Meshmorizer Manual

Chao Wang
chaowang15@gmail.com

This manual is a reference for **Meshmorizer**, a pipeline to generate a simplified mesh model with preserved features. Given an original mesh model (generated by Marching Cube algorithm) in PLY format as input, the pipeline is composed of the following five steps to obtain the final desired simplified mesh model in PLY format.

1 DE-NOISING

The original model is obtained from scanned data with many noises, so it is very necessary to de-noise this model first. This step only involves ***ModelRepair*** code, and contains the following sub-steps:

1.1 CLEAN DUPLICATE POINTS

The first step here is to clean duplicate points and their correlated faces. Duplicate points are such a group of points in the original mesh that some of them share almost the same coordinates with each other.

This step uses ***ModelRepair*** code like this

```
ModelRepair -cleanduplicate input_mesh output_mesh
```

Where `input_mesh` is the original model in PLY format, and `output_mesh` is the output model in PLY format.

1.2 DELETE FLOATING POINTS

The next step is to clean floating points which are always regarded as noisy points in the model. A floating point is such a point that it is not contained in the connectivity of the mesh.

This step uses ***ModelRepair*** code like this

```
ModelRepair -cleanfloating input_mesh output_mesh
```

Where `input_mesh` is the original model in PLY format, and `output_mesh` is the output model in PLY format.

1.3 DELETE SMALL CONNECTED COMPONENTS

Some small connected components can also be considered as noises, although they are contained in the connectivity. This step is to clean these small floating components in the model.

This step uses ***ModelRepair*** code like this

```
ModelRepair -deletecomponents input_mesh vertex_number_threshold output_mesh
```

Where `input_mesh` is the original model in PLY format; `vertex_number_threshold` is a positive integer value denoting a vertex-number threshold, while the connected components in the model with vertex number smaller than this threshold will be deleted; `output_mesh` is the output model in PLY format.

2 Poisson Reconstruction

2.1 Use Poisson reconstruction code to build a globally enclosed model

This step is to use Poisson Reconstruction algorithm to reconstruct the model in order to fill the holes and close the gaps to get a globally enclosed model. The reason is that the original model usually contains a great deal of holes or gaps, especially some large obvious parts such as walls, floors or ceilings. Therefore, it is often necessary to modify the model by filling these apparent gaps first before simplification.

The Poisson Surface Reconstruction (and its advanced algorithm too) is one of the effective reconstruction algorithms in Computer Graphics to reconstruct a point cloud model to obtain a triangular mesh. Here we use the Screened Poisson Surface Reconstruction method [1], an advanced version of original Poisson Reconstruction method. The version 5.5 of source code and usage of Screened Poisson Surface Reconstruction can be obtained from http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version5.5/. (Any version higher than 5.5 is also feasible in this step.)

2.1.1 Generate an input point cloud file from a mesh file

The Poisson Reconstruction code takes **a point cloud model with normal directions** as input. Therefore, before reconstruction, we need to get a point cloud from the mesh model obtained in Step 1. Meanwhile, in order to ensure that the output reconstructed mesh is enclosed, we must create a bounding box for the model. A qualified point cloud model can be created manually in the software ***MeshLab*** under the following instructions:

1) Get a point cloud of input mesh model

- Open the PLY mesh model obtained in Chapter 1 in *MeshLab*;
- Delete all faces and keep all vertices: select all faces and vertices by *Filters/Selection/Select All*, and then delete all faces of input mesh by *Filters/Selection/Delete Selected Faces*;
- Normalize vertex normals by *Filters/Normals, Curvatures and Orientations/Normalize Vertex Normals*. Now we get a point cloud with normalized normal directions;

2) Create a point cloud bounding box of the above input point cloud

- Generate a bounding box for the point cloud. Here we choose to create a bounding sphere as an example by choosing *Filters/Create New Mesh Layer/Sphere* in *Meshlab.* Note that the radius of the sphere must be chosen carefully that the sphere could enclose the whole point cloud totally. (Other kinds of bounding box is also feasible for this, such as a cube).
- Delete all faces and keep all vertices: select all faces and vertices by *Filters/Selection/Select All*, and then delete all faces of the sphere by *Filters/Selection/Delete Selected Faces.*

- Normalize vertex normals by *Filters/Normals, Curvatures and Orientations/Normalize Vertex Normals*. Now we get the bounding box point cloud of input model.

