first initial step of the simulation flow, all subsequent results depend on this fundamental step.

A parallel meshing and adaptation approach, combining Delaunay and advancing front algorithms suitable for finite volume and finite element discretization schemes for three dimensions, has therefore been developed [1, 2, 3, 4].

Parallelization and the robustness of the algorithms are facilitated by employing a rigorous surface treatment, which not only enforces the prescribed quality criteria such as the Delaunay property, but also allows the decoupling of the subsequent parallel volume meshing steps. This decoupling is of fundamental importance to the volume meshing step. If the surface remains consistent, there is no need to exchange any data between the individual threads of the volume meshing process. This also enables the inherent utilization of many-core CPUs. Parallelization is implemented by suitably combining multiple programming paradigms and following modern design guidelines, which is necessary in order to keep the development on multi-core processors as simple as possible, while not forsaking any of their computational power. The use of multi-core processors can drastically reduce the mesh generation time as well as the time for execution of subsequent modules, e.g., a linear solver. The availability of high quality and robust high performance mesh generation tools is therefore of utmost importance.

# 2 Installation

By default, executables of all of the utilities are available in the

```
bin/
```

folder. The executables are statically linked to the used external libraries. Therefore they should be ready to use. However, if ViennaMesh has to be rebuild, this section provides the required informations.

## 2.1 Folder Hierarchy

The package is modularized, meaning, that the tools are available as distinct applications. Therefore further development can be specifically oriented. In the following, the folder hierarchy is shown.

```
bin/                    :: this folder contains all built executables
doc/                    :: the manual sources are here
examples/               :: an example input file is provided
hull_adaptor/           :: source files for the hull mesh adaption tool
hull_converter/         :: source files for the mesh conversion utility
hull_orienter/          :: source files for a simple orientation repair tool
mesh_classifier/        :: source files for a mesh quality evaluation utility
volume_mesher/          :: source files for the volume mesher
LICENSE                 :: the LGPL license file
build_all.sh            :: script which builds all tools
clean_all.sh            :: script which removes all build related files
```

## 2.2 Dependencies

The whole package comes with almost all dependent software libraries. However, it requires the Boost libraries [5] to be present.

ViennaMesh only requires the header files of the Boost libraries.

By default, the build environment expects the Boost libraries to be present in the system path

```
/usr/include
```

If the Boost libraries are present in a location different to the system path, it can be mentioned at the build scripts at the different utilities. For example, the build script of the volume mesher looks like this:

```
#!/bin/bash

# configure waf
#
./waf configure

# build
#   note: waf uses automatically all available cores
./waf --progress

# strip executable and copy to bin folder
#
strip build/volume_mesher
cp build/volume_mesher ../bin/
```

To specify a Boost path, the configuration step has to be altered.

```
# configure waf
#
./waf configure --boost-includes=${HOME}/libraries/boost_1_45_0
```

Note, that the above external Boost path example is based on the assumption, that a Boost package has been downloaded and extracted. The path mentioned therefore points to the folder, which contains a `boost/` subfolder. This subfolder contains all the header files of the Boost project. However, the used Boost installation path can be verified by the configuration line of the build process. The line

```
Checking for boost include path     : /home/weinbub/libraries/boost_1_45_0 (ver 1_45)
```

provides information about the detected and used Boost version. If the path is wrong, the color of the path changes from `green` to `yellow`, and an error message is shown:

```
Checking for boost include path     : not found
boost headers not found! (required version min:  max: )
(complete log in /home/weinbub/git/ViennaMesh/hull_converter/build/config.log)
```

Further note, that the presented procedure of providing an external Boost installation is analogous with all the other ViennaMesh tools.

## 2.3 Building

A script is provided which builds all available executables. Executing the script

```
./build_all.sh
```

builds all utilities and moves the executables to the

```
bin/
```

folder.

Note, that the build system is based on Waf [6].

# 3 Usage

Based on the provided input mesh in the folder

```
examples/
```

the complete toolchain is introduced.

