CMSC 427 Computer Graphics
Programming Assignment 3: Geometry Processing
Due Date: 11:59pm, April 1, 2017 ~~11:59 PM, Mar. 30, 2017~~
Project Submission:

> **1) Delete all intermediate files (run the command make distclean) and the Makefile created during the compilation process. Also delete the executable file.**
> **2) Place all files for the project in a folder and ZIP up the folder. You will submit your project via the submit server. To submit a zip file, login to the submit server webpage and look for the link to make a *web submission*.**

Project Description
In this assignment, you must choose operations from the following list to implement. These operations have been divided into groups. Within each group, you may choose what operations to implement. At a minimum, you must implement the required number of operations per group. You will also need to submit a write-up demonstrating your implementation of each operation with screenshots.

- Analysis (**required – implement all operations**)
  - Compute average edge lengths - Compute the average length of edges attached to each vertex. This feature should be implemented first. To do it, you must design a data structure that allows O(K) access to edges attached to each vertex, where K is the number of edges attached to a vertex. Note that this is used in the warps section below.
  - Compute *per-vertex* normal - Compute the surface normal at a vertex. This feature should be implemented second. To do it, you must design a data structure that allows O(K) access to faces attached to each vertex, where K is the number of faces attached to a vertex. To compute the normal for a vertex, take a weighted average of the normals for the attached faces, where the weights are determined by the areas of the faces. Store the resulting normal in the "normal" variable associated with the vertex. Note also this is used in the warps section below.
- Warps (**required – implement all operations**)
  - Inflate - Move every vertex along its normal direction. The input parameter *factor* should be multiplied by the average length of the edges attached to the vertex to determine the displacement of each vertex along its normal direction. Note that the *factor* can be negative, which means that the vertex should move in the direction opposite to the normal vector.
  - Random noise - Add noise of a random amount and random direction to the position of every vertex, where the input parameter *factor* should be multiplied by the average length of the edges attached to the vertex to determine its maximum displacement (i.e., displacement distances should be between 0 and factor * vertex->AverageEdgeLength().
- Filters (**required – implement 2 operations**)
  - Smooth – Smooth the mesh by moving every vertex to a position determined by a weighted average of itself and its immediate neighbors (with weights determined by a Gaussian with sigma equal to the average length of edges attached to the vertex, normalized such that the weights sum to one).
  - Sharpen – Accentuate details in the mesh by moving every vertex away from the position determined by a weighted average of its neighbors (with weights determined by a Gaussian function of distance). This filter moves vertices in the direction opposite from smoothing.
  - Center vertices tangentially – Smooth a mesh vertex by using the average of its immediate neighbors. This is also called Laplacian mesh smoothing. Describe in your writeup how this differs from a Gaussian weighted smoothing operation.
- Remeshing (**required – implement 2 operations**)

    - Split faces - Split every triangle face into 4 triangles. Create a new vertex at the midpoint of every edge, remove the original face, create a new face connecting all the new vertices, and create new triangular faces connecting each vertex of the original face with the new vertices associated with its adjacent edges.
    - Split long edges - Iteratively split edges longer 4/3rds the average edge length (across all edges). Note that every edge split produces a new vertex at the edge midpoint and replaces the two adjacent faces with four. Edges should be split repeatedly until there is none longer than the given threshold. Note that after an edge split, the new edges might still be too long, so you might need to split these new edges again. Note: extra points will be given if longer edges are split first (which produces better shaped faces). If you do this, note it in your writeup.
    - Collapse short edges - Iteratively collapse edges shorter than 4/5th the average edge length (across all edges). Note that every edge collapse merges two vertices into one and removes up to two faces (if the adjacent faces are triangles). Place the new vertex at the midpoint of the collapsed edge.
- Subdivision (**required – implement all operations**)
    - Loop subdivision – Subdivide every face using the same method as Split Faces (see above). Then update the positions of all vertices according to the Loop subdivision weights. This only must work correctly for meshes with triangular faces.