3) Merge the input point cloud and its bounding box point cloud together

- Open the mesh layer panel by clicking the *Show Mesh Layer Dialog* button (i.e. button) in the Standard Toolbox;
- Merge the two point clouds together by right clicking the mouse on the input mesh and choosing *Flatten Visible Layers* (Note: must check *Keep unreferenced vertices* in the new *Flatten Visible Layers* dialog window). Now the two point clouds are merged into one.
- Normalize all vertex normals again by *Filters/Normals, Curvatures and Orientations/Normalize Vertex Normals*.
- Export the point cloud in PLY format. Note that the *Normal* checkbox must be checked during export. Now we get the point cloud of input model together with its bounding box point cloud.

2.1.2 Poisson Surface Reconstruction

The Poisson Surface Reconstruction code ***PoissonRecon*** can be used like this:

```
    PoissonRecon --in input_points --depth reconstruction_depth –out output_mesh
```

Where `input_points` is a point cloud with normal directions in PLY format obtained above; `reconstruction_depth` is the depth value for reconstruction (usually around 10); `output_mesh` is the output model with faces in PLY format. The larger the `reconstruction_depth` value, the denser the points and faces in output mesh.

2.1.3 Delete the bounding box of reconstructed mesh

After reconstruction, we can get a mesh model composed of a reconstructed model with holes and gaps filled, and a bounding box mesh outside. So we need to delete this bounding box outside and keep only the inside mesh. This can be easily dong in ***MeshLab*** like this:

- Open the result PLY mesh model obtained in Poisson reconstruction above;
- Click *Select Connected Components in a Region* button (i.e. button) in the *Edit Toolbox* and then select a tiny small region of the bounding box of the opened mesh by dragging the mouse. Then all the bounding box mesh can be selected, since it is a connected component separated from the inside mesh.
- Delete the bounding box mesh by *Filters/Selection/Delete Selected Faces and Vertices.*
- Export the mesh in PLY format.

Now we get a result constructed mesh with all holes and gaps filled.

2.2 DELETE DIFFERENCE BETWEEN INPUT MESH AND RECONSTRUCTED MESH

The model obtained from Poisson Reconstruction last step is totally enclosed. However, comparing with the mesh obtained in Chapter 1, this model is different and is very likely to contain some large parts we do not need. Therefore, these different parts should be deleted before simplification.

This step uses ***ModelRepair*** code to delete difference between two meshes like this

```
ModelRepair -removedifference original_mesh compared_mesh component_number output_mesh
```

Where `original_mesh` is the reconstructed mesh in PLY format; `compared_mesh` is the original mesh obtained in Step 1; `component_number` is the number of connected components from the difference between the two meshes users want to delete (the deletion order is from large to small); `output_mesh` is the output model in PLY format.

3 SMOOTHING

3.1 MESH SMOOTHING

In this step, a Feature-Preserving Mesh Smoothing method [2] is utilized to smooth the model. The most important advantage of this method is that it is a non-iterative method which can be used to smooth large, noisy and non-manifold meshes efficiently with important features preserved.

3.1.1 Convert a PLY model into a GTS model

The smoothing code in [2] uses a GTS model instead of a PLY model as input, so firstly we need to convert the PLY mesh using ***Plyconverter*** like this:

```
                    Plyconverter –gts input_ply_model output_gts_model
```

Where `input_ply_model` is the input model in PLY format, and `output_gts_model` is the output GTS model.

3.1.2 Feature-Preserving Smoothing

The smoothing code ***Smoother*** in [2] can be used like this

```
      Smoother spatial_weight influence_weight < input_mesh_gts  > output_mesh_gts
```

Where `spatial_weight` and `influence_weight` are two constant predefined parameters by users (a good choice is `spatial_weight` = 4.0, `influence_weight` = 2.0 according to the paper); `input_mesh_gts` is the input mesh in GTS format; `output_mesh_gts` is the output mesh in GTS format. Note that **do not miss** the “<” and “>” marks.

3.1.3 Convert a GTS model back into a PLY model

After obtaining smoothed result model in GTS format, we also need to convert it back into a PLY model. We can use ***MeshLab*** to export a GTS model as a PLY model very easily. Note that it is very likely to get **incorrect** result if we use ***MeshLab*** to convert a PLY model into a GTS model. Therefore, the best way to do this is to use ***Plyconverter*** as described in the above content.

3.2 TRANSFER VERTEX QUALITY FROM ORIGINAL MODEL TO THE CURRENT MODEL

The original model contains vertex quality information, while the current model does not. Then, it is necessary to transfer the information.

