Project Submission:
1) Delete all intermediate files (run the command `make clean`) 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 project, 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)
  o 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.
  o 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)
  o Inflates - 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.
  o 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 1 operation)
  o 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).
  o 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.
  o 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.

• Remeshing (required – implement 2 operations)
- Split faces - Split every face into K+1 faces (where K is the number of vertices on the face). 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.
- Star faces - Split every face into N triangles (where N is the number of vertices on the face). That is, create a new vertex at the centroid of the face, remove the original face, create N new triangular faces connecting the new vertex with each pair of adjacent vertices of the original face. Position the new vertex at a point that is offset from the centroid of the face along the normal vector by a distance equal to factor times the average edge length for the face.
- Split long edges - Iteratively split edges longer than max_edge_length. 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: 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 min_edge_length. 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. Note: extra points will be given if shorter edges are collapsed first (which produces better shaped faces). If you do this, note it in your writeup.
- Cluster vertices - Simplify the mesh by clustering vertices residing in the same cell of a grid defined by x, y, and z spacing parameters. All vertices within the same grid cell should be merged into a single vertex. That vertex should be then placed at the centroid of the cluster vertices, and all edges and faces that collapse as a result of the vertex merging should be removed.

**Subdivision Surfaces (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 10% extra credit, awarded based on difficulty**
- 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.