Extra credit – up to 20% extra credit, awarded based on difficulty

- Flip Edges – For a fixed number of iterations (your choice), for all vertices with >6 edges attached (degree > 6), flip one of the neighboring edges.
- Truncate – For every vertex, create a new vertex a parameter t [0-1] of the way along each of its attached edges, and then "chop off" the pyramid whose base is formed by the new vertices and whose apex is the original vertex, creating a set of faces to triangulate the hole.
- Crop - Crop the input mesh to the positive side of the plane. This feature requires clipping each polygon crossing the plane and discarding any part of any face on the negative side of the plane.
- Bilateral smoothing: Smooth the mesh using a bilateral filter as in http://people.csail.mit.edu/thouis/JDD03.pdf
- Implement mesh simplification using quadrics, as used in progressive meshes. Provide a writeup detailing how you used quadrics http://research.microsoft.com/en-us/um/people/hoppe/proj/pm/
- Implement any of the boolean operations by rasterizing the meshes onto two voxel grids, then using max(), min(), or subtraction elementwise to compute the boolean operation, then extract the isosurface to produce the final mesh.

Code Requirements

You may add helper methods to the code, but you are prohibited from changing the options in the popup context menu (accessed by right-clicking the mesh model).

Your program must be executable from the command line using qmake and make.

Project Grading

There are 2 parts that will be graded. 1) The write-up and 2) your code. The write-up should include screenshots demonstrating each operation you chose to implement. There should be 2 screenshots per operation – one before and one after applying the operation to the mesh. Screenshots are required for all operations except the *analysis* group of operations. The TA will use an undisclosed set of meshes to test your implementation as well so we encourage you to find more meshes online to test your program. We have provided you with several meshes to start off with.

Your code should be well commented in header files and source files. Note that it will also be judged subjectively based on the simplicity and clarity of implementation. An implementation that is easy to understand, but has few minor bugs will be scored higher than a messy implementation with the same number of minor bugs.

Documentation Grading

Please include documentation in the form of a Markdown (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet) document with your assignment. It should be a text file in the folder ./cmsc427 called DOCUMENTATION.md. You will be graded on the completeness, clarity, and conciseness of your documentation. Your documentation should links to inline images showing the before and after of your image processing operations. It should also include an explanation of the algorithm you implemented. The more complete and clear your explanation is in this documentation the easier it will be for us to assign partial credit. It is in your interest to be as concise and clear as possible. Please make sure you only use relative links in your mark down file. If the TA has difficulty finding your images and the links, you will lose points. It is additionally suggested that you convert your Markdown file to .PDF so the TA.

Additional Submission Details

- Exactly one person of your team should be responsible for the submission. The latest submission will be the one your team be graded on.
- Your document should be properly written with markdown notations, and with correct extension names: it should be either .md or .markdown, not .txt. (If you prefer an extended version of Markdown (e.g. GitHub flavored Markdown or Pandoc's Markdown), it is also fine if you have included your PDF file.)
- It's recommended (required if you are using extended version of Markdown) to generate a PDF file for your markdown document. and put it under the same directory with same basename (i.e. DOCUMENTATION.pdf), take a look at it to make sure the layout is intended and screenshots are properly included. If the PDF file is missing, your document will be converted from your DOCUMENTATION.md or DOCUMENTATION.markdown to a PDF file using pandoc before grading.
- Most online markdown editors would have the option to export your document as PDF files, and there are tools that do this Markdown-PDF conversion online.
- It's not recommended to include your images as website URLs or encode your images into URLs. and do not use local or private URLs otherwise I can't find your screenshots. You should include your images as separated files in your submission.
- Make sure to crop your screenshot properly, it should be enough to show your work, but not too much information (e.g. another screen with youtube or twitch opened).
- In your submission there should be just project files, screenshots, documents and other things as requested. Top-level zip files are fine, but do not add any other level of archive in your submission, which means things like *.zip, *.7z are not recommended, compressing data more than once barely gives you a smaller file anyways.
- For your project directory, it's recommended to use just a combination of alphabetic and numeric characters and underlines (or in other words, your project directory name should be an ASCII string accepted by regular expression \w+), otherwise qmake would have some trouble with project path.
- Your source code should be compatible with C++11 standard and cross-platform:
  - one way to make your compiler keep an eye on this is to ensure c++11 appears in the *.pro file of your project in CONFIG += ... line (which it should have been if you download the latest starter project). also try to address compiler warnings instead of suppressing them.
  - be consistent with file names, which means if you name one of your header Foo.h, you should write your include directive like #include "Foo.h", not #include "foo.h" or #include "FOO.H"