Hexi 1.1 Documentation
Part 1

by

Gary Bollenbach
October 26, 2021

Text that is highlighted conforms with Outline or Bookmarks listed in the dropdown at page top (when viewing online). See symbol:  or 

Introduction

Welcome to Hexi, a toolchain for obtaining hexahedral mesh. In 2019, a significant academic paper was published, Ref [1], hereafter referred to as the Paper. The Paper's authors chose to make the enabling code available to the public, and Hexi makes use of the project's prime executable. Thanks to the Paper's authors for their generosity.

Perhaps the first question that needs to be addressed is computation time. At the time this document was created, supplemental resources of the Paper project were available, including a directory of sample meshes, along with their hexahedral analogs, and execution timings. These ranged from 'bone' at 3.25 min to 'red_circular_box' at 1193 min. Creation of hexahedral mesh requires CPU power, and time.

Platform-specific notes:

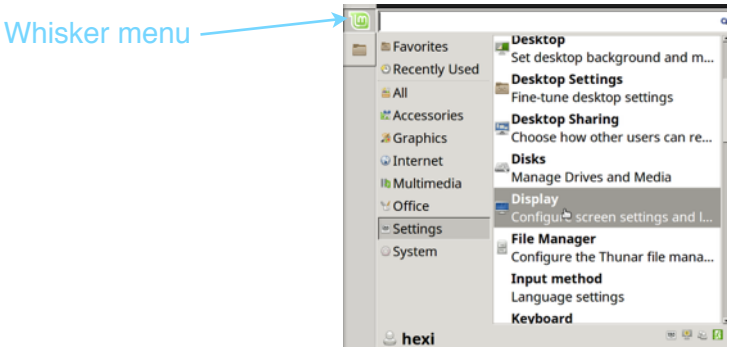
Docker. Presently the preferred platform. It runs the fastest, and can reside as a client application on the newest OS, taking advantage of recent kernel advances, while meanwhile retaining the ability to operate old software. Docker is really the only option with reasonable performance at the moment. (The readme file in the Docker folder has instructions for downloading the Hexi executable in Docker format.)

Linux. A stable option. For those used to interacting with Linux, a familiar interface, though far slower than Docker.

Windows. The Sample Mesh comparison table in Part 3 testifies that Windows partnered with Docker (on a virtual machine) is the best combination, numerically, at present, of those combinations tested.

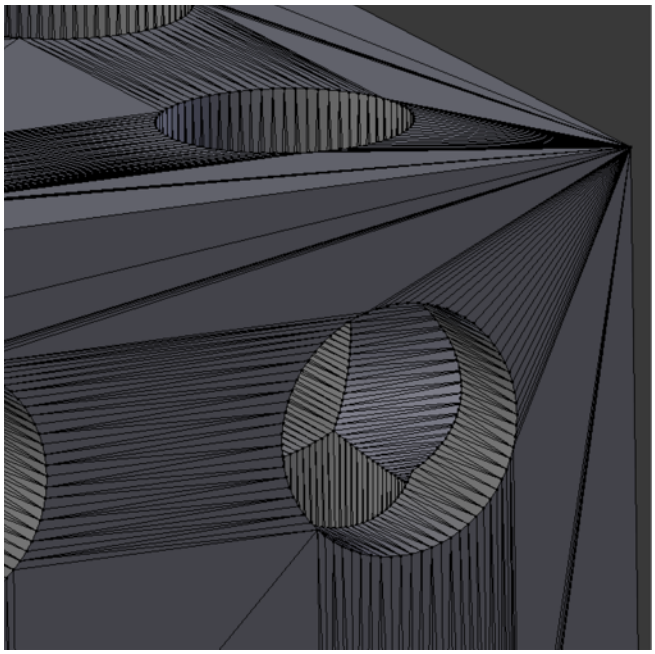
The default desktop password for Hexi is hexi.

Hexi ships with display resolution set up for 3200 X 1800. This resolution is possible because of an installed Nvidia control package. Whatever graphic display hardware is installed, it is assumed that Whisker menu setting adjustments will be available and accessible, to adapt the display parameters as required.

