Cartes Combinatoires

Le canevas du projet implémenant la structure de données GMap sont dans le répertoire src. Comme les TPs précédents pour compiler le projet, dézippez le, créez un répertoire build et utilisez `cmake ..`.

Les spécifications de la structure sont dans le fichier `gmap.hpp`. Les fonctions a compléter sont dans `gmap.cpp`. Les fonctions déja implémentées sont dans `gmap_helper.cpp`. La fonction d'affichage (inspirée du TP précédent) est dans le fichier `gmap_display.cpp`. Le fichier de test des différentes fonctions s'appelle `gmap_main.cpp`. Ses fonctions sont nommées `questionX` et correspondent aux exercices qui suivent. Pour les tester, il suffit de les appeler dans la fonction `main` à la fin du fichier `gmap_main.cpp`.

1/ Implémenter une structure de 2-G-carte.

Cette GMap sera encodée en représentant chaque brin par un identifiant. Chaque relation alpha (0, 1 et 2) sera codée par un dictionnaire (dict). Etudiez et completez si necessaire la structure suivante

```
class GMap {
public:

    // id type
    typedef unsigned long int id_t;
    // degree type. should be in [0,1,2]
    typedef unsigned char degree_t;

    // a triplet of alpha value
    struct alpha_container_t {
        alpha_container_t(id_t a0 = 0, id_t a1 = 0, id_t a2 = 0) {
            values[0] = a0; values[1] = a1; values[2] = a2;
        }
        id_t& operator[](degree_t index) { return values[index]; }
        const id_t& operator[](degree_t index) const { return values[index];
}

        alpha_container_t flip() const { return
alpha_container_t(values[2],values[1],values[0]); }

        id_t values[3];
    };

    // a type for list of id. usefull for processing
    typedef std::vector<id_t> idlist_t;
    // a type for set of id . usefull to check if darts have already been
processed.
    typedef std::unordered_set<id_t> idset_t;
    // a type of list of degree
    typedef std::vector<degree_t> degreelist_t;
    // a type of map of dart id to alpha values for this id.
    typedef std::unordered_map<id_t, alpha_container_t> idalphamap_t;

```

```
    ...

protected:
    // use to generate dart id
    id_t maxid;
    /* a map with key corresponding to dart id and value a alpha_container_t
       that contains the 3 value of alpha for the given dart.
      example : { 0 : [0,1,2] }.
    */
    idalphamap_t alphas;
};

```

Complétez les fonctions suivantes dans le fichier `gmap.cpp`

```
    // Return the application of the alpha_deg on dart
    id_t alpha(degree_t degree, id_t dart) const;

    // Return the application of a composition of alphas on dart
    id_t alpha(degreelist_t degrees, id_t dart) const;


    //  Test if dart is free for alpha_degree (if it is a fixed point)
    bool is_free(degree_t degree, id_t dart) const;

    /*
        Test the validity of the structure.
        Check that alpha_0 and alpha_1 are involutions with no fixed points.
        Check if alpha_0 o alpha_2 is an involution
    */
    bool is_valid() const;

    /*
        Create a new dart and return its id.
        Set its alpha_i to itself (fixed points)
    */
    id_t add_dart();

    // Link the two darts with a relation alpha_degree if they are both
free.
    bool link_darts(degree_t degree, id_t dart1, id_t dart2);
```

Une fois completer, vous pouvez tester cette structure avec la function `question1` du fichier `gmap_main.cpp`.

