Assignment 2: Mesh Processing

In this assignment, you will implement a data structure to represent a triangle mesh, and use it to perform some basic mesh processing tasks. We have provided starter code at https://git.iitd.ac.in/col781-2202/a2 to interactively visualize the mesh in 3D, given its vertex data and triangle indices. Your responsibility is to work with the mesh data structure itself to create and modify meshes.

Part 1

1. Define a mesh data structure to store the connectivity and geometry of a triangle mesh.

    - The connectivity representation should allow efficient access to the neighbouring elements of each vertex and of each triangle.
    - Meshes with boundaries should be supported. Even for a boundary vertex, it should be straightforward to traverse all its neighbouring triangles.
    - Both positions and normals of all vertices should be stored.
    - You should also implement functionality to send your mesh to the rasterization API for rendering, like in Assignment 1.

    Further implementation choices beyond this are up to you, e.g. whether to use half-edges or triangle neighbours (with or without edges), whether to use indices or pointers, etc.

2. Implement functions to create the following simple meshes, including vertex normals and all the connectivity information:

    - A unit square in the *xy*-plane, divided into a grid of m rows and n columns. Each grid cell will be a $1/m \times 1/n$ rectangle divided into two triangles, for a total of $(m+1)(n+1)$ vertices and $2mn$ triangles.
    - A unit sphere discretized into m “slices” (longitudes) and n “stacks” (latitudes) along the z-axis. For example, $m = 4$ and $n = 2$ should give an octahedron. Look up spherical coordinates if you’re not sure how to set the vertex positions. Be careful not to create duplicate vertices at the poles!

3. In the `meshes` directory of the starter code, we have provided a few example meshes in the OBJ file format, a fairly self-explanatory text-based format for polygon meshes. Write a parser that loads a mesh from such a file. You may assume that all the polygons are triangles. Be careful about a few things:

- Indexing in the OBJ format starts from 1, not 0.
- The file only contains vertex indices for the polygons, so you will have to build the rest of the connectivity information yourself while loading.
- The file may or may not contain vertex normals. If it does not, set the normals to zero while loading; you will fix them in the next part.

4. Implement functionality to recompute vertex normals for a given mesh. For each vertex, we can estimate the surface normal as a weighted average of the normals of adjacent triangles. You may use equal weights, or weights proportional to triangle area, though the best choice is probably the scheme derived by Nelson Max (1999) (see eq. 2) which gives exactly correct normals for vertices lying on a sphere.

Part 2

1. The simplest way to smooth out a mesh is using the so-called "umbrella operator" $\Delta \mathbf{v}_i = \frac{1}{|N(i)|} \sum_{j \in N(i)} (\mathbf{v}_j - \mathbf{v}_i)$, which for each vertex gives the vector from its position to the average position of its neighbours. Moving each vertex by some fraction in this direction, $\mathbf{v}_i \leftarrow \mathbf{v}_i + \lambda \Delta \mathbf{v}_i$ causes a small amount of smoothing over the mesh; repeating it multiple times leads to more and more smoothing.
   - Implement a function that does performs this process, taking as parameters the factor λ and the number of iterations.
   - The problem with naïve smoothing is that repeated iterations also cause the mesh to shrink without bound. (Unlike subdivision surfaces, repeated smoothing will eventually cause the mesh to shrink to a point.) Gabriel Taubin proposed a simple way to prevent shrinkage: in each iteration, first smooth the mesh with factor λ, and then again with a *negative* factor $\mu < -\lambda$. Implement a function that performs Taubin smoothing, taking as parameters λ, μ, and the number of iterations. Test out your code with λ = 0.33 and μ = –0.34.
2. Implement any *one* of the following three applications of the basic mesh editing operations:

   - Loop subdivision (Loop 1987) of a triangle mesh, as discussed in class. I can think of two ways you can implement this:

     - Create a brand new mesh from scratch with many more vertices and 4 times as many faces, assigning the vertices their positions according to the Loop subdivision rules. You will then have to build the connectivity data as well, so that subdivision can be applied again.
     - Implement edge split and flip operations to modify the given mesh while updating the connectivity data. Then Loop subdivision can be performed by splitting all the original edges of the mesh, then flipping every edge that

connects an original vertex to a newly created vertex. After that, update all the vertex positions. (It's probably better to precompute the updated vertex positions before doing any splits, because it's easier to find the relevant neighbourhoods.)

- Isotropic remeshing (Botsch and Kobbelt 2004, Sec. 4). Here you will need to implement all three remeshing operations, i.e. split, flip, and collapse. Then, you would have to apply them repeatedly as discussed in class:

  - While there is any edge longer than 4/3 times the target length, split it.
  - While there is any edge shorter than 4/5 times the target length, and collapsing it would not create a long edge, collapse it to its midpoint.
  - While there is any edge such that flipping it would improve vertex degrees by reducing $\sum_i (d_i - 6)^2$, flip it.
  - Optionally, smooth out the vertex distribution by averaging along the tangent plane. See the paper for details; this step is optional for this assignment.

  Carry out this cycle multiple times (about 5) to get a mesh with edge lengths close to the target.

- Mesh simplification (Garland and Heckbert 1997). This requires only edge flips and collapses; however, you also need to understand and implement the quadric error metric used in this paper. Briefly, for any plane $\mathbf{n}^T\mathbf{p} - c = 0$, we can define a 4×4 matrix $\mathbf{K} = \begin{bmatrix} \mathbf{n}\mathbf{n}^T & -d\mathbf{n} \\ -d\mathbf{n}^T & d^2 \end{bmatrix}$ such that the squared distance between any point and the plane is simply $(\mathbf{n}^T\mathbf{p} - c)^2 = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}^T \mathbf{K} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$. At each vertex, we store a matrix $\mathbf{Q}$ which is the sum of the $\mathbf{K}$ matrices of its adjacent faces. Then the algorithm is simple:

  - For an edge between vertices i and j, find the homogeneous position $\tilde{\mathbf{p}}$ which minimizes the cost $\tilde{\mathbf{p}}^T(\mathbf{Q}_i + \mathbf{Q}_j)\tilde{\mathbf{p}}$.
  - Collapse the edge with lowest cost, and place the new vertex not at its midpoint but at the optimal position $\tilde{\mathbf{p}}$.
  - Repeat until the desired number of vertices is attained.

Submission

Submit your assignment on Moodle before midnight on the due date. Your submission should be a zip file that contains your entire source code for the assignment, not including any binary object files or executables. The zip file should also contain a PDF report that includes the screenshots of the following:

- both simple meshes (plane and sphere) created by your code,
- the cube, teapot, and bunny meshes loaded from OBJ files,
- results of naïve smoothing and Taubin smoothing on the noisy cube mesh, and
- results of one of the mesh editing operations (subdivision / remeshing / simplification) on any of the given meshes. If you do subdivision, show results both of subdividing once and subdividing again.

If there are any components of the assignment you could not fully implement, in the PDF you should also explain which ones they are and what problems you faced in doing them.

Separately, each of you will *individually* submit a short response in a separate form regarding how much you and your groupmate(s) contributed to the work in this assignment.