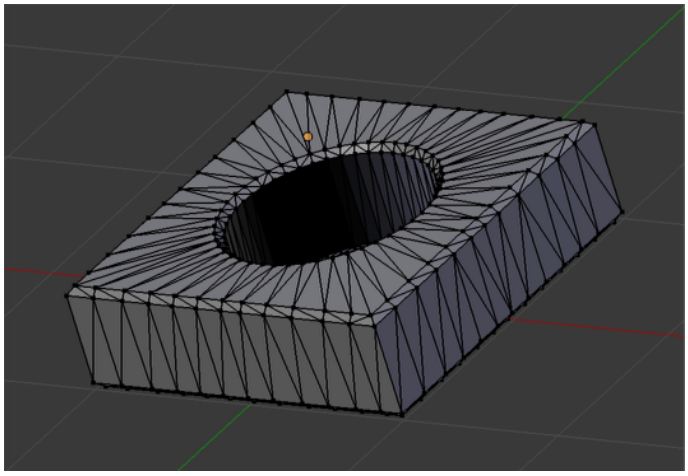


Obtaining and preparing a surface mesh

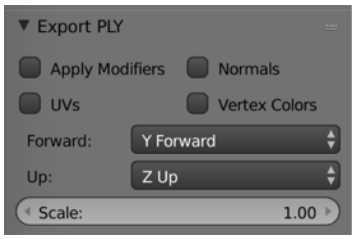
At the top of the first page of the Paper are shown two examples, the one on the right designated as ‘cheese4’ in the supplemental materials. These two examples showcase the abilities of RPHM well, demonstrating that even a mesh model with a starting edge ratio of 70.7, like ‘cheese4’, can be used as input for the creation of a well-proportioned hexahedral mesh. Mesh models which contain sharp edges, a category which includes most models of interest, are normally run with a Feature file. The subject of Feature files is covered in Part 2 of the docs. For mesh models which do not require a Feature file, or even those that do, it is sometimes useful to remesh, a process of rebuilding to make the surface triangles more uniform. The all-inclusive version of Hexi includes five programs which can be used for remeshing: Netgen, Ref [3], Meshlab, Ref [4], GTK-Remesher, Ref [5], OpenFlipper, Ref [6], and mmgs, Ref [7]. I will use Netgen, which I think is the easiest to handle.



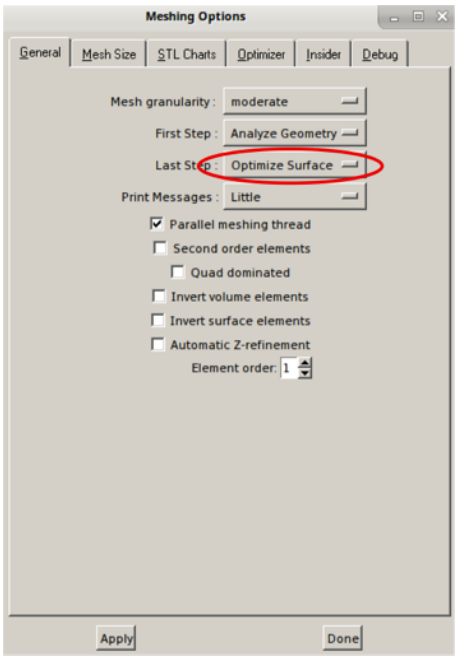
Before I can remesh, I must first have a mesh. At right is shown a pierced cube created in Blender. It will serve as the model for the mesh to be made and processed in the demo. As you can surmise, it is a surface mesh composed of all triangles.



Getting the mesh into Netgen. Netgen needs to have an .stl file to open. Netgen can usually read exported .stl files of reasonable quality from Blender provided that they are in ASCII format. Otherwise, .stl output from Paraview is normally reliable.

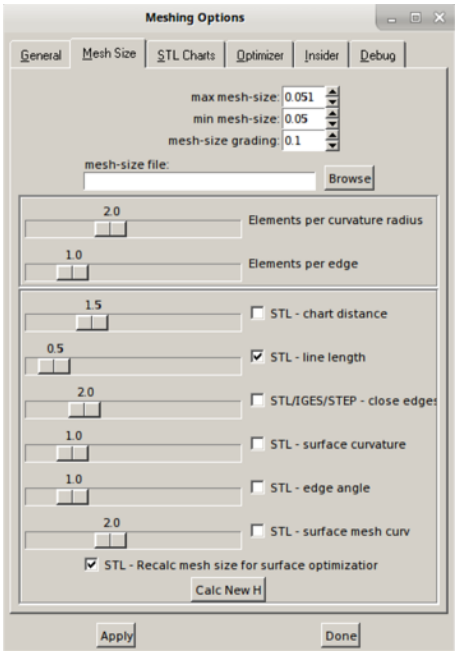


Once imported into Netgen, I open the mesh options menu. In the first tab of the menu, the Last Step option needs to be set to Optimize Surface, so that only a surface mesh will be generated.

