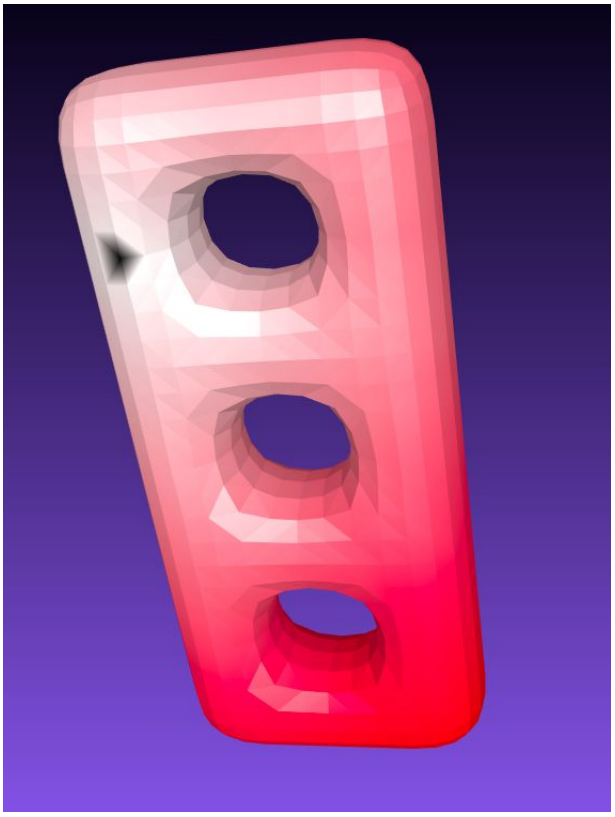
Rapport du TP4
FERREIRA Donovan / LETERRE Maxime

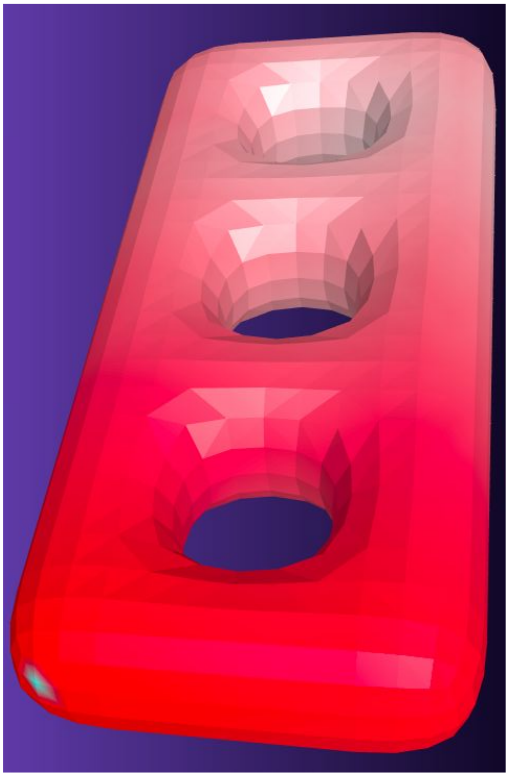
Dans notre projet, nous avons séparé les classes Facet, Halfedges et Vertex pour rendre le code plus lisible.
Nous regroupons les fichiers .off créés par notre programme dans différents dossiers :

- figures_normales : les fichiers de base
- figures_compo : coloration par composantes connexes
- figures_geo : coloration par distance géodésique
- figures_genre : coloration des composante en fonction de leurs genre