This step uses ***ModelRepair*** code to transfer the quality information like this

```
    ModelRepair -transfercolor original_mesh_with_color compared_mesh_without_color
                                output_mesh_with_color
```

Where the `original_mesh_with_color` is the model with vertex quality information; `compared_mesh_without_color` is current model without vertex quality information; `output_mesh_with_color` is the output model in PLY format with transferred vertex qualities.

4 COMPUTE CREST LINES

This step is to compute crest lines, which will be used in mesh simplification next. Crest lines can be regarded as feature lines composed of ridges and ravines. Here we use a fast and robust crest-line-detection method [3], which is based on estimating the curvature tensor and curvature derivatives via local polynomial fitting. The most important advantage of this method is that it is efficient and fast, and the amount of result crest lines can be adjusted by setting thresholds of different parameters defined in this method.

4.1 CONVERT A PLY MODEL INTO A SPECIAL TXT MODEL

The crest-line-detection code in [3] uses a special TXT format file as input, so we firstly need to convert a PLY model into this TXT format. This can be done using ***Plyconverter*** like this:

```
Plyconverter –txt input_ply_model output_txt_model
```

Where `input_ply_model` is a model in PLY format, and `output_txt_model` is the output TXT file.

4.2 DETECT CREST LINES

The crest-line-detection code ***setCurvature*** in [3] can be used like this:

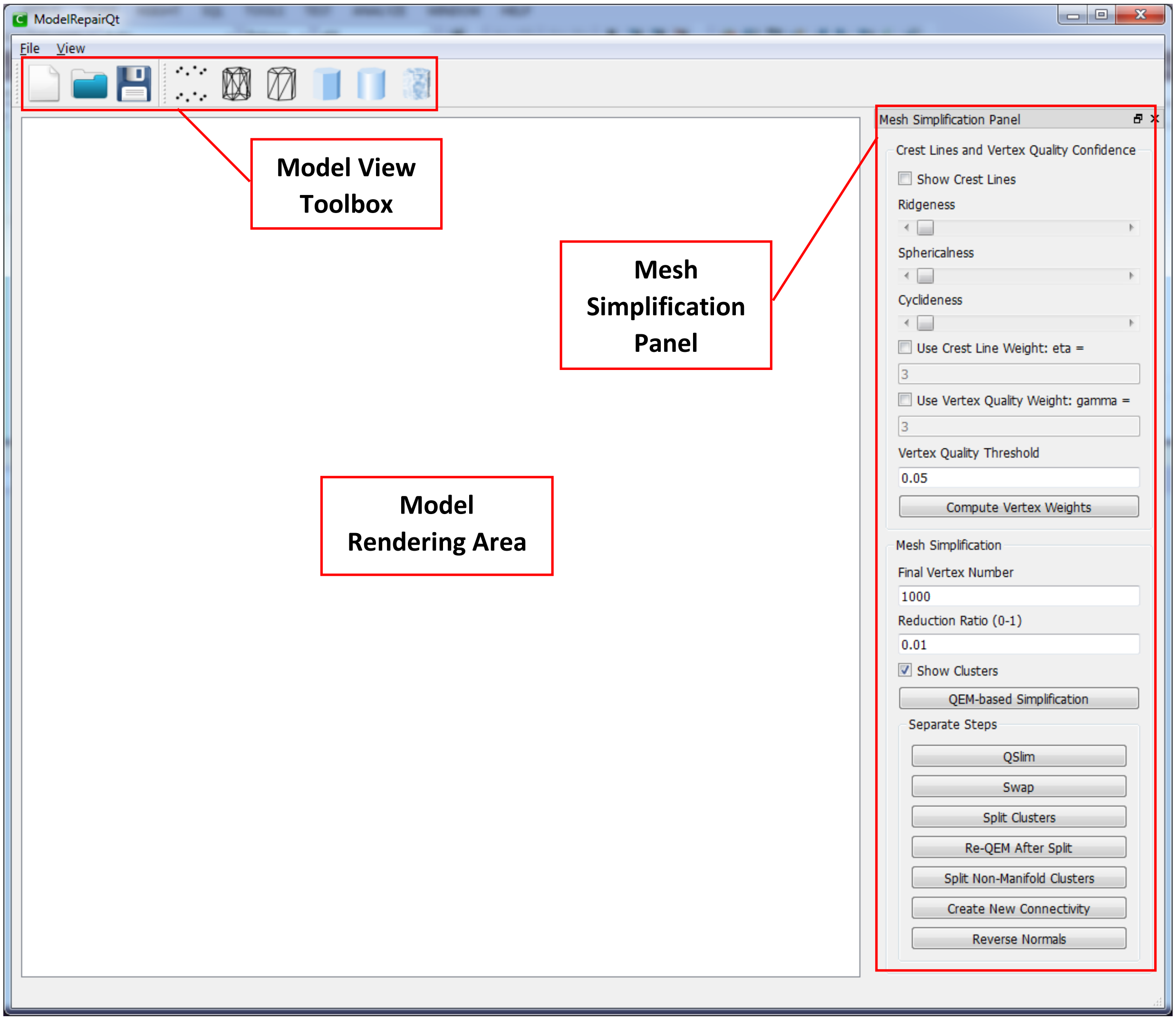
```
setCurvature input_txt_model output_txt_model
```