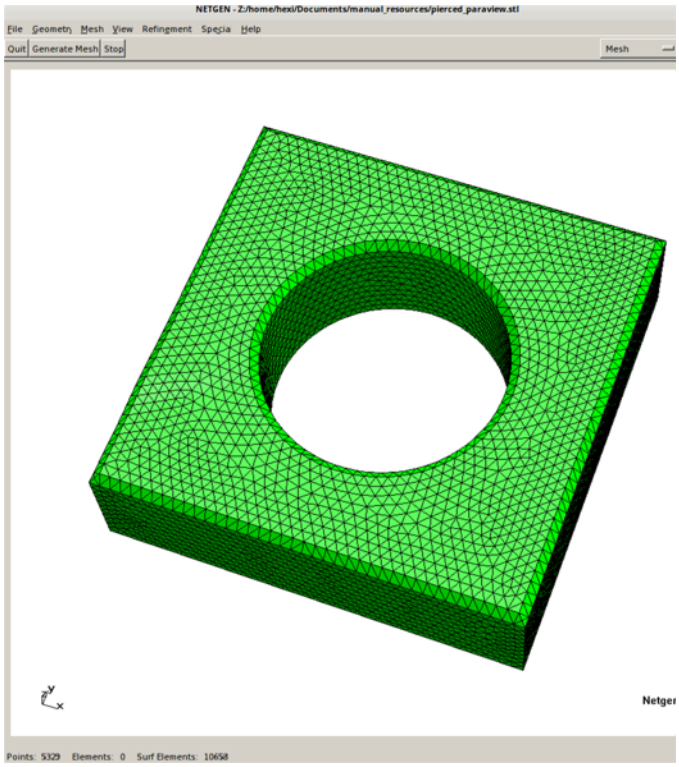


The Mesh Size tab is the one with the critical effect. The max mesh-size and min mesh-size fields regulate not only the density of the elements, but also, I suspect, the approximation to equilaterality. Following this guess, I set the max and min close together. And choose the smallest mesh-size grading.

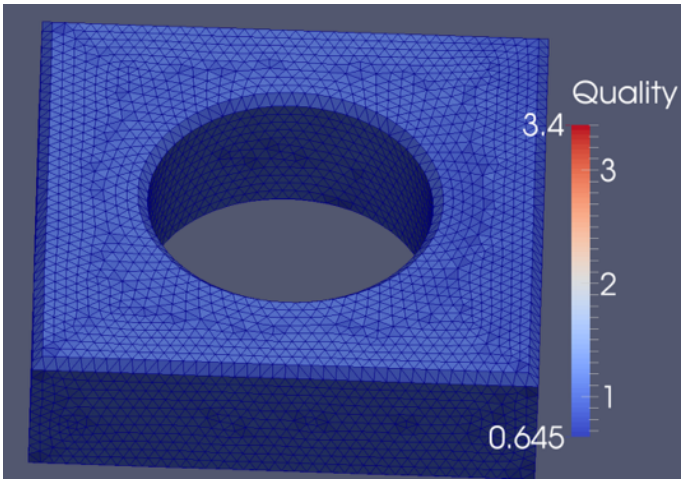
Based on personal testing, I believe that the single option of line length equals or exceeds the quality of output achieved by including several parameters.



Netgen puts out a nice looking mesh. To see how good the quality actually is, I need to take it into Paraview, Ref [9].



In Paraview, I import the .stl output file from Netgen. Note that it is necessary to press the green Apply button to view an opened item. The first time the Mesh Quality filter is used, it has to be selected from the alphabetical list; thereafter it will be located under Recent. The drop-down box for triangle quality must be opened and Scaled Jacobian selected. Then pressing Apply followed by Quality > Show and (always, by habit) Rescale, gives the view shown at right.



The quality level shown here is excellent for most jobs, and in Part 3 are shown some cases where a much lower Scaled Jacobian is found to be acceptable.

