Meshes and Computational Geometry
Master 2 Informatique ID3D + ECL

TP2 and 3 – Differential properties on surfaces

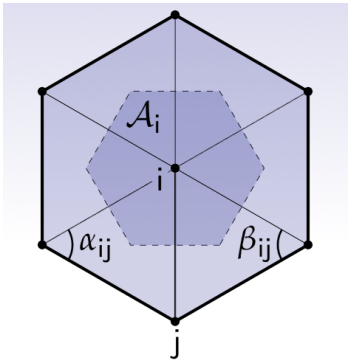
In this project, the triangular mesh data structure is used to store triangulations of the surface of an object (vertices corresponding to 3D points) as well as triangulations of points in 2D. The aim of this session is **analyzing the differential properties of signals** defined on these surfaces.

Before proceeding, add to your work the routine of loading a mesh stored in an OFF in the triangulated mesh data structure by implementing the face adjacency identification algorithm.

Laplacian of a discrete function defined on a mesh

- Given a surface embedded in $\mathbb{R}^3$, we only have an approximation of this surface in the form of a triangulated mesh. We can obtain an estimate of the Laplacian of a scalar function `u` known at each vertex of the mesh. `u` can be given in the form of a vector in which the `u` values are indexed in the same order as the vertices to which they correspond: `u[i]` is the value u_i of the function at the vertex of index `i`). To do this, we use the cotangent formula.

$$(Lu)_i = \frac{1}{2A_i} \sum_j (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$$



- A_i corresponds to the area of the patch of surface corresponding to the vertex s_i. You can estimate it as one-third of the area of the triangles incident to vertex `s`. If you prefer, you can improve this approximation by considering the barycenters of the faces, so as to divide them into more relevant area elements (as illustrated above).

Heat diffusion equation on a surface

- The Laplacian is useful for solving the heat diffusion equation on a surface by taking into account the intrinsic geometry of the surface, independent of any parametrization.

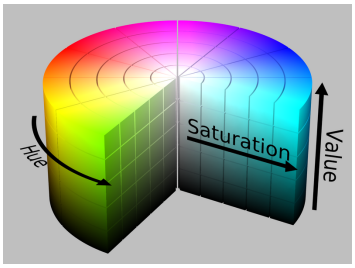
$$\Delta u = \frac{\delta u}{\delta t}$$

- Add a Laplacian operator to your triangular mesh data structure. You can test it by adding a source of heat at a given vertex of your surface (the temperature at this point is constant) and consider the propagation of temperature over the other vertices of the mesh. This is an evolution over the time.

Normal and average curvature estimation at a vertex

- In order to get an estimate of the normal and average curvature at each vertex, we compute the Laplacian of each of the coordinate functions:
    - Compute the Laplacian of the function `u` corresponding to the abscissa `x` (resp. ordinate `y` and altitude `z`) of each vertex position **`s`**.
    - The Laplacian vector $\Delta\mathbf{s}$ is composed of `(Δx, Δy, Δz)`.
    - $\Delta\mathbf{s}$ corresponds to `2H`**`n`**, where `H` is the mean curvature at **`s`** and **`n`** is the normal vector.

- Display the mesh with curvature-dependent coloring (you can, for example, express your colors in the HSV coordinates https://fr.wikipedia.org/wiki/Teinte_Saturation_Valeur).



Improved software implementation in C++

Your implementation of the mesh data structure can be improved by adding a set of iterators to navigate on a triangulation.

Iterators

Complete your `mesh` module with a set of iterator classes to visit the faces and the vertices of a `Mesh`:

- `Iterator_on_faces`: Iterator that can be used to visit the faces of a `Mesh` triangulation, in the same way as STL iterators can be used to visit the elements of an STL container. The order in which the faces are visited is not important. Do not forget to provide the additional member functions `faces_begin()` and `faces_past_the_end()` in the `Mesh` data structure. These should return some `Iterator_on_faces` values.

  In your implementation, the `Iterator_on_faces` will contain an access to the associated `Mesh` triangulation (and therefore to its face container), as well as the current position (an index). The class will also have an overload of `operator ++` to shift the current position to the next one, and an overload of `operator *` returning a reference on the current face (or the global index of the current face, depending on the design you prefer).

- `Iterator_on_vertices`: Iterator that can be used to visit the vertices of a `Mesh` triangulation. As previously, your `Mesh` data structure should provide the member functions `vertices_begin()` and `vertices_past_the_end()` returning some values of `Iterator_on_vertices`.

- `Circulator_on_faces`: Special topological iterator used to iterate counterclockwise on the set of incident faces at a given vertex. In the `Mesh` data structure, you should provide a member function `incident_faces(Sommet & v)` returning a `Circulator_on_faces` positioned on one of the incident faces at vertex `v`. An other design is to provide a member function `incident_faces(VertexIndex vi)` returning a `Circulator_on_faces` positioned on one of the incident faces at the vertex of global index `vi`. In your implementation, the iterator will contain an access to the associated triangulation, an access to the vertex around which it turns (an index), as well as an access to the current face (an index). The class will also have an `operator ++` overload in order to switch from the current face to the next one, and an `operator *` overload returning a reference on the current face (or the global index, depending on the design you prefer).

- `Circulator_on_vertices`: Iterator used to iterate, counter-clockwise, on all the vertices that are adjacent to a vertex. The class `Mesh` should contain some functions `adjacent_vertices(Sommet & v)` returning a `Circulator_on_vertices` positioned on one of the vertices adjacent to vertex `v`. As previously you can choose an other design of this function by using vertex indexes.

- Test all your iterator classes in a small test program:

```
Mesh titi;
titi.loadOFF(''cube.off'');
Iterator_on_vertices its;
Circulator_on_faces cf ;
for (its=titi.vertices_begin();
     titi !=titi.vertices_past_the_end();
     ++its)
{
  Circulator_on_faces cfbegin
         =titi.incident_faces(*its) ;
  int cmpt=0 ;
  for (cf=cfbegin,++cf;
       cf!=cfbegin;
       cf++)
     cmpt++ ;
  std ::cout<< ''valence of the vertex ''
            << cmpt <<std ::endl ;
}
```

- Improve your implementation of your Laplacian by using your iterators.