2/ Encoder les parcours de la structure

Complétez les fonctions de parcours par calcul d'orbite.

```
    /*
        Return the orbit of dart using a list of alpha relation.
        Example of use : gmap.orbit(0,[0,1]).
    */
    idlist_t orbit(degreelist_t alphas, id_t dart);
    /*
    Canvas:
```

```
    {
        idlist_t result;
        idset_t marked;
        idlist_t toprocess = {dart};
        # Tant qu'il y a des elements à traiter
            # prendre un element d à traiter
            # si d n'est pas dans marked
                # rajouter d dans result et dans marked
                # pour chaque degree de list_of_alpha_value
                    # rajouter alpha_degree(d) dans toprocess

        return result;
    }
    */

    /*
        Return the ordered orbit of dart using a list of alpha relations by
applying
        repeatingly the alpha relations of the list to dart.
        Example of use. gmap.orderedorbit(0,[0,1]).
        Warning: No fixed point for the given alpha should be contained.
    */
    idlist_t orderedorbit(degreelist_t list_of_alpha_value, id_t dart);
    /*
    Canvas:
    {
        idlist_t result;
        id_t current_dart = dart;
        unsigned char current_alpha_index = 0;
        size_t n_alpha = list_of_alpha_value.size();
        # Tant que current_dart est different de dart
            # ajouter current_dart au resultat
            # prendre le prochain alpha de list_of_alpha_value avec
current_alpha_index
            # incrémenter current_alpha_index
            # changer current_dart par alpha_current_alpha(current_dart)

        return result;
    }
    */
```

Cela permettra de faire marcher la fonction `elements` qui détermine les i-cellules de la carte.

```
GMap::idlist_t GMap::elements( degree_t degree) {
    idlist_t elements;
    idlist_t vdarts = darts();
    idset_t sdarts (vdarts.begin(), vdarts.end());

    degreelist_t  list_of_alpha_value = {0, 1, 2};
    list_of_alpha_value.erase(list_of_alpha_value.begin()+degree);

    while (!sdarts.empty()){
        id_t dart = *sdarts.begin();
        sdarts.erase(sdarts.begin());
        idlist_t elementi = orbit(list_of_alpha_value, dart);
        for (id_t d : elementi) sdarts.erase(d);
        elements.push_back(dart);
```

```
    }

    return elements;
}
```

Vous pouvez tester cette structure avec la fonction `question2`.

3/ Encoder le plongement géométrique

La classe GMap est sous classée en GMap3D pour intégrer le plongement géométrique.

```
template<class T>
class EmbeddedGMap : public GMap {
public:
    typedef T property_t;
    typedef std::unordered_map<id_t, property_t> idpropmap_t;

    EmbeddedGMap() {}
    ~EmbeddedGMap() {}

    /*
        Check if a dart of the orbit representing the vertex has already
been
        associated with a value in propertydict. If yes, return this dart,
else
        return the dart passed as argument.
    */
    id_t get_embedding_dart(id_t dart) ;



    // Retrieve the coordinates associated to the vertex <alpha_1, alpha_2>
(dart)
    const T& get_property(id_t dart) ;


    // Associate coordinates with the vertex <alpha_1,alpha_2>(dart)
    id_t set_property(id_t dart, T prop) ;

protected:

    // A map that associate a property to each dart.
    idpropmap_t properties;
};

/*
    The GMap3D extent GMap class with embedding.
    The property added is an index of position.
    A list of indexed position is also maintained.
*/
class GMap3D : public EmbeddedGMap<id_t> {
public:

    ...

    // A new indexed position is added to the list
```

```
    // The index is associated to the dart.
    void set_position(id_t dart, vec3_t pos) {
        id_t pid = positions.size();
        positions.push_back(pos);
        set_property(dart, pid);
    }

    // A vector of indexed position
    std::vector<vec3_t> positions;

};
```

Implementer la fonction qui permet de determiner le brin qui contient l'information de plongement pour une i-cellule donnée. Pour cela, il vous faut parcourir l'orbite de la i-cellule et vérifier si chaque brin n'a pas de valeur dans le dictionnaire de valeur de plongement. Grace a cela, les fonctions `get_position` et `set_position` permettront d'associer une position à une 0-cellule.

A noter que cette fonction est dans le fichier `gmap.hpp`.

```
template<class T>
GMap::id_t EmbeddedGMap<T>::get_embedding_dart(id_t dart)
/*
Canvas
{
    Tester pour chaque brin de l'orbit<1,2> de dart s'il possede une propriété dans properties.
    Si oui, le retourner.
    Si aucun brin de l'orbite n'a de propriété, retourner dart.
}
*/
```

Vous pourrez tester cela avec la fonction `question3` qui associe des coordonnées aux sommets d'un carré.