Running Robust Pure Hex Meshing

Char	Type	Switch descrip	Descrip from src	Default value or speculative
“-h” **“--h”**		Print this message and exit		
“--ch”	TEXT	functionality choice		Seems to expect “GRID”
“--in”	TEXT	Input mesh		Filename including extension
“--out”	TEXT	Output mesh		Filename including extension
“--o”	UINT	Octree meshing	bool octree = true	1
“--n”	INT	num-cells for voxel meshing		skip for octree
“--h”	FLOAT	Hausdorff_ratio_t	default = 0.005	paper = “epsilon”; regulates strictness of conformity to input boundary; larger number -> fewer cells
“--e”	FLOAT	edge_length_ratio	ℓ (as in “log”)	paper = “ℓ”, i.e. “ell’; accepted range -> 14.0 to 30.0, though with some models a lower value than 14.0 can be used; larger number -> fewer cells
“--w”	FLOAT	weight_opt	double weight opt = 1	Optional switch; the --fw switch is recognized whether the --w switch is set or not.
“--fw”	FLOAT	feature_weight_opt	double LAMDA_FEATURE_PROJECTION = MESHRATIO*args.feature_weight; default = 0.05	regulates strictness of conformity to edge definition; larger number -> tighter adherence to edge lines
“--r”	UINT	bounding box style	bool pca_oobb = true	default = 1; refers to “principal component analysis -- optimal oriented bounding box”; an accepted standard
“--s”	INT	scaffold_type	int scaffold type = 1; types: 1.box scaffold free boundary; 2. layered scaffold free boundary; 3. box scaffold fixed boundary; 4. layered scaffold fixed boundary	The default appears to be faster; however, if it produces drop-outs or incomplete edges, try --s 2.
“--f”	UINT	Hard_Feature		Feature file is recognized whether this switch is set or not. Either it places special emphasis on edge definition, or it is redundant.
“--Iter”	INT	optimization Iteration_Base	default = 3; capital ‘I’ as in India	larger than default is low cost in terms of time added; does not increase detail but can raise quality level; can be used in lieu of Hausdorff

RobustPureHexMeshing, the core of Hexi, is a console application, at least as I built it. It has the thirteen switches shown above. I do not pretend to understand very much about the significance or range of the switch settings, but some deductions can be made by looking at the Paper and its source code. The application is split into two basic operational paths: one for conformance mesh, and one for voxelization. The voxelization (producing a mesh bounded by cubic surfaces) is useful because it speeds calculations in fields such as Computational Fluid Dynamics. For my own purposes, traditional conformance mesh is the path that holds primary interest.

A typical command to start RPHM might look like:

```
./RobustPureHexMeshing --ch GRID --o 1 --in infile.obj --out outfile.vtk --e 21.0 --f 1 --fw 0.1 --h 0.003
```

The minimum command would specify only the first five entities, up to and including the output file name.

Comments on the console log:

1. Immediately after a run begins, RPHM ascertains whether the input file is a manifold mesh. If not it halts with a message. (Resources for inspecting a .stl protofile, such as are found in Blender, are a good place to try to make repairs.)

2. Next the program needs to generate a node map of the mesh. The projected number of cells as estimated by the process can be seen to gradually increase until a satisfactory map is found. If six tries are not enough to generate the map, RPHM restarts the search. It is possible for certain characteristics of an input mesh to cause an infinite loop of restarts. After an unsuccessful setup spanning many restarts, it may be best to quit and reconfigure the model in some way.

3. Following the node map section is the curve map test. The first curve map examination line in the log serves to judge the compatibility of the Feature file, if present, with the mapping algorithm. If information about line lengths is printed immediately after the curve mapping test line, it means that the Feature file is satisfactory. If the Feature file is unsatisfactory, the program crashes with a segmentation fault, and the Feature file must be altered (or omitted).

4. Beyond the node and curve mapping tasks comes the patch creation phase. The "patch matched!" announcement appears in the log, and RPHM proceeds to begin building a hex mesh, in a work routine which creates a network of trial hexahedra. At an individual level or on a larger scale, intermediate construction is evaluated for energy level. In mesh parlance, a high energy level is undesired, and iterative action is taken to reduce it. A normal log sequence would be a series of lines showing a reducing trend, such as:

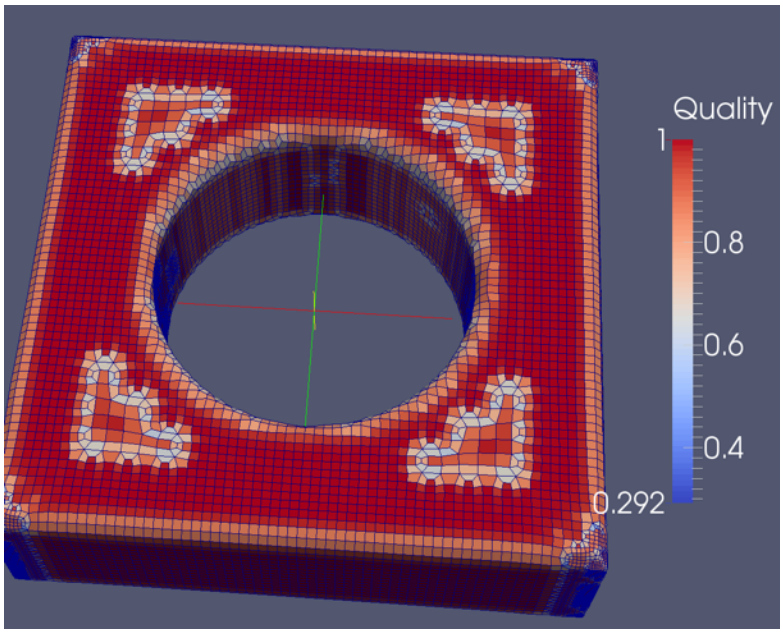
```
energy at iter: 0 :6.91265
energy at iter: 1 :6.8054
energy at iter: 2 :6.7831
energy at iter: 0 :7.53747
energy at iter: 1 :7.08908
energy at iter: 2 :7.02169
energy at iter: 0 :8.01082
energy at iter: 1 :7.6517
energy at iter: 2 :7.54321
energy at iter: 0 :7.3993
energy at iter: 1 :7.33538
energy at iter: 2 :7.31308
energy at iter: 0 :7.27272
energy at iter: 1 :7.25253
energy at iter: 2 :7.2333
```

