1

R3: $<, >, <=, >=$
ordre lexicographie sur les pair est deja dans la stl

1.1 Multimaps

(Requires a less-than comparison function.)

$O(3n_T)$: jede Kante jedes Dreiecks als Schlüssel
$|K|$ $O(log(n_T))$: Zugriff auf Container fur jede Kante
$\Rightarrow O(n_T log(n_T))$
speichern in 2dim Array: Zugriff in konstanter Zeit

unordered map(hash tables) has constant time performance on all operations provided no collisions occur. When collisions occur, traversal of a linked list containing all elements of the same bucket (those that hash to the same value) is necessary, and in the worst case, there is only one bucket; hence O(n)

1.2 Listen

$O(3n_T)$, push front(): jede Kante jedes Dreiecks einfügen
$O(NlogN)$, container size N: sort(): definiere hierfür $<$ für R3 (Nach ersten und dann nach zweitem Element sortieren etc)

doppelt verlinkt! und dann mit Kante und Sommet prev und next von Listenelement!
konstanter Zugriff mit Iterator!

$O(3n_T)$: pop front und speichern in 2 dim array

Alternative für 2D array für konstanten Zugriff?
Daten könnten direkt in 2D array gespeichert werden, Listen und Maps überflüssig

1.3 aire

orientierter Flächeninhalt

$$(a_i, b_i, p) = (a_i \times b_i) \cdot p = \det(a_i, b_i, p)$$

bei det Vektoren in Zeilen

2 Première tentative de documentation

3 Le projet

4 La classe template T3 et la classe Triangle

T3 O

4.1 The class triangle

The class triangle inherits form the class T3. Its derived members x, y, z are specialized as integers representing the position of its defining vertices in the array of the points of the given mesh. This array is a member of the class mesh (should be changed from maillage to mesh...) in 5. In addition, it has three extra integer members *neighbor1, neighbor2, neighbor3* representing the position of the adjacent triangles in the array of triangles which is also a member of the class mesh. An adjacent triangle $t^{'}$ is said to be *neighbor1* of a triangle t if x is the first vertex of t and x is not a vertex of $t^{'}$. The same holds for the indices $2, 3$. Note that this relation is not symmetric, that is to say that $t^{`}$ is *neighbor1* does not imply that t is *neighbor1* of $t^{'}$. At the creation of a new triangle the latter three members are initialized by -1 which means that a triangle has no neighbors when it is created. The neighbors are set via the functions *setAdjacencyViaMultimap* 5.2.1, respectively *setAdjacencyViaList* 5.2.2. After the execution of one of these two functions $neighbor1 = -1$ describes the case that there is no adjacent triangle on the opposite side of the first vertex. Since the neighbors are private members there are getter and setter in order to read and manipulate them.

5 The class mesh

The class mesh contains all necessary information of the mesh and the search of a point in (or outside) the mesh can be realized by its member functions. Its private members are pointers to the arrays of points and triangles of the mesh as integers who store the size of these two lists. In order to create an object of type mesh the name of the desired .msh file has to be transmitted. The file is read by the functions *LoadNodes* and *LoadTriangles* 5.1 who then initialize the members *triangles, sommets, numbTriangles, numbSommets*. There are getter and setter for *numbTriangles, numbSommets* as well as getter for *triangles, sommets*. The functions *LoadTriangles, LoadNodes* are their setters.

5.1 LoadNodes and LoadTriangles

Both functions are called in the constructor of the class mesh setting the members *nodes (VERTICES??)* and *triangles*. They work basically the same way with the small difference that they create arrays of different data types and searching for different key words in the .msh file. At first the .msh file is opened according to its name. Then the functions search for the line indicating the number of nodes respectively the number of triangles. To make the code work the next line must include the according number which is then stored in *numbSommets*, respectively *numbTri*. These numbers also define the size of the arrays of type T3<double> and Triangle which are created by new. The following lines must include the coordinates of all nodes, respectively the positions of the triangles in the table of nodes. Both functions finally return the arrays filled with according nodes or triangles.

Adj vis Map, Exp Gnu
O Prom, Adj vis List, find Sommets und Rec

5.2 Trouver le triangle adjacent

5.2.1 setAdjacencyViaMultimap

Le but de cette méthode est l'initialisation des membres *neighbor1, neighbor2, neighbor3* de la classe triangle. Les membres sont décrits pas leurs position dans la liste (ARRAY) des triangles. L'initialisation est réalisé en utilisant le container *multimaps*. Pour chaque triangle (a_1, a_2, a_3) on ajoute trois éléments au multimap où les arêtes $\{a_i, a_j\}$, $i, j \in \{1, 2, 3\}$, $i \neq j$ présentent les clés et le numéro de triangle est le valeur appliqué. Par conséquent, l'initialisation se produit dans $O(3n_T)$ et si deux triangles (t_1, t_2) sont adjacents par l'arête $\{a, b\}$, les éléments $(\{a, b\}, t_1)$, $\{a, b\}, t_2)$ ont les mêmes clés. Les triangles suivants sont trouvés par la procédure suivante:

1. On parcourt sur l'entière multimap en sauvegardant l'indice de triangle $t_1 = (a_1, a_2, a_3)$ associé à l'arête $\{a_i, a_j\}$ récente et écrasant l'élément $(\{a_i, a_j\}, t)$ de la map. La complexité est $O(3n_T)$.

2. On cherche si le clé $\{a_i, a_j\}$ est associé à un autre triangle t_2. Dans ce cas les triangles t_1, t_2 sont adjacents. L'opération de recherche se passe selon la recherche dans un multimap dans $O(logn_T)$.

3. On met t_2 comme voisin $k \in \{1, 2, 3\}$ si le sommet de t_1 au position k n'est pas un sommet de t_2 et vice versa. Cet opération est réalisé dans un temps constant.

5.2.2 setAdjacencyViaList

As in setAdjacencyViaMultimap, this method sets the members, describing the neighbors, for each triangle in the considered mesh. In the same manner as before, the members are described using their position in the triangle array.
The idea is to store each triangle, in varying order, three times in a list. For the triangle t with the points (a_1, a_2, a_3), the triangles $(a_1, a_2, t), (a_1, a_3, t)$ and (a_2, a_3, t) are appended to the list. The creation of this list has complexity $O(3n_T)$.
After the creation, the list is sorted in lexicographic order. The complexity for sorting a list with n_T elements is $O(n_T logn_T)$.
Due to the lexicographical ordering, the list allows now to define adjacencies between triangles. Two consecutive triangles of the list are considered. The first two entries of the triangles are compared. If they are equal, the triangles share an edge and can be defined as adjacent. $(a, b, t_1), (a, b, t_2)$
complexity $O(3n_T)$
In summary, the method setAdjacencyViaList has complexity $O(n_T logn_T)$.
Laufzeitvergleich

5.3 L'algorithme promenade

$min_negrandom_neg$

5.4 La visualisation avec gnuplot

6 5 find covering Triangles

7 Testumgebung - main