## 3.1 Conversion

First, the input file has to be converted to a file format which is used throughout the toolchain. We consider that all executables are being present in the

```
bin/
```

folder, and the working directory is the root folder of the package.

To convert the file, the following command has to be executed

```
bin/hull_converter examples/device.hin device.gau32
```

## 3.2 Hull Orienter

It may happen, that the hull mesh files are not oriented. This most probably results in a failing hull adaption processing step. This is exactly the case with the present input mesh. Therefore, prior to any further adaption or volume meshing steps, the mesh has to be oriented.

> The hull orientation utility is a prototype. Therefore it may introduce problems on perfectly fine input meshes. Only use it, when the hull adaption process fails for a particular input mesh.

This can be done by the following command.

```
bin/hull_orienter device.gau32 device_oriented.gau32
```

For further meshing steps, the oriented mesh shall be used.

## 3.3 Hull Adaption

The converted, oriented hull mesh can now be processed by the hull mesh adaption tool. This utility increases the quality of the hull mesh, and by doing so, it significantly improves the quality of the generated volume mesh, which will be generated based on the adapted hull mesh.

```
bin/hull_adaptor device_oriented.gau32
```

Note, that this utility has no option for a specific output mesh name, it always produces output meshes with the filename

```
surface_mesh.gau32
```

Therefore, we rename the adapted output mesh to something like this

```
mv surface_mesh.gau32 device_adapted.gau32
```

## 3.4 Volume Mesher

The adapted hull mesh can now be volume meshed by the following command:

```
bin/volume_mesher device_adapted.gau32 device_adapted.gau3
```

## 3.5 Mesh Classifier

The resulting volume mesh can now be investigated regarding the quality of the mesh. Therefore a utility has been developed which evaluates the quality of a mesh [4, 3]. This can be done by the following command:

```
bin/mesh_classifier device_adapted.gau3
```

The tool outputs a statistics on the different mesh element types. In the following, the mesh element type statistics is presented of the volume mesh under consideration.

```
---------------------------------------------
  cap       0.604686 %     <-- a small number is good
---------------------------------------------
  needle    0.748986 %     <-- a small number is good
---------------------------------------------
  round     80.0866 %      <-- a large number is good
---------------------------------------------
  slat      2.7898 %       <-- a small number is good
---------------------------------------------
  sliver    6.98138 %      <-- a small number is good
---------------------------------------------
  spade     1.30557 %      <-- a small number is good
---------------------------------------------
  spindle   0.542843 %     <-- a small number is good
---------------------------------------------
  wedge     6.94015 %      <-- a small number is good
---------------------------------------------
```

Of those presented types, the `round` and the `sliver` are of particular importance. The `round` type is considered good, because the dihedral angles of the tetrahedron are in a good mid-range. Those types of elements are usually good for discretization schemes, like Finite Volume Methods or Finite Element Methods.

On the contrary, a `sliver` tetrahedron is a highly degenerated one. Consequently, a small number is desirable. Aside from the `sliver` element type, there are other degenerated elements. Obviously, the number of those should be as small as possible.

Note, that the mesh classifier tool also produces two files, one containing the present dihedral angles, and the other contains Latex code. The file containing angles can be used, for example, to create histograms. Therefore, the angle distribution throughout the whole mesh can be investigated. The file containing Latex code, contains a table with the overview of different mesh element types and a histogram like figure representing the different mesh element types. Those outputs significantly support investigations of the mesh quality.

# 4 Results

This section should provide an overview of the capabilities of the introduced mesh generation, adaption and classification software package.

## 4.1 CMOS structure

In the following, different views of the original and the adapted volume mesh are presented. Additionally, a comparison of the mesh quality is depicted.