The overall process of creating a collection of cells with desirable attributes seems to repeat in a cyclical manner, but on an obscure scale and with uncertain interval, sometimes seeming to start over at the beginning.

In some cases, well into the run, the program encounters a situation where the energy is not reduced, even minutely, using the available algorithms, and after 15 or 20 identical iterations the program resorts to a complete restart. This situation presages an anticipated boost in estimated element count in the next developing loop, sometimes a large increase. Escalation of numbers shown on the V H log lines to over a million, a not infrequent occurrence, has so far spelled certain doom for any run in which it appears.

At the completion of a successful run, there is a closing pair of lines in the log which include the time spent by RPHM on the job, similar to the following:

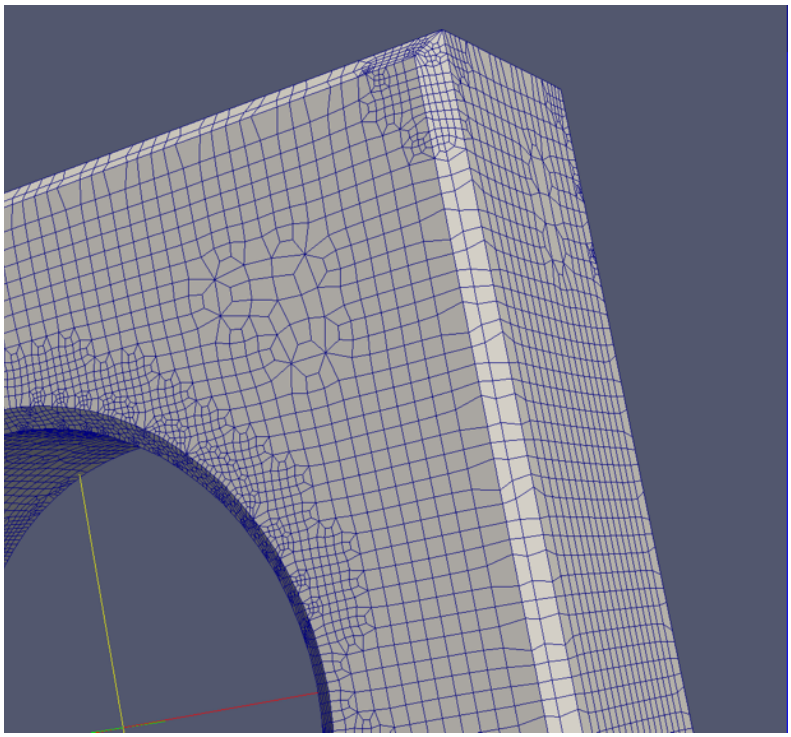
```
done. (took 164438689 ms, END MESHING)
TIMING: 164438689ms
```



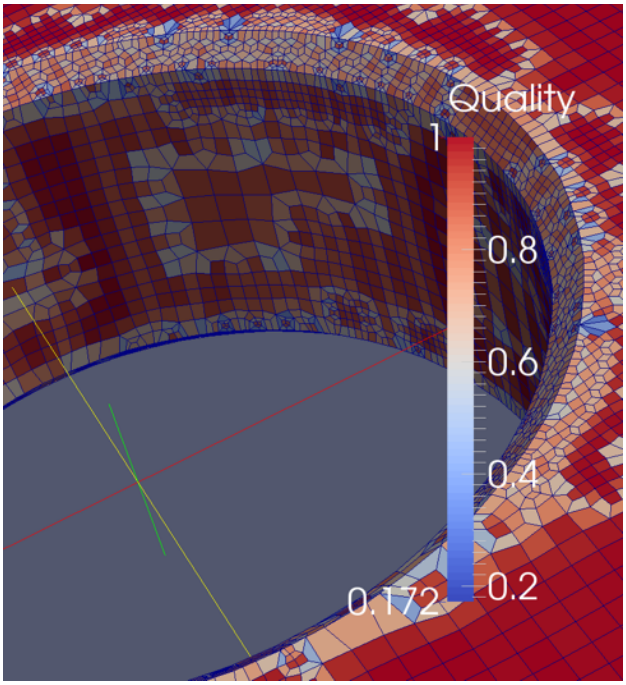
Above is seen an output mesh based on the triangular mesh described above. One of the command switches used is --h 0.0035, which calls for a 0.0035 Hausdorff setting. Although the mesh does not look too bad, notice that the edge which bounds the penetration is rounded. This is because the Hausdorff setting was the only device used to monitor the shape of the edges.

