Université de Montpellier

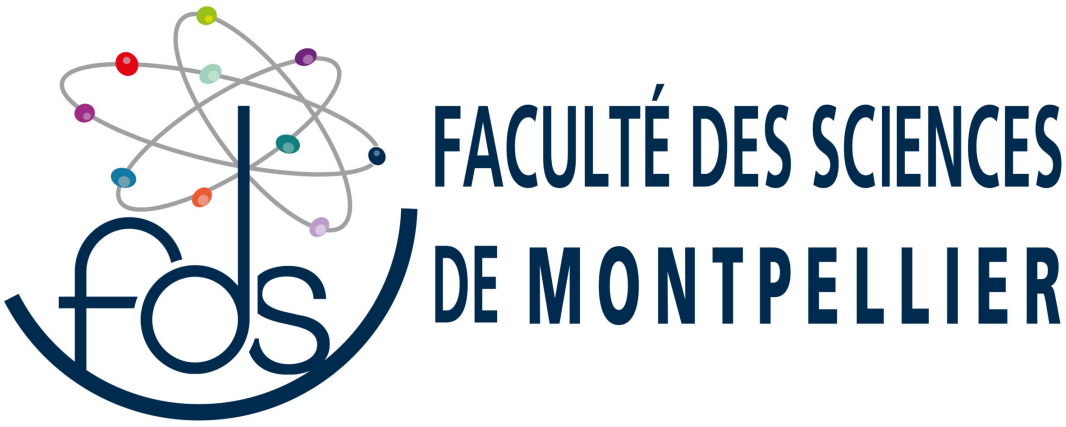
M1 - IMAGINE - Modélisation et Géométrie discrète
Compte Rendu TP6 - Représentation par carte combinatoire généralisée

Etudiant :
Guillaume Bataille

Encadrant :
Noura Faraj
Marc HARTLEY

Année 2022-2023

Lien du TP github



Sommaire

1 La structure de 2-G-Carte **2**

1.1 add_dart : Créer un nouveau brin 2

1.2 alpha (degree unique) : Renvoie le alpha degree resultant 2

1.3 alpha(vecteur de degrés) : Renvoie le alpha degree resultant 2

1.4 is_free : Teste si un dart est libre pour un alpha degré particulier 3

1.5 link_darts : Lie un dart1 a dart2 s'il sont libre 3

1.6 is_valid : Teste si pour tout brins, si on est bien cohérent sur les alphas 0, 1 et 2 . 4

2 Le parcours de structure **5**

2.1 orbit : Trouve les brins accessible via une liste l'alpha degrés 5

2.2 orderedorbit : Trouve les brins accessible via une liste d'alpha degré en utilisant l'involution 6

3 Le plongement géométrique **7**

4 La couture permettant de lier deux éléments de degrée 'degree' **8**

5 Caractéristique d'Euler-Poincaré (S - A + F) **9**

6 Visualisation **10**

7 Le dual d'une carte 2-G-Carte **11**

8 Comparaison avec des structures alternatives **14**

8.1 Changement necessaires pour encoder une simple 2-Carte 14

8.2 Half-edge vs 2Gmap 14

8.3 Vertices_indexed vs 2Gmap 14

Contexte Nous avons vu aujourd'hui différentes façon de représenter des maillages. Et nous nous sommes arrêter sur la représentation par carte combinatoire généralisé.
Objectif Nous allons essayer de recréer une structture de données de brins et de les manipuler afin d'avoir un comportement similaire à ce que nous avons pu observer lors du cours d'aujourd'hui. Dans cette structure, un dart est un brin, il possèdera un alpha_container, ce dernier contient les pointeurs vers les dart obtenu si on applique un alpha d'un degré particulier.

1. La structure de 2-G-Carte

1.1 add_dart : Créer un nouveau brin

```
// Créer un nouveau brin qui pour tout alpha 0 1 ou 2 pointe vers lui même
GMap::id_t GMap::add_dart() // NEW 1.a ajout d'un nouveau brin
{
    id_t dart = maxid; // Récup maxid pour l'id du dart courant
    maxid++; // Incrément de maxid pour le prochain dart
    alphas[dart] = alpha_container_t(dart,dart,dart); // Set les alpha a lui même
    return dart;
}
/*
Selected question :1a
2 | 2 2 2
1 | 1 1 1
0 | 0 0 0
add_dart seems valid
*/
```

