CSE 5443 Lab 5b
(90 points)
Autumn, 2021

Write and submit (on carmen) a PROGRAM that constructs a Catmull-Clark subdivision. Your program should read the mesh from a .off file and write the Catmull-Clark subdivision to a (different) .off file.

Program details:

1. Name the program `CCsubdivide`.
2. Run the program as:
   `CCsubdivide [-num_iter <n>] <input .off file>`
   or
   `python CCsubdivide.py [-num_iter <n>] <input .off file>`.
   or
   `java CatmullClark.CCsubdivide [-num_iter <n>] <input .off file>`
   (Name the package for your java program, `CatmullClark`.)
3. Output is written to the file `out.off`.
4. The option "`-num_iter <n>`" determines the number of iterations of the Catmull-Clark subdivision. `<n>` should be 1 or more. Your program should apply the Catmull-Clark subdivision `<n>` times to the mesh, each time creating a more refined Catmull-Clark subdivision. When the option "`-num_iter <n>`" is omitted, apply the Catmull-Clark subdivision just once.
5. You are free to add other options.
6. Write the program in C++, Python or Java.
   (a) Half edge mesh data structures are provided in each language. (More information below.) This includes routines to read and write the half edge mesh data structure from and to .off files.
   (b) You are free to write your own half edge mesh data structure, but note that writing such a data structure can be time consuming and tricky. Furthermore, I will be adding more functionality to these data structures which split or join cells and implementing such operations is even tricker.
   (c) You can also use half mesh data structures that you find on-line, but you will need to submit all the code that goes with that data structure. You will also need to write routines to read and write the mesh from and to .off files.
7. Your program should do the following:
   (a) Read the mesh into a half edge mesh data structure.
      - In the input mesh, mesh cells (polygons) could have 3 edges (triangles), 4 edges (quadrilaterals), 5 edges (pentagons), or even more edges.
   (b) Create a new half edge mesh data structure and add the vertices and cells of a single Catmull-Clark subdivision to this data structure.

- Create the Catmull-Clark subdivision as described in Section 12.17 of Agoston's text, Computer Graphics and Geometric Modelling. Be sure to locate the vertices at the vertex positions as described in that text. DO NOT simply subdivide the mesh without moving the vertex positions.
- DO NOT attempt to modify the original half edge mesh data structure using "cell subdivision" operations. Because Catmull-Clark subdvides all the edges of the mesh, trying to modify the data structure is problematic. Instead use the old data structure to create a new one.

(c) If the number of iterations is greater than 1, create a new half edge mesh data structure for each iteration, using the previous one as input.

(d) Write the mesh to a .off file.

8. Sample input .off files containing meshes are in `Carmen->Files->OFF Files->Lab5`.

Half edge mesh code:

1. Half edge mesh code is provided in C++, Python and Java. The code can is in GitHub repositories:

   - C++: `https://github.com/rafewenger/halfEdgeMesh-cpp.git`
   - Python: `https://github.com/rafewenger/halfEdgeMesh-Python.git`
   - Java: `https://github.com/rafewenger/halfEdgeMesh-Java.git`

   Code documentation is 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`

2. The data structure in each language is pretty similar, with a base HalfEdgeMesh-Base (or `HALF_EDGE_MESH_BASE`) class representing the mesh, and classes VertexBase, HalfEdgeBase, CellBase (or `VERTEX_BASE`, `HALF_EDGE_BASE`, `CELL_BASE`,) representing the vertices, half edges and cells in the mesh.

3. Each provided class is meant to be a base class for subclasses created by you that may contain more fields/information.

   - 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.)

