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 will provide starter code 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. We will provide a few example meshes in the OBJ file format, which is 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. Implement mesh fairing using the “umbrella operator” and Taubin smoothing [to be added]
2. Implement Loop subdivision [to be added]