1.2 alpha (degree unique) : Renvoie le alpha degree resultant

```
// Si on veut récup un alpha deg particulier
GMap::id_t GMap::alpha(degree_t degree, id_t dart) const // NEW 1.a fix l'affichage
{
    assert(degree < 3);
    assert(dart < maxid);
    return alphas.at(dart)[degree];
}
//Selected question :1b
//alpha and is_free seems valid
```

1.3 alpha(vecteur de degrés) : Renvoie le alpha degree resultant

```
// Si on veut récup une sucession de alpha deg successif stocké dans un vect (a lire a l'envers)
GMap::id_t GMap::alpha(degreelist_t degrees, id_t dart) const //NEW  1.b gère un vecteur de degré
{
```

```
    std::reverse(degrees.begin(),degrees.end()); // Inversion des degré
    for(degree_t degree : degrees){ // Pour tout les elements degree de degrees
        assert(degree < 3);
        assert(dart < maxid);
        dart = alphas.at(dart)[degree];
    }
}
//Selected question :1b
//alpha and is_free seems valid
```

1.4 is_free : Teste si un dart est libre pour un alpha degré particulier

```
// Teste si on a un point fixe : c'est un point qui lorsque qu'il subit alpha deg il reste le même
bool GMap::is_free(degree_t degree, id_t dart) const // NEW 1.b teste si un vect de degré est libre
{
    assert(degree < 3);
    assert(dart < maxid);
    return alpha(degree,dart) == dart;
//Selected question :1b
//alpha and is_free seems valid
}

```

1.5 link_darts : Lie un dart1 a dart2 s'il sont libre

```
// Lie un dart a un autre avec comme relation alpha degree ssi ils sont libres tout les deux.
bool GMap::link_darts(degree_t degree, id_t dart1, id_t dart2) // NEW 1.c
{
    // Test si ils sont libres
    if(!is_free(degree,dart1)) return false;
    if(!is_free(degree,dart2)) return false;

    //Association de 1 a 2 et 2 a 1 pour l'alpha degree;
    alphas[dart1][degree] = dart2;
    alphas[dart2][degree] = dart1;
    return true;
}

//Selected question :1c
//link_darts  seems valid
```

1.6 is_valid : Teste si pour tout brins, si on est bien cohérent sur les alphas 0, 1 et 2

```
1
2  bool GMap::is_valid() const
3  {
4
5      //darts() retourne la liste de tout les darts créé
6      for (id_t dart : darts()){ // Pour tout dart
7
8      //Alpha0
9      if(alpha({0,0},dart) != dart) // Si alpha 0 n'est pas une involution
10         return false; // pas valide
11     if(is_free(0,dart)) // Si dart est libre
12         return false;// pas valide
13
14     //Alpha1
15     if(alpha({1,1},dart) != dart) // Si alpha 1 n'est pas une involution
16         return false; // pas valide
17     if(is_free(1,dart)) // Si dart n'est pas libre en alpha1
18         return false; // pas valide
19
20     //Alpha2
21     if(alpha({2,2},dart) != dart )// Si alpha 2 n'est pas une involution
22         return false; // pas valide
23     if(alpha({0,2,0,2},dart) != dart) // Si on fait le tour et qu'on retombe pas sur dart
24         return false; // pas valide
25     }
26     return true;
27 }
28 //Selected question :1d
29 //is_valid seems valid
```

2. Le parcours de structure

2.1 orbit : Trouve les brins accessible via une liste l'alpha degrés

```
1
2  GMap::idlist_t GMap::orbit(const degreelist_t &alphas, id_t dart) const // NEW 2
3  {
4      idlist_t result;             // Liste resultat
5      idset_t marked;              // Set des objets marqués (set contient la methode .count())
6      idlist_t toprocess = {dart}; // Liste contenant les elements a traiter
7      id_t current_dart;           // Le dart courant
8      while (toprocess.size() != 0)
9      {                                    // Tant qu'il y a des elements a traiter
10         current_dart = toprocess.back(); // Recup le dernier element de la liste a traiter
11
12         if (marked.count(current_dart) == 0) // Si d n'est pas marqué
13         {
14             marked.insert(current_dart);    // On l'ajoute dans marked
15             result.push_back(current_dart); // On l'ajoute dans result
16             toprocess.pop_back();           // On passe au prochain a traiter
17             for (degree_t degree : alphas)  // Pour chaque degree de list_of_alpha_value
18             {
19                 toprocess.push_back(alpha(degree, current_dart)); // Rajouter alpha_degree(d) dans to_process
20             }
21         }
22         else // Si il est deja marqué
23         {
24             toprocess.pop_back(); // On passe au prochain a traiter
25         }
26     }
27     return result;
28 }
29 /*Selected question :2
30 7 | 6 0 7
31 6 | 7 5 6
32 5 | 4 6 5
33 4 | 5 3 4
34 3 | 2 4 3
35 2 | 3 1 2
36 1 | 0 2 1
37 0 | 1 7 0
38 Element de degree 0 :[0,1,3,5]
39 Element de degree 1 :[0,2,4,6]
40 Element de degree 2 :[0]*/
```