Where `input_txt_model` is a model in txt format, and `output_txt_model` is the output file containing some information we do not need. This code also generates two files: `ridges.txt` and `ravines.txt` in the same folder. These two files are the crest line files we need in the mesh simplification step next. Note that in order to render the crest lines, we must change the names of the two files `ridges.txt` and `ravines.txt` manually into `XXX.ridges` and `XXX.ravines`, where `XXX` is the name of PLY model.

5 MESH SIMPLIFICATION

The idea of this mesh simplification step is mainly based on the Quadric Error Metric (QEM) Simplification method [4] and one of QEM's advanced method --- user-guided simplification [5]. QEM [4] is a very classic, simple and efficient mesh simplification algorithm and has been used very widely in computer graphics and other related areas. The most important advantage of QEM is that it is a very fact algorithm with simple principle. The advanced version of QEM in [5] is a simplification method to allow users to selectively control the importance of different surface regions and preserve various features through the imposition of geometric constraints. The main idea in [5] is used in our code ***ModelRepairQt*** by utilizing the crest line results as important regions to be preserved in QEM.

This step uses the ***ModelRepairQt*** code, which utilizes some other libraries including Qt 5.0 (or higher) and Eigen library. The interface of ***ModelRepairQt*** code in Windows 7 is like this:



Crest Lines and Vertex Quality Confidence

☐ Show Crest Lines

Ridgeness

Sphericalness

Cycldeness

☐ Use Crest Line Weight: eta = 3

☐ Use Vertex Quality Weight: gamma = 3

Vertex Quality Threshold: 0.05

Mesh Simplification

Final Vertex Number: 1000

Reduction Ratio (0-1): 0.01

☑ Show Clusters

5.1 View Crest Lines

To view crest line features in ***ModelRepairQt*,** follow these instructions in order:

- Put the two crest line files obtained in Section 4.2, i.e. `ridges.txt` and `ravines.txt`, in the same folder with the ply model;
- Change the names of the two crest-line files into `XXX.ridges` and `XXX.ravines` respectively, where `XXX` is the name of PLY model;
- Open model in ***ModelRepairQt*** by clicking the *Open Model* button (i.e. 📂 button) in the *Model View Toolbox*;
- Check the *Show Crest Lines* checkbox in the *Mesh Simplification Panel* to show the crest lines. Initially all crest lines (i.e. all ridges and ravines) will be shown by default.

- Use the three threshold parameters (the three scrollbars under the checkbox Show Crest Lines) to filter the crest lines to be viewed and used in simplification (the descriptions come from the Readme.txt file of the original crest-line-detection source code):
  - Ridgeness ($0 \le T_0 \le 10$): changing ridgeness which is integration of the curvature along the connected crest lines.
  - Sphericalness ($0 \le T_1 \le 10$): changing sphericalness which is average of the window functioned Shapeindex for the connected crest lines (0 < s <1).
  - Cyclideness ($0 \le T_2 \le 10$): changing cyclideness which is integration of the MVS (minimum variation surface) functional along the connected crest lines.

The three threshold parameters are used to filter crest lines, since there are a great deal of noisy crest lines we do not want to use in simplification. The larger each parameter, the less crest lines to be viewed. According to the experiences of this manual's author in most experiments, ridgeness T_0 is used for precision tuning, and sphericalness T_1 is used for coarse tuning, while cyclideness T_2 is not used at all. For instance, two groups of parameters $T_0 = 10.0, T_1 = 0.0, T_2 = 0.0$ and $T_0 = 9.1, T_1 = 0.6, T_2 = 0.0$ are both used in simplifying the *convenience store* model in the author's experiments, and both results are satisfying. The value of the corresponding parameters are prompted in the console window when users change a scrollbar.