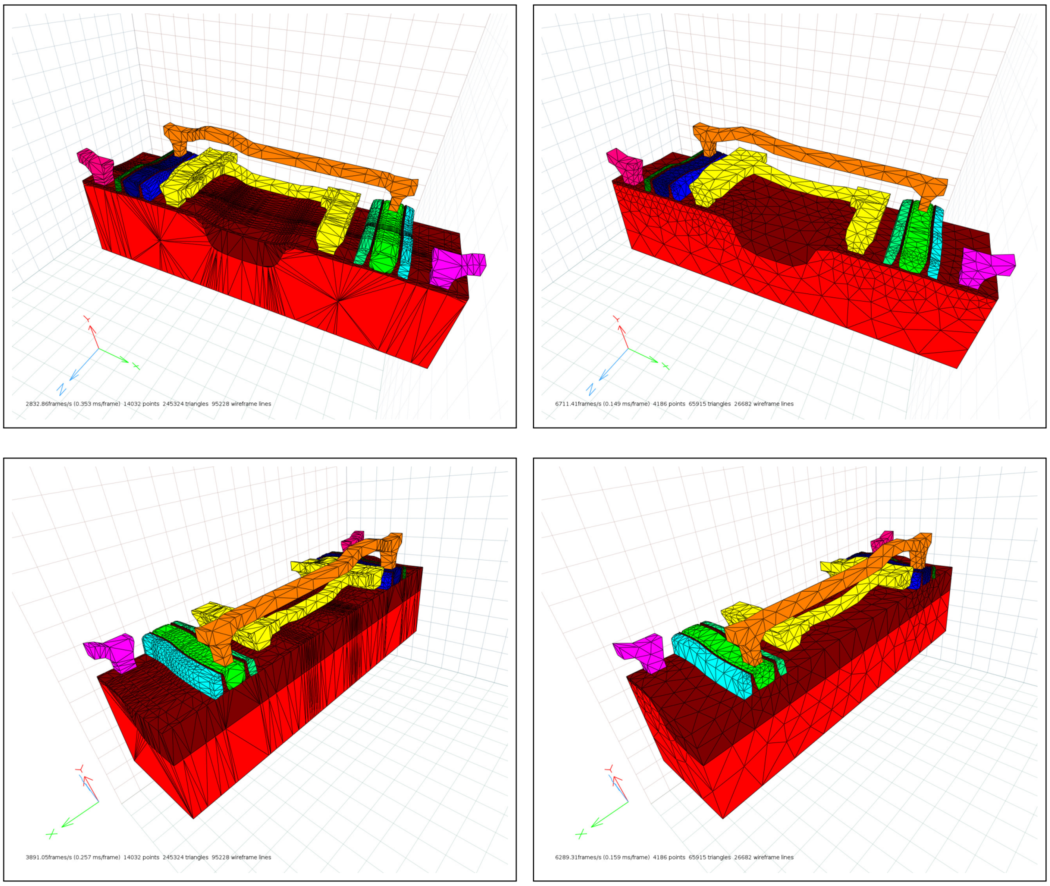


Figure 1: A CMOS structure. **left:** The initial mesh is shown. **right:** The adapted mesh is shown.