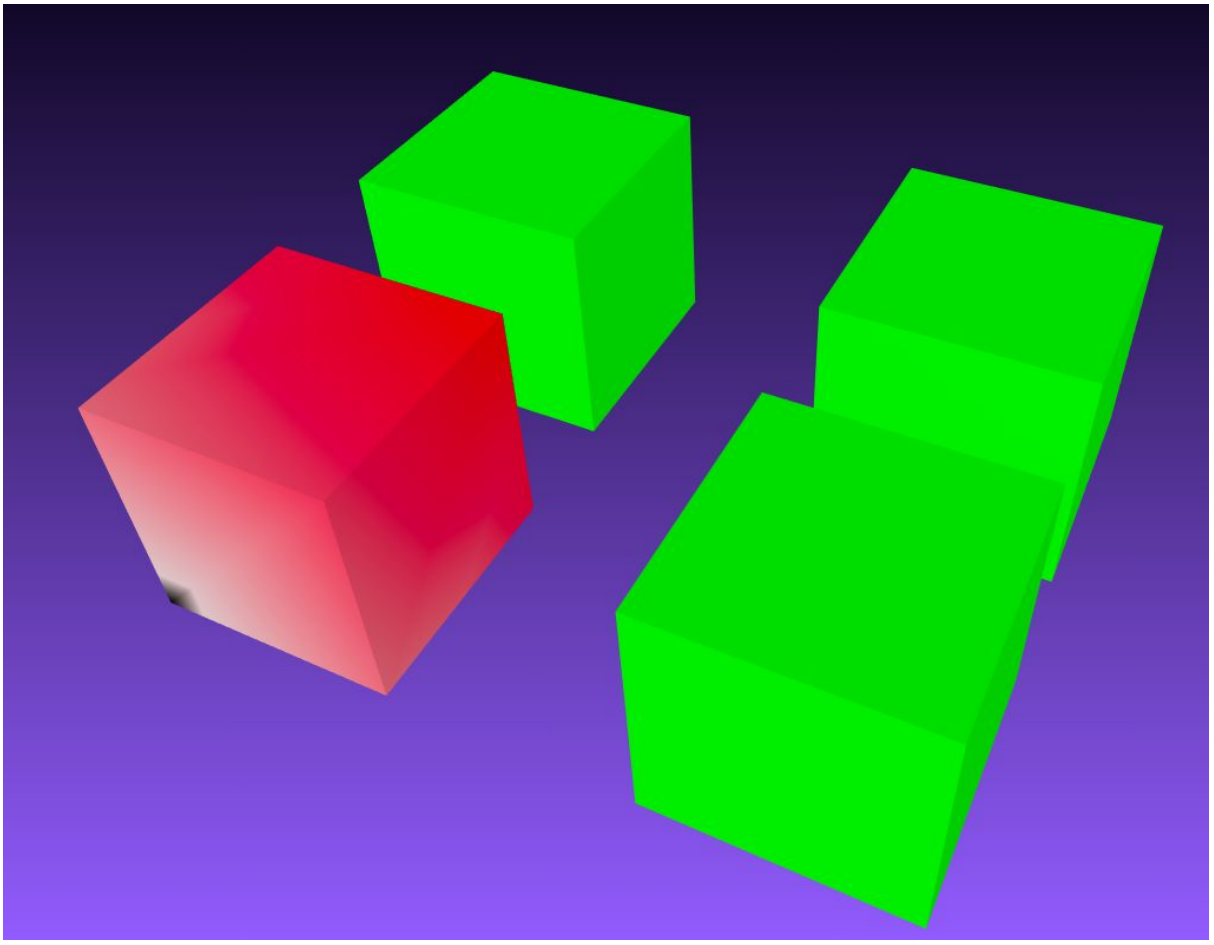
Distance géodésique :

Nous avons implémenté la distance géodésique avec un algorithme basé sur le parcours de Dijkstra et donc nous pouvons colorer les sommets du maillage en fonction de leur distance avec le sommet de départ (ici en noir).
On représente la distance par un dégradé de rouge (plus c'est rouge plus c'est loin).
Le point le plus loin est turquoise.





Les sommets qui ne sont pas parcourus, car appartiennent à d’autres composantes connexes, sont verts.



Coloration des composantes connexes :

Nous avons aussi implémenté le regroupement des sommets par composantes connexes en parcourant de voisin en voisin jusqu'à ce que la composante soit parcourue entièrement, puis on continue avec un point qui n'a pas encore été traité. Pour chaque composante connexe la couleur est choisie aléatoirement.

Complexité temps :

Nous avons implémenté un fonction qui nous affiche la complexité temps d'une fonction et de paramètres donnés