At this point I will mention the inclusion in the mesh of hard feature information. The Paper discusses two classes of features: soft and hard. Soft features are those defined by starting and ending vertices, where the location of intermediate vertices are interpolated algorithmically. Hard (or sharp) features are those edges which must be incorporated into the final mesh either complete or with only linear divisions.

In the pic right the superior handling of the chamfer areas, with defined hard features, may be compared with the semi-radius treatment seen on p. 11.



The pic right suggests that there may be some sacrifice in minimum Scaled Jacobian quality level when hard features are included. (The average Scaled Jacobian in this case is 0.829.)



For information on creating Feature files, refer to Appendix C.

Comments on minimum Scaled Jacobian value

1. Ref [10] is a paper exploring the influence which various quality indicators of the Verdict standard, Ref [11], have on the accuracy of finite element results in a few major categories of FEA problems. It is a statistical study which has as one of its main take-home points the conclusion that minimum Scaled Jacobian quality in a mesh correlates poorly with accurate finite element performance. Of much greater importance is the average Scaled Jacobian quality level of the whole mesh. The average Scaled Jacobian of the test mesh described here is 0.877, an impressive achievement for RPHM. Viewed from the standpoint of Ref [10], there is an implied prediction that the mesh will perform reliably as-is, without attempts at improvement. In considering how far to apply this viewpoint, weighing the caveats put forward in Ref [10] may be advisable.

2. A specific device for eliminating the possible significance of element quality level in a given problem is through adaptive p-refinement (or hp-refinement) of the mesh. The packages associated with Ref [18], PolyFEM, and [20], the deal.II finite element library, both open source, make this method available.

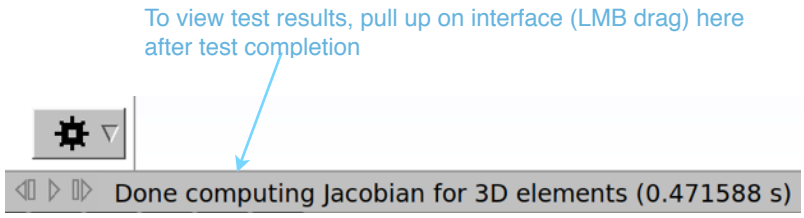
3. A third relevant procedure is the outcome of Ref [19]. The executable in question, complex_simplification, rebuilds the input mesh from basic topological principles. Currently the usage of this app within these docs is limited to the heat exchanger head example in Part 3. The following section describes how to acquire and build this freely available resource.

Improvement demo

Some comments about the Ref [19] project, Robust-Hexahedral-Re-Meshing. It is easy to build the repository on Windows 10. It is only necessary to make sure that CMake is in the environment path (taken care of while CMake is installing), and that at least one Hello-World-type project with .sln has been constructed on Visual Studio. Using VS2019, the first step is to clone the repository, as shown in the pic right. After the clone process has completed, a glance at the bottom tab bar in the main left panel shows the first tab as 'Solution Explorer'. After this tab is selected, the panel changes to contain a list of 31 .sln files. Also reconfigured is the menu bar at the top of the interface, in which a Build menu is now available. The Build All process should be launched. The resulting executable, complex_simplification, should be built in Release mode as well as Debug. Both processes should finish without errors, and the folder ..\out\build\x64-Release should contain an executable.

Validity check

Sometimes elements with an acceptable Scaled Jacobian can have contorted geometry. As a final test it is a good idea to do a validity check of the mesh, using the test that is described in Refs [15] and [16]. The method seems particularly thorough. It is very easy to apply also, because it has been incorporated into the Gmsh application, Ref [17]. The pic below shows how to access the results from the Gmsh interface.



To perform the test, you merely have to load a mesh into Gmsh, then execute the very first of the plugins, in the path Tools > Plugins > AnalyseMeshQuality. In the output, the critical line for the example mesh I am working with here is the one that says:

Info: minJ/maxJ = 0.0148, 0.674, 1 (worst, avg, best)

Since all signs are positive, it means that all elements in the mesh are valid.