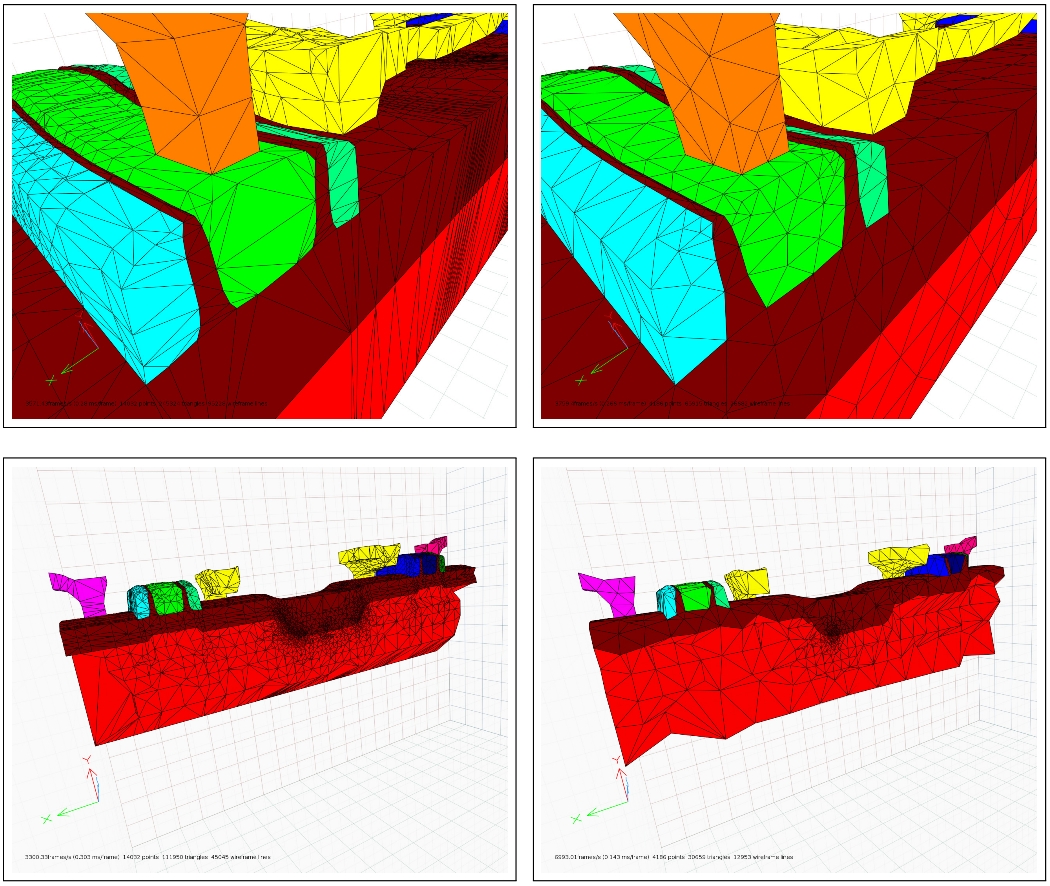



Figure 2: A CMOS structure. **left:** The initial mesh is shown. **right:** The adapted mesh is shown.

| | **original** | **adapted** |
|---|---|---|
| **vertex size** | 14032 | 4186 |

| | | |
|---|---|---|
| cap | 485 | 0.62% |
| needle | 3557 | 4.55% |
| round | 51745 | 66.16% |
| slat | 4363 | 5.58% |
| sliver | 4037 | 5.16% |
| spade | 1082 | 1.38% |
| spindle | 513 | 0.66% |
| wedge | 12435 | 15.90% |

| | | |
|---|---|---|
| cap | 108 | 0.52% |
| needle | 393 | 1.88% |
| round | 15842 | 75.63% |
| slat | 817 | 3.90% |
| sliver | 1399 | 6.68% |
| spade | 320 | 1.53% |
| spindle | 140 | 0.67% |
| wedge | 1929 | 9.21% |

