CSE 5443 Lab 6
(100 points)
Autumn, 2021

Write and submit (on carmen) a program to maximize the minimum angle of any mesh triangle using decimation operations collapse edge, split cell, split edge and join cell. Make sure your program preserves mesh topology by checking that operations will not change the mesh to a non-manifold.

Program details:

1. Name the program `Qmesh` (for quality mesh.)
2. Run the program as:
   `Qmesh <input .off file>`
   or
   `python Qmesh.py <input .off file>`.
   or
   `java Qmesh.Qmesh <input .off file>`
   (Name the package for your java program, `Qmesh`.)
3. Output is written to the file `out.off`.
4. You are free to add other options to control the running of your program, but your program will be tested with the default values (no options) for all control parameters.
5. Your program SHOULD NOT be interactive. One should be able to run/test your program from a single command line without entering any additional values.
6. Write the program in C++, Python or Java.
   (a) Extensions to the half edge mesh data structures that support decimation are provided in each language. (More information below and in a separate file, `hmesh_decimation.pdf`.)
   (b) The provided software includes routines to report minimum and maximum edge lengths and minimum and maximum angles in each mesh cell, and to collapse cell edges, split cells with a diagonal, split mesh edges at a midpoint and join two cells by removing a mesh edge.
7. Input to your program are meshes created using the Marching Cubes or dual contouring algorithms.
   (a) The meshes are guaranteed to be oriented manifolds.
   (b) Meshes MAY NOT be composed only of triangles. The meshes may have quadrilaterals or even possibly pentagons. Use the split cell operations to triangulate the cells with four or more vertices.
   (c) You may assume that each mesh cell has at most six vertices. (If you apply the join cell operation, then, of course, your mesh cells may have more vertices.)

8. Programs `meshinfo` is provided in C++, Python and Java, to return mesh information, including minimum angle,
9. You are free to combine the following steps in any order you choose to produce meshes with small angles:
   (a) Compute the length L of the longest mesh edge and collapse any edges whose edge length is less than R*L for some fraction R. (Choose R based on experimentation. Use a ratio R instead of an absolute measurement so that your algorithm is scale invariant.)
   (b) For each mesh cell, compute the ratio of the shortest to longest mesh edge lengths and collapse the shortest edge length if the ratio is less than some R'. (Choose R' based on experimentation. Use a ratio R' instead of an absolute measurement so that your algorithm is scale invariant.)
   (c) For each mesh cell, compute the ratio of the shortest to longest mesh edge lengths and split the longest edge if the ratio is greater than some R". (Choose R" based on experimentation. Use a ratio R" instead of an absolute measurement so that your algorithm is scale invariant.)
   (d) For each mesh TRIANGLE with a very large angle A, delete the longest mesh edge (the one opposite A,) joining the triangle to the cell on the opposite side.
   (e) For each mesh TRIANGLE with a very large angle A', split the longest triangle edge.
   (f) For each mesh TRIANGLE with a very small angle A" but no very large angles, collapse the shortest mesh edge (the one opposite A".)
   (g) Split every mesh cell with more than three vertices until the entire mesh is a triangle.
      i. Simplest is to split the mesh cell from the cell vertex with maximum angle.
      ii. Better is to compute the angles created by each split before the split, and choose the split that maximizes the minimum angle.
      iii. A function `ComputeCosTriangleAngle(v0, v1, v2)` is provided to compute the cosine of angle `(v0, v1, v2)`.

   You can combine the steps above in any order, and/or repeat steps using different parameters, or come up with your own determination of when to collapse, join or split edges or mesh cells. You can also do cominations of operations, e.g. split two long mesh edges and then split the mesh cells with a diagonal connecting the two new vertices.

   (a) Do not resample the mesh, or use some global optimization to compute "best" operations. The idea is to use simple, local decimation operations to improve mesh quality.
   (b) Do not do any sorting (priority queue) of minimum/maximum cell angles, edge lengths, etc. The idea is to keep all operations local so that they potentially could be easily implemented in parallel.

10. When your program has completed the decimation, it should print out the following information:
    (a) Number of mesh vertices.
    (b) Number of mesh edges.
    (c) Number of boundary mesh edges.
    (d) Number of mesh cells.
    (e) Minimum mesh edge length.
    (f) Maximum mesh edge length.
    (g) Minimum cell angle.
    (h) Maximum cell angle.

