TP 1: C structs and meshes

1 Indexed triangular meshes

1) Implement a C structure (key word `struct`), that we shall name `Vertex`, which models a point in 2D space. Use `double` precision for coordinates.

2) Implement a structure named `Triangle` for an indexed triangle in the plane. Indexed here mean that that the triangle vertices are not given explicitly, but through their index (i.e. position) in an array of vertices.

3) Implement a `Mesh2D` data structure that models a triangular 2D mesh. That structure should contain the following 4 pieces of information :

   - an integer `nv` representing the number of vertices in the mesh,
   - (the address of) an array of `Vertex`, named `vert`, where actual vertices are to be found or loaded,
   - an integer `nt` which represents the number of triangles in the mesh,
   - (the address of) an array of `Triangle`, named `tri`, where actual triangles are to be found or loaded.

4) Write a function that initialises a `Mesh2D` data structure able to load a given number of vertices and triangles, with the following prototype: `int initialize_mesh2D(struct Mesh2D* m, int vtx_capacity, int tri_capacity)`. This function should allocate memory. Write a corresponding dispose function : `void dispose_mesh2D(struct Mesh2D* m)`, which shall in particular release allocated memory.

5) Write a function with prototype `double area_mesh2D(struct Mesh2D* m)` that computes the (signed) area of a mesh.

6) (*More challenging*) Modify the data structures above to describe a surface mesh in 3D space. Devise then a function that computes the volume enclosed by a surface mesh, assuming the latter is closed.
   Hint: Recall that by the divergence theorem

$$\int_\Omega \mathrm{div}(\vec{X}) = \int_{\partial\Omega} \vec{X} \cdot \vec{n}$$

whenever Ω (with piecewise $\mathcal{C}^1$ boundary) is an open bounded set and $\vec{X}$ is a $\mathcal{C}^1$ vector field, and where $\vec{n}$ refers to the unit outward normal to Ω. Choosing $\vec{X}(x, y, z) = (x, y, z)$ (for which $\mathrm{div}\vec{X} = 3$), allows to transform volume integrals into flux integrals, over triangles in our case. For the latter, check that for an affine vector field the flux is equal to the product of the triangle area with the scalar product of the triangle normal vector and the vector field evaluated at the triangle barycentre.

2 Reading, writing (and visualizing) a mesh

There are many different file formats for encoding meshes, in particular because a mesh data structure can contain much more information (normals, additional scalar or vector fields associated to vertices and/or triangles), depending on the application, than the most simple one we have described above. In these exercise notes, we shall restrict to the INRIA mesh format.

1) Open the scientific report introducing the `medit` mesh visualization software at `https://ljll.fr/frey/publications/RT-0253.pdf` and find in there the specs for the `.mesh` file format.

2) Create a function with prototype `int read_mesh2D(struct Mesh2D* m, const char* filename)` that reads a 2D mesh from a `.mesh` file and loads it into a `Mesh2D` data structure. What to do in case the file contains a different kind of mesh or unnecessary vertex or triangle attributes is up to you.

3) Create a function with prototype `int mesh2D_to_gnuplot(struct Mesh2D* m, const char* filename)` which writes the data of a `Mesh2D` data structure into a new file which can then be sent directly to `gnuplot` for visualization.

4) Write a function with prototype `int write_mesh2D(struct Mesh2D* m, const char* filename)` that writes the data of a `Mesh2D` data structure into a new file using the `.mesh` file format.

5) (*More challenging*) Browse or clone the code at `https://github.com/didiersmets/Myosotis`. It contains C++ code for loading and visualizing surface meshes with a potentially huge number of triangles, by using a multi resolution simplification process. Your goal here is not to understand the whole code archive, but its overall structure and build process (using `cmake`) in order to add the possibility to use INRIA `.mesh` mesh files as inputs (the present version in the archive only accepts Wavefront `.obj` and Stanford `.ply` mesh file formats).