Licensing

The Hexi project makes use of source code which is subject to open source licensing. The following sections are applicable:

I. Feature-Preserving-Octree-Hex-Meshing-master

Refers to the principal code within Hexi, comprising both source and binary. The following list of licenses corresponds to copies of the documents themselves in the /home/hexi/Licenses subdirectory:

1. 2-Clause_BSD_License
2. 3-Clause_BSD_License
3. Apache_License_v2.0
4. Creative_Commons_Legal_Code_License
5. GNU_Affero_General_Public_License_v.3
6. GNU_General_Public_License_v2.1
7. GNU_General_Public_License_v3.0
8. Mozilla_Public_License_v2.0
9. OpenGL_Extension_Wrangler_Library_License
10. The_AntTweakBar_License
11. The_Eigen-Intel-MKL_License
12. The_Freeglut_License
13. The_imgui-lua-bindings_License
14. The_Khronos-Valve-LunarG_License
15. The_LLVM_Release_License
16. The_MIT_License
17. The_SIL_Open_Font_License_v1.1
18. The_stb_truetype_License
19. The_Wenzel_Jakob_License
20. The_Zlib_License
21. Zero_Clause_BSD_License
22. The_Minpack_License
23. The_Google_License
24. Not used
25. The_Kronos_License
26. The_Sandia_License
27. The_Geogram_License
28. The_HLBFGS_License
29. The_LUA_License

As a locating tool, the numbering in the list refers to occurrences tracked in the file RPHM-licenses.gnumeric.

II. Mesquite

Refers to the mesh improvement application, present in Hexi as both source and binary. All Mesquite source files are subject to the GNU General Public License, version 3, a copy of which resides in its corresponding folder in the /home/hexi/Licenses subdirectory.

References

[1] Gao, X., Shen, H., Panozzo, D. (2019). Feature Preserving Octree-Based Hexahedral Meshing. Eurographics Symposium on Geometry Processing, Volume 38, Number 5 [Document]
[2] Baum, D., Dupoux, F. (2020). QT-FSArchiver Live-ff-64-0.8.5-18 [Computer software]
[3] Schöberl, J. (1997). NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. Computing and Visualization in Science, 1 (1), pages 41-52 [Document]
[4] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. Sixth Eurographics Italian Chapter Conference, pages 129-136 [Document]
[5] Furhman, S., Ackermann, J., Kalbe, T., Goesele, M. (2010). Direct Resampling for Isotropic Surface Remeshing. Vision, Modeling, and Visualization [Document]
[6] Möbius, J., Kremer, M., Kobbelt, L. (2013). OpenFlipper - A Highly Modular Framework for Processing and Visualization of Complex Geometric Models. Sixth Workshop on Software Engineering and Architectures for Realtime Interactive Systems [Document]
[7] Dobrzynski, C., Frey, P. (2008). Anisotropic Delaunay mesh adaptation for unsteady simulations. Proceedings of the 17th International Meshing Roundtable [Document]
[8] Stichting Blender Foundation. (2016). Blender 2.79b [Computer software]
[9] Kitware, Inc. (2013). Paraview 4.0.1 [Computer software]
[10] Gao, X., Huang, J., Xu, K., Pan, Z., Deng, Z., Chen, G. (2017). Evaluating Hex-mesh Quality Metrics via Correlation Analysis. Eurographics Symposium on Geometry Processing, Volume 36, Number 5 [Document]
[11] Stimpson, C., Ernst, C., Knupp, P., Pébay, P., Thompson, D. (2007). The Verdict Library Reference Manual [Document]
[12] Bollenbach, G. (2019). Lifted 1.4 [Computer software]
[13] Brewer, M., Diachin, L. A. F., Knupp, P. M., Leurent, T., Melander, D. (2003). The Mesquite Mesh Quality Improvement Toolkit. Proceedings of the 12th International Meshing Roundtable, pp. 239–250 [Document].
[14] Bollenbach, G. (2019). Blenbridge 1.20 [Computer software]
[15] Johnen, A., Remacle, J.-C., Geuzaine, C. (2013). Geometrical Validity of Curvilinear Finite Elements. Journal of Computational Physics 233 [Document]
[16] Johnen, A., Weill, J.-C., Remacle, J.-C. (2017). Robust and Efficient Validation of the Linear Hexahedral Element. 26th International Meshing Roundtable [Document]
[17] Geuzaine, C., Remacle, J-F. (2017). Gmsh 4.5.6 [Computer software]
[18] Schneider, T., Hu, Y., Dumas, J., Gao, X., Panozzo, D., Zorin, D. (2018). Decoupling Simulation Accuracy from Mesh Quality. ACM Trans. Graph. 37, 6, Article 280 [Document]
[19] Gao, X., Panozzo, D., Wang, W., Deng, Z., Chen, G. (2017). Robust Structure Simplification for Hex Re-meshing. ACM Trans. Graph. 36, 6, Article 185 [Document]
[20] Arndt D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J-P., Turcksin, B., Wells, D. (2021). The deal.II finite element library: design, features, and insights. Computers & Mathematics with Applications, vol. 81, pages 407-422 [Document]