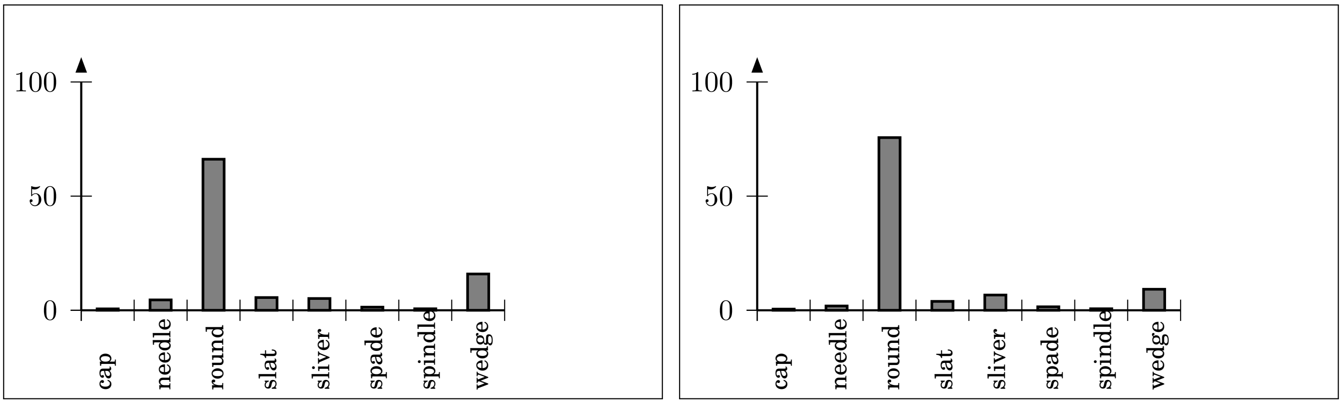



Figure 3: A mesh quality analysis of the original and the adapted CMOS structure meshes. The analysis of the initial mesh and the adapted mesh is depicted, on the **left right**, respectively. Apparently, the mesh adaption process introduces a bit more `slivers`, on the other hand, a considerable amount of `rounds` is introduced too. This result is quite good, as the significant reduction of the vertex size has to be taken into account.

## 4.2 Etched Hole with Cylindrical Mask

In the following, different views of the original and the adapted volume mesh are presented. Additionally, a comparison of the mesh quality is depicted.

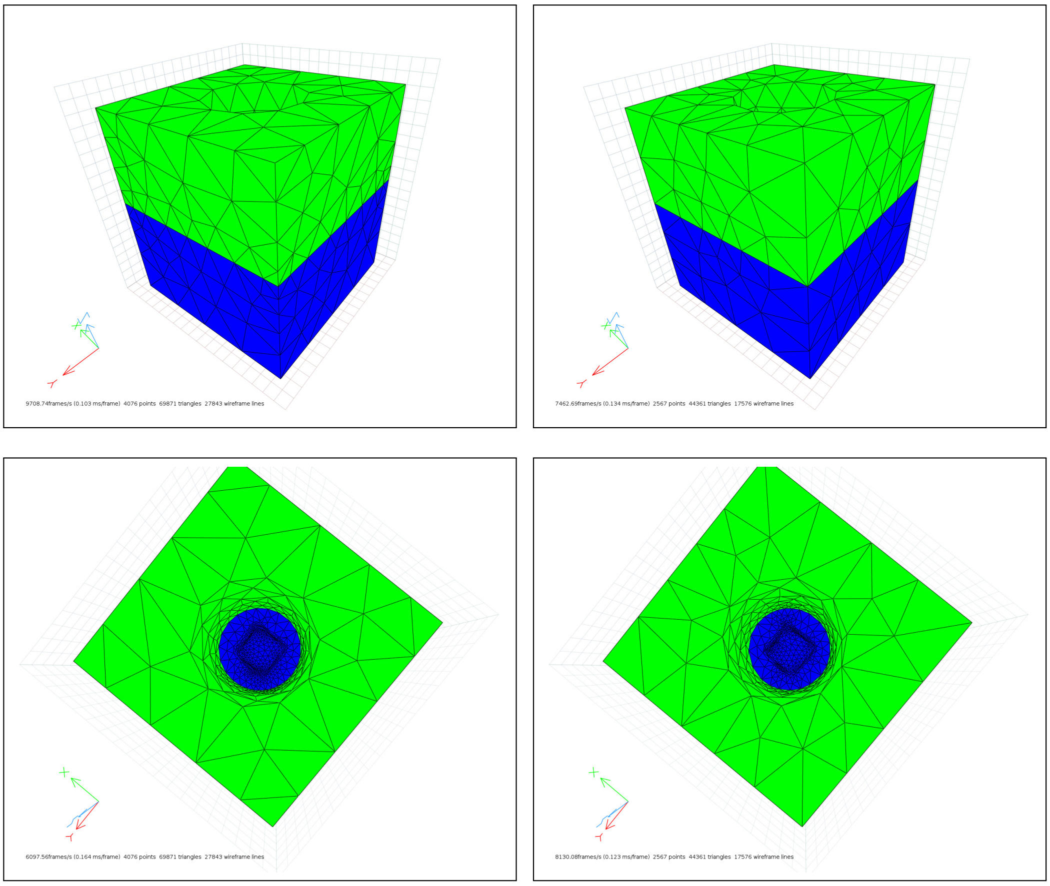


Figure 4: A CMOS structure. **left:** The initial mesh is shown. **right:** The adapted mesh is shown.