    The class `HalfEdgeMeshDCMT` or `HALF_EDGE_MESH_DCMT` has member functions (methods) that provided that information. See the source code for `meshinfo` for examples of calling those routines.
11. When your program has completed the decimation, call `CheckAll()`, `CheckManifold()`, `CheckOrientation()`. Print a warning message if any of those checks fail.
    - In the submitted version of your program, DO NOT make these calls after performing every decimation operation (collapse/split/join). These calls take time proportional to the entire mesh size. On a mesh with n cells, you can expect to perform $\Theta(n)$ decimation operations. Calling the check functions after each operation would make your program take $\Theta(n^2)$ time.
12. A sample program `decimate_mesh` (or `DecimateMesh`) is provided to give some simple examples of applying the mesh decimation operators.
13. Test your program on .off files containing meshes provided in `Carmen->Files->OFF files->Lab6`.

Submission Instructions:

Submit a .zip file containing:

1. All source code for your program including the files provided for this lab;
2. A file `Qmesh Algorithm` (text, Word or pdf,) outlining your algorithm.
    - List the steps you applied in your program in the order you applied them and give the default parameters (length ratios/angles) used at each step.
    - Describe any steps that you devised.
3. A README file showing how to run your program.

Half edge mesh decimate (DCMT) code:

1. The classes to support half edge mesh decimation are provided in the same repositories as before:
    - C++: `https://github.com/rafewenger/halfEdgeMesh-cpp.git`
    - Python: `https://github.com/rafewenger/halfEdgeMesh-Python.git`
    - Java: `https://github.com/rafewenger/halfEdgeMesh-Java.git`
2. NOTE: There have been UPDATES/ADDITIONS to the halfEdgeMeshBase (`HALF_EDG_MESH_BASE`) and related classes provided in lab 5.
    - Be sure to download/use the updated versions of the classes from lab 5. (They have been updated in the repositories.)
    - All the updates should be backwards compatible, with your lab 5 code still compiling and running as before. (Contact me if there is an issue.)
3. Code documentation is again at:
    - C++:
`http://web.cse.ohio-state.edu/~wenger.4/CSE5543/doc/halfEdgeMesh-Cpp`
    - Python:
`http://web.cse.ohio-state.edu/~wenger.4/CSE5543/doc/halfEdgeMesh-Python`
    - Java:
`http://web.cse.ohio-state.edu/~wenger.4/CSE5543/doc/halfEdgeMesh-Java`
4. As before, the data structures to support decimation in each language are pretty similar, with a base HalfEdgeMeshDCMTBase (or `HALF_EDGE_MESH_BASE_DCMT`) class representing the mesh, and classes VertexDCMTBase, HalfEdgeDCMTBase, CellDCMTBase (or `VERTEX_DCMT_BASE`, `HALF_EDGE_DCMT_BASE`, `CELL_DCMT_BASE`,) representing the vertices, half edges and cells supporting decimation operations. (`DCMT` stands for decimate.)
    (a) DO NOT (try not) modify the classes/files provided. Create new files that include/import the files provided, and new classes that are derived from the orignal classes. (If you do modify the classes, be sure to comment/document your modification. It is very likely that there may need to be some minor correction to these classes, and you will have to download and modify the class again.)
    (b) In C++, the classes are C++ template classes. Classes `*_DCMT_A`+ are provided as simple examples of instantiations of these templates. Create your own derived classes as needed. (Don't modify the `*_DCMT_A` classes.)
    (c) In python, the provided classes are python classes. You can use the provided classes directly or create classes derived from these classes. Do not modify the classes or the provided files. Pass the derived classes to `HALF_EDGE_MESH_DCMT_BASE` in creating a new object of type `HALF_EDGE_MESH_DCMT_BASE` and mesh will create objects with the derived classes whenever it needs to create a vertex, half edge or cell.

(d) In Java, the provided class are abstract classes. Concrete versions of these classes are provided in `*DCMTA.java`. Create your own subclasses as needed. You will need to write your own factory derived from `HalfEdgeMeshFactoryBase.java` for creating these subclass objects. See `HalfEdgeMeshDCMTFactoryA.java`, for an example of a factory creating objects of type `*DCMTA`.

5. Sample programs `decimate_mesh` (or `DecimateMesh`) are provided in each language to give some simple examples of applying the mesh decimation operators.