4/ Coder la fonction de couture qui permet de lier deux éléments de degrée ‘degree’.

Si vous liez deux brins par alpha_2, il faut aussi lier leurs images par alpha_0 pour satisfaire la contrainte que alpha_2(alpha_0) est une involution. La même chose pour lier par alpha_0, il faut également lier les images par alpha_2. Si le lien est fait par alpha_1, aucune contrainte s'applique. Il faut juste lier les brins.

```
/*
    Sew two elements of degree 'degree' that start at dart1 and dart2.
    Determine first the orbits of dart to sew and heck if they are compatible.
    Sew pairs of corresponding darts
    # and if they have different embedding  positions, merge them.
*/
bool GMap::sew_dart(degree_t degree, id_t dart1, id_t dart2)
/*
Canvas:
{
    Si degree est égal a 1:
```

```
        Simplement lier par alpha_1 les deux brins
    Sinon:
        Si degree == 0: Trouver les orbites par alpha_2 des deux brins.
        Si degree == 2: Trouver les orbites par alpha_0 des deux brins.

        Vérifier que les orbites sont compatibles (meme taille). Sinon retourner false

        Lié deux a deux par alpha_degree les brins des 2 orbites.

        return true
}
*/
```

Vous pouvez maintenant tester la création d'un cube et d'un cube troué (holeshape) avec les fonctions `question4a` et `question4b`.

5/ Coder une fonction qui calcule la caractéristique d'Euler-Poincaré (S-A+F)

Complétez la fonction qui calcule la caractéristique d'Euler-Poincaré

```
// Compute the Euler-Poincare characteristic of the subdivision
int GMap::eulercharacteristic()
/*
Canvas:
{
    return S - A + F
}
```

Vous pourrez tester cette fonction sur la structure de cube avec la fonction `question5`.

6/ Visualiser les objets

Complétez la fonction `display` du ficher `gmap_display.cpp` permettant de visualiser la carte simplement comme un ensemble de faces.

```
int display(const GMap3D& gmap)
```

Il faut transformer la gmap en liste de sommet et d'indices. A noter que les faces dans les exemples sont des quadrilatère qu'il vous faudra transformer en triangle ([a,b,c,d] -> [a,b,c]+[a,c,d]). Chaque arête étant représenté par 2 brins, il ne faudra considérer qu'un brin sur 2 pour déterminer les positions. A noter également qu'il vous faut calculer les normales de chaque faces. La valeur de propriété de chaque vertex peut pour l'instant etre définie aléatoirement pour la visualisation.

```
    std::vector<unsigned short> indices; //Triangles concaténés dans une liste
    std::vector<glm::vec3> indexed_vertices;
    std::vector<glm::vec3> indexed_normals;
    std::vector<unsigned int> property;
```

Vous pourrez alors tester la visualization du cube et du cube troué avec `question6a` et `question6b`.

7/ Coder une fonction qui calcule le dual d'une 2-G-Carte.

```
GMap3D GMap3D::dual()
/*
Canvas:
{
    GMap3D dual_gmap;
    dual_gmap.maxid = maxid;

    Créer un dictionnaire alpha egal a celui de this tel que les valeurs de alphas_2 et alphas_0 soit inversés.

    Pour chaque brin correspondant aux faces de this (et donc aux sommets de dual_gmap),
    Calculer le barycentre de la face avec la fonction element_center.
    Associer cette position au brin correspondant de dual_gmap


    return dual_gmap;

}
```

Vous pourrez tester cette fonction sur la structure de cube avec la fonction `question7`.

8/ Comparaison avec des structures alternatives

Evaluer les changements necessaires pour encoder une simple 2-Carte. Qu'est ce que vous y gagneriez ? perdriez ?

Comparer cette structure avec une structure Half-Edge ? Avec une simple liste de points indexés ?