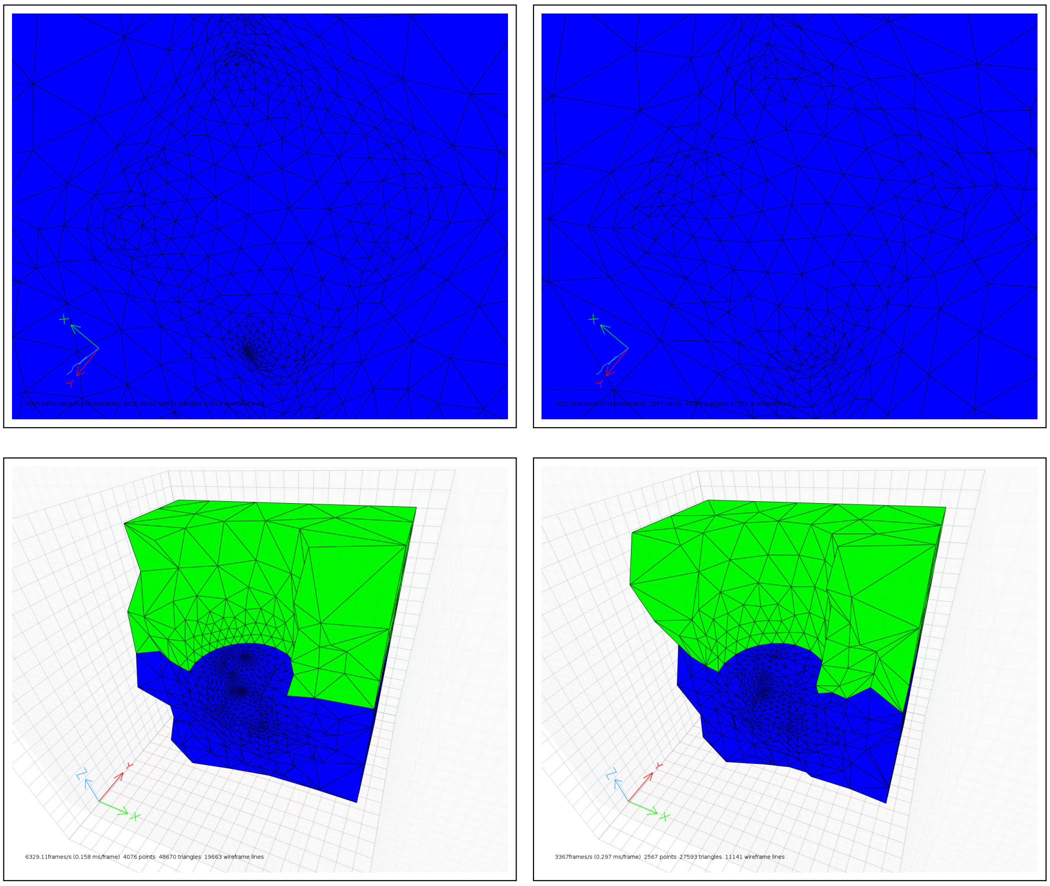
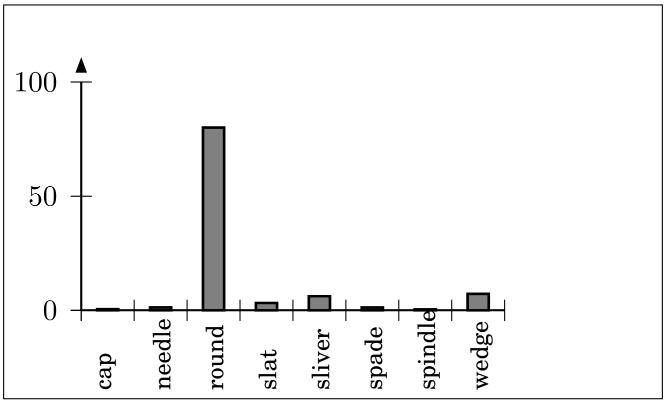