2.2 orderedorbit : Trouve les brins accessible via une liste d'alpha degré en utilisant l'involution

```
1
2  GMap::idlist_t GMap::orderedorbit(const degreelist_t &list_of_alpha_value, id_t dart) const
3  {
4      idlist_t result;
5      id_t current_dart = dart;
6      unsigned char current_alpha_index = 0;
7      unsigned char next_alpha_index;
8      size_t n_alpha = list_of_alpha_value.size();
9
10     do
11     {
12         result.push_back(current_dart);                                  // ajouter current_dart au resultat
13         next_alpha_index = list_of_alpha_value[current_alpha_index]; // prendre le prochain alpha de list_of_alp
14         current_alpha_index = (current_alpha_index + 1) % n_alpha;    // incrémenter current_alpha_index
15         current_dart = alpha(next_alpha_index, current_dart);         // changer current_dart par alpha_current_a
16     } while (current_dart = !dart);                                      // Tant que current_dart est différent de d
17
18     return result;
19 }
20
```

3. Le plongement géométrique

```
template <class T>
GMap::id_t EmbeddedGMap<T>::get_embedding_dart(id_t dart) const // NEW 3
{
    idlist_t list_orbit = orbit({1, 2}, dart); // Recupère la liste des orbites de degré 1,2

    for (id_t current_id : list_orbit) // Pour toute les orbites
    {
        if (properties.count(current_id) != 0) // Si il existe une propriété sur l'élement courant
        {
            return current_id; // On retourne l'id de cet element
        }
    }
    return dart;
}
/*Selected question :3
7 | 6 0 7
6 | 7 5 6 : (-5.000000,5.000000,0.000000)
5 | 4 6 5
4 | 5 3 4 : (-5.000000,-5.000000,0.000000)
3 | 2 4 3
2 | 3 1 2 : (5.000000,-5.000000,0.000000)
1 | 0 2 1
0 | 1 7 0 : (5.000000,5.000000,0.000000)*/
```

4. La couture permettant de lier deux éléments de degrée 'degree'

```
bool GMap::sew_dart(degree_t degree, id_t dart1, id_t dart2) // NEW 4.a et 4.b
{

    idlist_t list_orbit1, list_orbit2; // Liste des orbites 1 et 2
    bool res;                          // Le resultat booléen

    if (degree == 1) // Si le degr& vaut 1
    {
        res = link_darts(degree, dart1, dart2); // On tente directement le lien
    }
    else // Sinon
    {

        if (degree == 0) // Si degre = 0
        {                // On s'attarde sur les orbites de degrès 2 pour les lier (pour etre cohérents )
            list_orbit1 = orbit({2}, dart1);
            list_orbit2 = orbit({2}, dart2);
        }

        if (degree == 2) // Si degre = 2
        {                // On s'attarde sur les orbites de degrès 0 pour les lier (pour etre cohérents )
            list_orbit1 = orbit({0}, dart1);
            list_orbit2 = orbit({0}, dart2);
        }

        if (list_orbit1.size() != list_orbit2.size()) // On teste la taille des orbites, si pas égal
        {                                             // On ne lie pas et on retourne false
            return false;
        }

        for (size_t i = 0; i < list_orbit1.size(); i++) // Pour le nbr d'éléments  a lier
        {

            res = link_darts(degree, list_orbit1[i], list_orbit2[i]); // On lie l'élement de la liste 1 a celui
            if (!res)                                                 // Si ce liage échoue en renvoyant false,
                return false;                                         // Normalement il faudrait aussi unlink..
        }
    }

    return res;
}
// Tester code ./gmap 4a et 4b pour voir la liste des brins post coutures avec propriétés
```