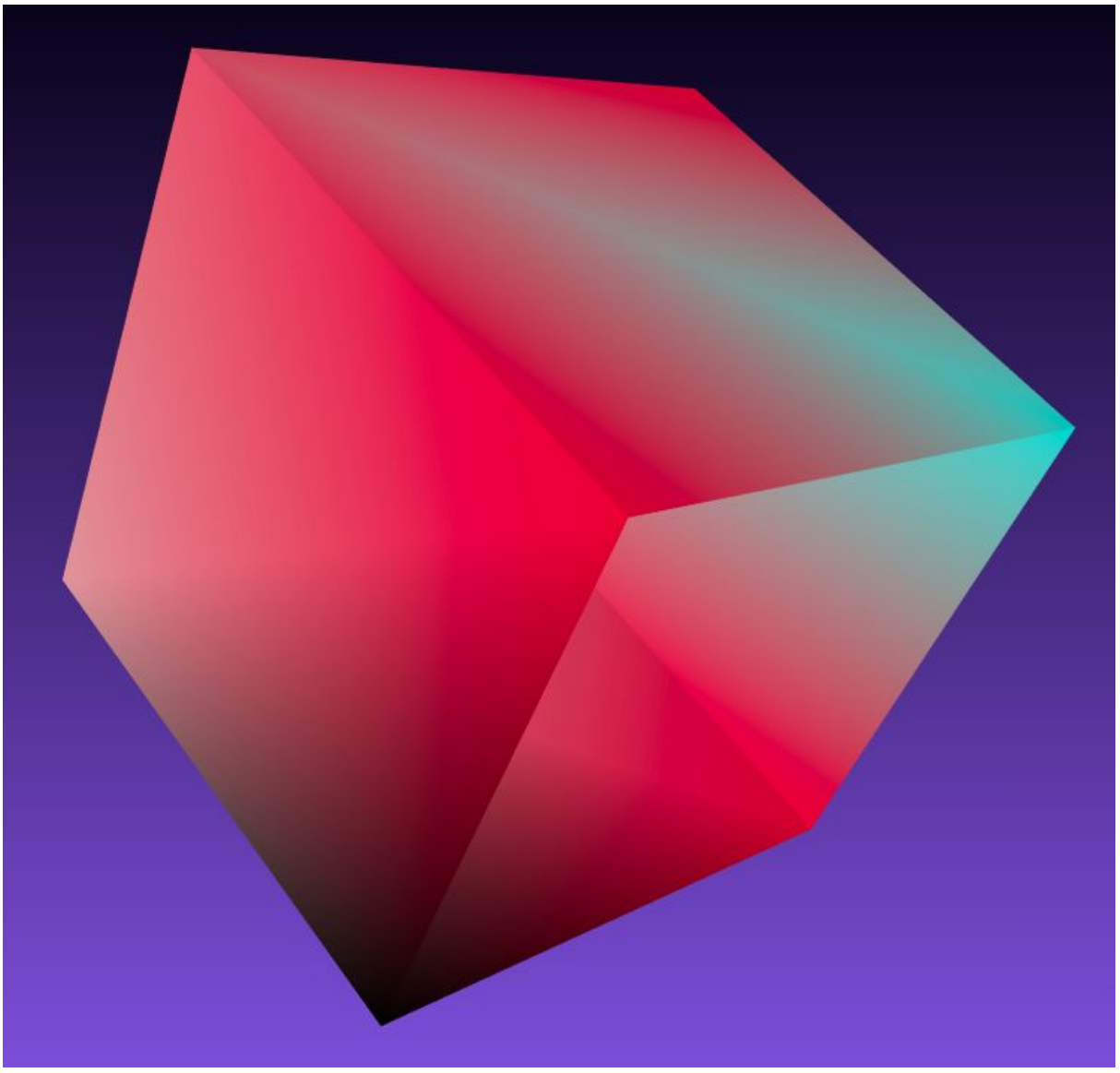
```
 La fonction
 <bound method HalfedgeMesh.geodesique of <halfedge_mesh.halfedge_mesh.HalfedgeMesh
object at 0x7efeee5d9b00>>
 met 0.18619108200073242 secondes pour s'executer

 La fonction
 <bound method HalfedgeMesh.set_composantes_connexes of <halfedge_mesh.halfedge_mesh
.HalfedgeMesh object at 0x7efeee5d9b00>>
 met 0.09722709655761719 secondes pour s'executer

 La fonction
 <bound method HalfedgeMesh.genre_by_composante of <halfedge_mesh.halfedge_mesh.Half
edgeMesh object at 0x7efeee5d9b00>>
 met 0.11127686500549316 secondes pour s'executer
```

On peut voir que pour le maillage "strange.off" qui possede 5632 sommets :

- La fonction géodésique, qui calcule les distances de tous les sommets depuis un sommet initial, prend 0.186 secondes pour s'exécuter
- La fonction set_composante_connexe, qui renvoie le nombre de composante connexe et qui affecte à chaque sommet sa composante, s'exécute en 0.097 secondes
- La fonction genre_by_composante appelle set_composante_connexe et calcule le genre de chaque composante, s'exécute en 0.111 secondes