- In C++, the classes are C++ template classes. Classes `VERTEX3D_A`, `HALF_EDGE_A`, `CELL_A`, and `HALF_EDGE_MESH_A` are provided as simple examples of instantiations of these templates. Create your own derived classes as needed. (Don't modify the `*.A` classes.)
- 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 files `half_edge_mesh.py` and `half_edge_mesh_IO.py`. Pass the derived classes to `HALF_EDGE_MESH_BASE` in creating a new object of type `HALF_EDGE_MESH_BASE` and mesh will create objects with the derived classes whenever it needs to create a vertex, half edge or cell.
- In Java, the provided class are abstract classes. Concrete versions of these classes are provided in VertexA.java, HalfEdgeA.java, CellA.java, HalfEdgeMeshBaseA.java. The java implementation also needs a "factory", HalfEdgeMeshFactoryBase.java, to create the vertex, half edge, and cell objects in HalfEdgeMeshBase.java. Create your own subclasses as needed. You will need to write your own factory derived from HalfEdgeMeshBase.java for creating these subclass objects. See HalfEdgeMeshFactoryA.java, for an example of a factory creating objects of type VertexA, HalfEdgeA, CellA.

4. Code also comes with a test program that tests the data structure and I/O routines by reading in a .off file to the mesh, running check mesh, check manifold and check orientation routines on the mesh, and then writing the mesh to a .off file.

5. Descriptions of the methods (member functions) associated with each class are in the instructions for lab5a and in the on-line code documentation.

   - The only major difference between the implementations is in the data structure storing references to the vertices, half edges and cells in the mesh.
   - The C++ implementation uses arrays `vertex_list`, `half_edge_list` and `cell_list` to store the references, where `vertex_list(iv)` is a pointer to the vertex with index `iv`, where `half_edge_list(ihalf_edge)` is a pointer to the half edge with index `ihalf_edge`, and `half_edge_list(icell)` is a pointer to the cell with index `icell`.
   - To process all the vertices in a mesh in C++, use:

```
for (int iv = 0; iv < mesh.VertexListLength(); iv++) {
    const VERTEX_TYPE * v = mesh.Vertex(iv);
    if (v != NULL) {
        // Process vertex v.
        // If v == NULL, then there is no vertex with identifier
        //     iv in the mesh.
```

   Similarly, to process all the cells in a mesh in C++, use:

```
for (int icell = 0; icell < mesh.CellListLength(); icell++) {
    const CELL_TYPE * cell = mesh.Cell(icell);
    if (cell != NULL) {
        // Process cell.
        // If cell == NULL, then there is no cell with identifier
        //     icell in the mesh.
```

`VERTEX_TYPE` and `CELL_TYPE` are the types of your vertex and cell classes. Similar code should be used to process all the half edges.

- The implementations in Python and Java use hash tables (hash maps/dictionaries), `vertex_hashtable`, `half_edge_hashtable`, and `cell_hashtable` to store the references. Keys are the vertex, half edge and cell indices. `Vertex(iv)` returns the vertex with index `iv`, `HalfEdge(ihalf_edge)` returns the vertex with index `half_edge`, `Cell(icell)` returns the vertex with index `icell`.
- To process all the vertices, half edges or cells in a mesh in Python, use:

```
for iv in mesh.VertexIndices():
    v = mesh.Vertex(iv)

for ihalf_edge in mesh.HalfEdgeIndices():
    half_edge = mesh.HalfEdge(ihalf_edge)

for icell in mesh.CellIndices():
    cell = mesh.Cell(icell)
```

- To process all the vertices, half edges or cells in a mesh in Java, use:

```
for (Integer iv: mesh.VertexIndices()) {
    VERTEX_TYPE v = mesh.Vertex(iv);

for (Integer ihalf_edge: mesh.HalfEdgeIndices()) {
    HALF_EDGE_TYPE half_edge = mesh.HalfEdge(ihalf_edge);

for (Integer icell: mesh.CellIndices()) {
    CELL_TYPE cell = mesh.Cell(icell);
```

`VERTEX_TYPE`, `HALF_EDGE_TYPE`, `CELL_TYPE` are your vertex, half edge and cell classes.