5. Caractéristique d'Euler-Poincaré (S - A + F)

```
1  int GMap::eulercharacteristic() const // NEW 5
2
3  { // Version spéciale car on ne tavaille qu'avec des cubes dans ce tp
4      size_t sommet,
5          aretes, faces, dart_numbers;
6
7      sommet = elements(0).size(); // recupère le nombre de structure 0cellule
8      aretes = elements(1).size(); // recupère le nombre de structure 1cellule
9      faces = elements(2).size();  // recupère le nombre de structure 2cellule
10     std::cout << " sommet : " << sommet << " / aretes : " << aretes << " / faces : " << faces << " /  darts size
11     return sommet - aretes + faces;
12 }
13 /*Selected question :5
14  sommet : 8 / aretes : 12 / faces : 6 /  darts size : 48
15 Euler characteristic : 2*/
```

6. Visualisation

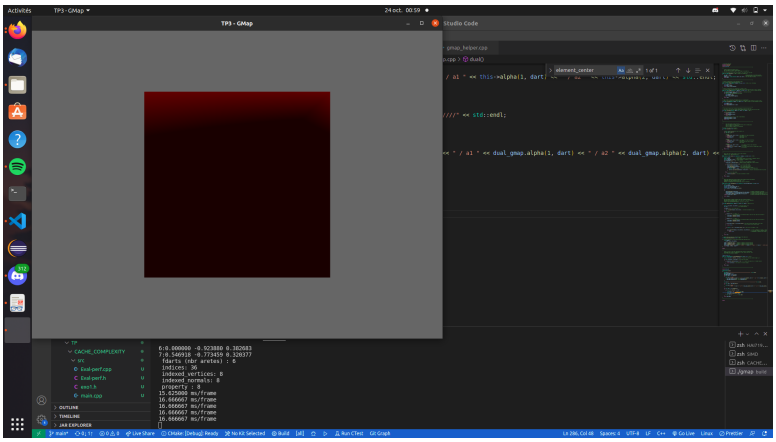


FIGURE 6.1 – Cube - Affichage montre que les calculs d'orbit et les autres fonctions sont bien fonctionnels

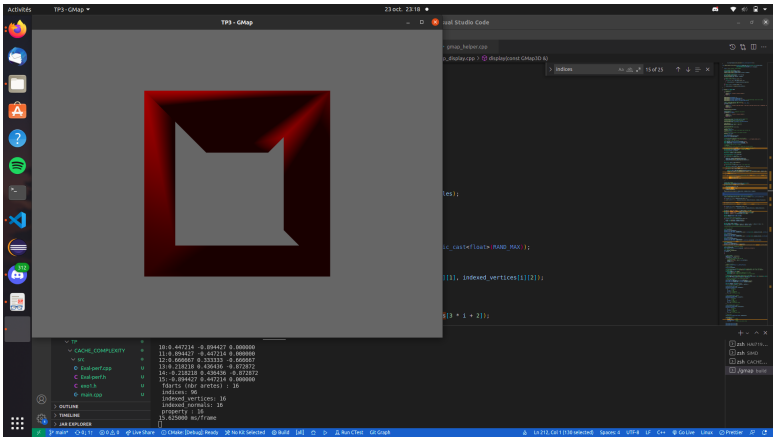


FIGURE 6.2 – Cube troué - Affichage montre que les calculs d'orbit et les autres fonctions sont bien fonctionnels