Users can refer to the crest-line-detection paper [3] for more detailed information about the crest line information and the relative threshold parameters.

5.2 MESH SIMPLIFICATION

The entire mesh simplification step contains two sub-steps: first round simplification step and second round simplification step. The reason is that sometimes the result after simplifying the input model once still contains redundant vertices and faces. Then, we have to simplify the result model again in the "second round" to get a satisfying result.

5.2.1 First round mesh simplification

This step is to simply the input mesh model obtained in Chapter 4 to get a simplified mesh with important features preserved. The first round mesh simplification step contains the following sub-steps:

1) Compute weight distribution of all vertices

- Open the mesh model;
- Check the *Show Crest Lines* checkbox in the *Mesh Simplification Panel* to show the crest lines, and drag the three scrollbars underneath to view the crest lines to be used in the following steps;
- Check the *Use Crest Line Weight: eta=* checkbox to put the *eta* value and the *Use Vertex Quality Weight: gamma=* checkbox to put the *gamma* value. These two parameters, *eta* and *gamma*, are used to control the relative influence of crest line weight and vertex quality weight, respectively;
- Put a *Vertex Quality Threshold* value, which is a percentage value in range 0 to 1. The part of vertices with quality value below this percentage value will be used;

- Compute the weight distribution for all vertices by clicking the *Compute Vertex Weights* button. Now we can get a rendering result of all vertices with attached weights. The blue points denote small weights, the yellow ones denote middle weights, and the red ones denote large weights.

2) Run mesh simplification

- Put *Final Vertex Number* or *Reduction Ratio* value. Here the Reduction Ratio value is the final percentage of vertex number to the original vertex number in the simplified mesh users desire to get. If the *Reduction Ratio* is 0, then the *Final Vertex Number* value will be used; otherwise, the *Reduction Ration* value will be used in the simplification step;
- Run the entire simplification step by clicking the *QEM-based Simplification*;
- Export the result mesh by clicking the *Save* button (💾 button) in the *Model View Toolbox;*

Now we get the "first round" simplified model. If this model is already satisfying to the user, it can be regarded as the final model and the whole Meshmorizer pipeline is over now. If not, we have to continue to the "second round mesh simplification" step.

Some important things should be noted about this step:

- The result mesh of this step is guaranteed to be manifold, even though it may contains some small triangular holes;
- It may take a long time to get the final result under a very large input model. User can refer to the prompt information in the console window to see the current progress in the simplification step;
- The buttons below the *QEM-based Simplification* button in the *Mesh Simplification Panel* are separate steps of the mesh simplification process. User can click these buttons from top to bottom one by one except the last button *Reverse Normals*, which is used to reverse all normal directions of output mesh in case they are inverse).

5.2.2 Second round mesh simplification

This step is to simply the model on the result model obtained above for the second time. In this step, only QEM is used in ***ModelRepairQt*** to simplify the model:

- Open the mesh model obtained in Section 5.2.1;
- Put *Final Vertex Number* or *Reduction Ratio* value. Here the Reduction Ratio value is the final percentage of vertex number to the original vertex number in the simplified mesh users desire to get. If the *Reduction Ratio* is 0, then the *Final Vertex Number* value will be used; otherwise, the *Reduction Ration* value will be used in the simplification step;
- Run only QEM method by clicking the *QSlim* button;
- Export the result mesh by clicking the *Save* button (i.e. 💾 button) in the *Model View Toolbox.*

Now we get the final simplified result model and the total Meshmorizer pipeline is over.

REFERENCE

[1] Kazhdan M, Hoppe H. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)*, 2013, 32(3): 29.

[2] Jones T R, Durand F, Desbrun M. Non-iterative, feature-preserving mesh smoothing. *ACM Transactions on Graphics (TOG)*. ACM, 2003, 22(3): 943-949.

[3] Yoshizawa S, Belyaev A, Seidel H P. Fast and robust detection of crest lines on meshes. *Proceedings of the 2005 ACM symposium on Solid and physical modeling*. ACM, 2005: 227-232.

[4] Garland M, Zhou Y. Quadric-based simplification in any dimension. *ACM Transactions on Graphics (TOG)*, 2005, 24(2): 209-239.

[5] Kho Y, Garland M. User-guided simplification. *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM, 2003: 123-126.