Figure 5: Etched Hole with Cylindrical Mask. **left:** The initial mesh is shown. **right:** The adapted mesh is shown.

| | original | adapted |
|---|---|---|
| **vertex size** | 4076 | 2567 |

| | | |
|---|---|---|
| cap | 116 | 0.51% |
| needle | 290 | 1.27% |
| round | 18230 | 80.00% |
| slat | 728 | 3.19% |
| sliver | 1408 | 6.18% |
| spade | 283 | 1.24% |
| spindle | 96 | 0.42% |
| wedge | 1636 | 7.18% |

| | | |
|---|---|---|
| cap | 55 | 0.38% |
| needle | 169 | 1.18% |
| round | 11296 | 78.60% |
| slat | 483 | 3.36% |
| sliver | 964 | 6.71% |
| spade | 191 | 1.33% |
| spindle | 93 | 0.65% |
| wedge | 1121 | 7.80% |



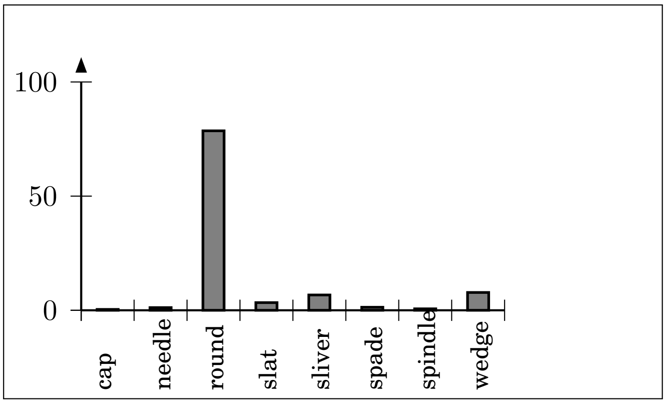




Figure 6: A mesh quality analysis of the original and the adapted Etched Hole meshes. The analysis of the initial mesh and the adapted mesh is depicted, on the **left right**, respectively. Apparently, the mesh adaption process introduces a bit more `slivers`, on the other hand, a significant amount of rounds is introduced too. The mesh elements remained more or less the same, however, the mesh vertex size is again considerably decreased.

# 5 License

As ViennaMesh is based on Netgen [7] and DeLink, [8] the licenses are therefore inherited. Consequently, ViennaMesh is published under the `GNU LESSER GENERAL PUBLIC LICENSE (LGPL)` [9]. The complete license text can be found in the file

```
LICENSE
```

located in the root folder of the ViennaMesh package.

## References

[1] F. Stimpfl, R. Heinzl, P. Schwaha, and S. Selberherr, "A Robust Parallel Delaunay Mesh Generation Approach Suitable for Three-Dimensional TCAD," in *International Conference on Simulation of Semiconductor Processes and Devices, Hakone*, 2008, pp. 265–268.

[2] ——, "High Performance Parallel Mesh Generation and Adaptation," in *Workshop on State-of-the-Art in Scientific and Parallel Computing, Trondheim*, 2008.

[3] F. Stimpfl, R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser, "Multi-mode Mesh Generation for Scientific Computing," in *European Simulation and Modelling Conference, Malta*, 2007, pp. 425–429.

[4] R. Heinzl and T. Grasser, "Generalized Comprehensive Approach for Robust Three-Dimensional Mesh Generation for TCAD," in *International Conference on Simulation of Semiconductor Processes and Devices, Tokyo*, 2005, pp. 211–214.

[5] "Boost C++ Libraries." [Online]. Available: http://www.boost.org/

[6] "Waf - The flexible build system." [Online]. Available: http://code.google.com/p/waf/

[7] "Netgen." [Online]. Available: http://www.hpfem.jku.at/netgen/

[8] "deLink." [Online]. Available: http://www.iue.tuwien.ac.at/software.0.html

[9] "GNU Lesser General Public License." [Online]. Available: http://www.gnu.org/licenses/lgpl.html