7. Le dual d'une carte 2-G-Carte

```
1
2  GMap3D GMap3D::dual() // NEW 7
3  {
4
5      GMap3D dual_gmap;                 // Création d'un nouveau GMap3D : dual_gmap
6      dual_gmap.maxid = maxid;          // On lui donne comme maxid le maxid courant de this
7      dual_gmap.alphas = this->alphas; // On lui attribue tout ce que alphas de this contenait
8      //dual_gmap.properties = this->properties;
9      for (id_t dart : this->darts()) // Pour chaque dart on inverse les alpha 0 avec les 2 pour le dual
10     {
11         dual_gmap.alphas[dart][0] = this->alphas[dart][2];
12         dual_gmap.alphas[dart][2] = this->alphas[dart][0];
13     }
14
15     for (id_t ele : this->elements(2)) // Pour toute les faces de this.
16     {
17         for (id_t id : orbit({0, 1}, ele)) // On parcours les faces de this via les orbit des 2-cellule
18         {
19             vec3_t pos = this->element_center(2, ele); // On determine les baricentres de ces points la
20             dual_gmap.set_position(ele, pos);          // On les set dans dual_gmap
21         }
22     }
23     return dual_gmap;
24 }
25 /*
26 Selected question :7
27 ---------------------- Not_dual_Gmap
28 47 | 46 40 36
29 46 | 47 45 37 : (-5.000000,5.000000,-5.000000)
30 45 | 44 46 28
31 44 | 45 43 29 : (-5.000000,-5.000000,-5.000000)
32 43 | 42 44 20
33 42 | 43 41 21 : (5.000000,-5.000000,-5.000000)
34 41 | 40 42 12
35 40 | 41 47 13 : (5.000000,5.000000,-5.000000)
36 39 | 38 32 26
37 38 | 39 37 27
38 37 | 36 38 46
39 36 | 37 35 47
40 35 | 34 36 14
41 34 | 35 33 15
42 33 | 32 34 2
43 32 | 33 39 3
44 31 | 30 24 18
```

```
30 | 31 29 19
29 | 28 30 44
12 | 13 11 41
11 | 10 12 22
10 | 11 9 23
9 | 8 10 0
8 | 9 15 1
7 | 6 0 16
6 | 7 5 17 : (-5.000000,5.000000,5.000000)
5 | 4 6 24
4 | 5 3 25 : (-5.000000,-5.000000,5.000000)
3 | 2 4 32
2 | 3 1 33 : (5.000000,-5.000000,5.000000)
1 | 0 2 8
0 | 1 7 9 : (5.000000,5.000000,5.000000)
13 | 12 14 40
14 | 15 13 35
15 | 14 8 34
16 | 17 23 7
17 | 16 18 6
18 | 19 17 31
19 | 18 20 30
20 | 21 19 43
21 | 20 22 42
22 | 23 21 11
23 | 22 16 10
24 | 25 31 5
25 | 24 26 4
26 | 27 25 39
27 | 26 28 38
28 | 29 27 45
------------------- dual_Gmap
47 | 36 40 46
46 | 37 45 47
45 | 28 46 44
44 | 29 43 45
43 | 20 44 42
42 | 21 41 43
41 | 12 42 40
40 | 13 47 41 : (0.000000,0.000000,-5.000000)
39 | 26 32 38
38 | 27 37 39
37 | 46 38 36
36 | 47 35 37
35 | 14 36 34
34 | 15 33 35
33 | 2 34 32
32 | 3 39 33 : (0.000000,0.000000,0.000000)
31 | 18 24 30
30 | 19 29 31
29 | 44 30 28
```

```
12 | 41 11 13
11 | 22 12 10
10 | 23 9 11
9 | 0 10 8
8 | 1 15 9
7 | 16 0 6
6 | 17 5 7
5 | 24 6 4
4 | 25 3 5
3 | 32 4 2
2 | 33 1 3
1 | 8 2 0
0 | 9 7 1 : (0.000000,0.000000,5.000000)
13 | 40 14 12
14 | 35 13 15
15 | 34 8 14 : (5.000000,0.000000,0.000000)
16 | 7 23 17
17 | 6 18 16
18 | 31 17 19
19 | 30 20 18
20 | 43 19 21
21 | 42 22 20
22 | 11 21 23
23 | 10 16 22 : (0.000000,0.000000,0.000000)
24 | 5 31 25
25 | 4 26 24
26 | 39 25 27
27 | 38 28 26
28 | 45 27 29 : (-5.000000,0.000000,0.000000)*/
```

8. Comparaison avec des structures alternatives

8.1 Changement necessaires pour encoder une simple 2-Carte

Il va falloir retirer l'alpha 0 car dans les 2-Carte, il n'y a pas ce type de parcours et de structure. Cela simplifie la strucutre tout en gardant la topologie générale de la forme représentée. Cependant, la gestion de la 0-cellule(les sommets) devient compliqué et la force des 2-G-carte comme le dual facilement obtenu ect.. n'est pas conservé.

8.2 Half-edge vs 2Gmap

L'half-edge est plus concret car on a des notions comme les faces, le brin suivant et le brin précédent. Il semble aussi être particulièrement efficace lorsqu'il s'agit d'une structure triangulaire.
La 2Gmap, elle, est moins gourmande en mémoire et est plus "bas niveau" ce qui permet d'avoir des manipulations plus profondes sur les structures topologique que l'on manipule. De plus, les 2Gmap sont généralisable aux structures géométriques, ce qui est un gros avantages.

8.3 Vertices_indexed vs 2Gmap

Ses deux structures sont bas niveau et permette de manipuler simplement et aisément les structures souhaités. Cependant, la ou les 2Gmap semble être optimisé pour avoir des informations topologiques, les structures de sommets indexés excelle en simplicité. Cela permet de faire des associations simple entre ces liste indexés et les triangles que l'on souhaite représenter ou encore de les envoyer facilement au gpu. C'est à la fois la force et la faiblesse de cette représentation de points indexés qui reste l'approche favorisé dans le cadre de nos études actuelles.