Appendix A: Building *RPHM*

The following instructions apply to a clean installation of Mint 17.3, with user name hexi. It should also work on Ubuntu 14.04 and spinoffs having apt access to GCC 4.8.4. (Note: the procedure below was tested and found valid on 2021Feb13.)

1.Clean the apt pipeline.

```
sudo apt update
```

2. Install a recent CMake. The source stipulates a minimum CMake version of 2.8.11.2, whereas Mint 17.3 has access to 2.8.12.2, which would be okay, except the old versions of CMake cannot execute a couple of lines below. So it looks like a newer CMake version is indicated. However, building CMake from source is very slow. A quicker option:

```
wget -P /home/hexi/.local \
'https://cmake.org/files/v3.13/cmake-3.13.3-Linux-x86_64.tar.gz'
cd /home/hexi/.local
tar -xf cmake-3.13.3-Linux-x86_64.tar.gz
```

And to append to the environment path:

```
export PATH=/home/hexi/.local/cmake-3.13.3-Linux-x86_64/bin:$PATH
```

3. Install dependencies. This is where GCC 4.8.4 gets installed.

```
sudo apt install build-essential libgl1-mesa-dev libglu1-mesa-dev libxi-dev \
libsuitesparse-dev libxcursor-dev libxinerama-dev libxrandr-dev
```

4. Download the source code repository.

```
wget -P ~/ \
https://github.com/gaoxifeng/Feature-Preserving-Octree-Hex-Meshing/archive/master.zip
```

5. Go back up one level.

```
cd ..
```

6. Extract the compressed source.

```
unzip master.zip
```

7. Enter the resulting directory.

```
cd Feature-Preserving-Octree-Hex-Meshing-master
```

8. Make a working subdirectory.

```
mkdir build
```

9. Go into the new working subdirectory.

```
cd build
```

10. Give CMake some orientation.

```
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
```

11. Build the executable.

```
make -j$(nproc)
```

The new executable, RobustPureHexMeshing, has a size of 12.8 MB.

Windows

The attempt to build RPHM on Windows was successful, using Visual Studio 2019. An executable was generated. The executable is 30 MB in size, and appears to be fully functional. In Task Manager the process is shown as hyperthreading, but the CPU usage is low, and due to this anomaly the efficiency of the executable is questionable.

Appendix B: Troubleshooting

Recapping some troubleshooting events.

1. Non-manifold file. This error was due to a missing face on the input mesh. Easy to spot, but in some instances even a skilled examination in Blender cannot uncover the problem. In such a case treating the file as one destined to be exported to .stl is helpful, and using 3D printing tests and repairs, either in Blender of outside of it, can sometimes remedy the defects in the mesh.

2. Kahan's assertion failed in eigen.
    a. This error was due to two slightly mismatched vertices. Hard to spot. In a Paraview spreadsheet layout, (attribute: Cell Data, column: Cell Type), two quads were seen to be present. Looking at the location of the quads identified the mismatched vertices.
    b. The same error arose from a quad on the input mesh where two triangles should have been. The same diagnostic method was successful.

3. Memory exhausted. This error seems self-explanatory, but in fact seems to be the message issued in the case of more than one exception condition. If memory use climbs at a steady pace of a few GB per second or so, eventually equaling or exceeding the amount available, the bonafide nature of the exhaustion is evident. But if the result is a sudden failure of the run, with memory to spare, and without the expected gradual consumption of memory, then the likely cause may a bad input file. In one observed case some duplicate, spurious edges within a mesh were superimposed on a legitimate set. This can be difficult to diagnose.

4. Killed. Linux memory management, dealing with borderline insufficiency, elected to jettison RPHM. This is a well-documented system management strategy.

5. Segmentation fault. Arose early in the run, exhibiting the following log section:

```
 {{
   Feature Mapping
   node mapping
   curve mapping
   Abnormal program termination: received signal 11 (Segmentation fault)
 }}
```

Comment: crash caused by a defective Feature file. See Appendix C for information on Feature files. If an entered set of files can get past the first curve mapping line in the log, (with max and min line lengths reported), it has a good chance of finishing execution, barring system problems.