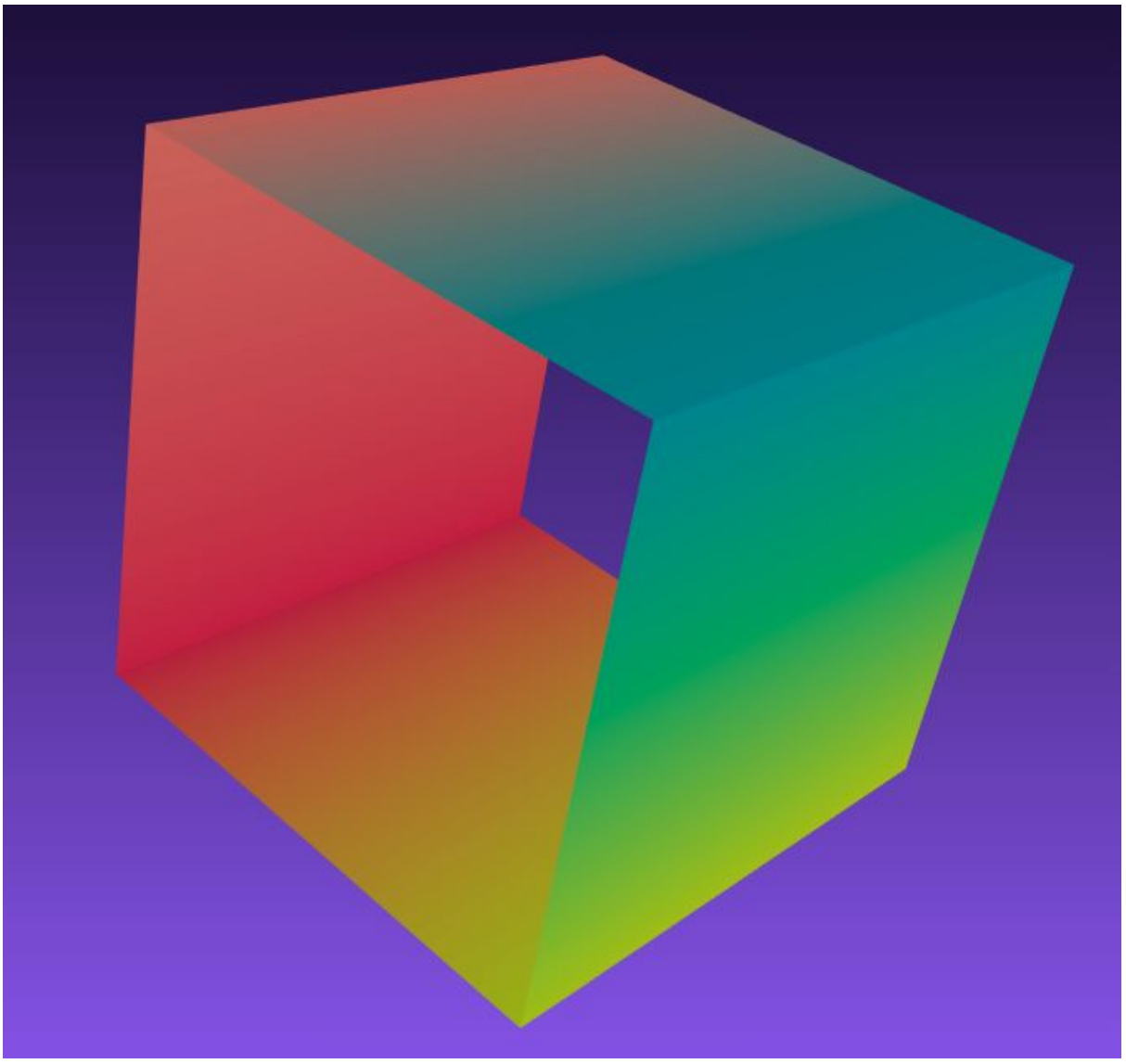
Tentative d'implémentation des bords :

On a essayé de faire en sorte que les colorations prennent en compte les bords, les colorations de composantes connexes et genre fonctionnent sauf pour certain cas précis, cube_bord2 ne fonctionne pas car chaque sommet du graphe ne possède qu'un seul sommet voisin à cause du manque de demi arête donc le parcours ne se fait pas correctement. cube_bord1 marche correctement pour ces deux colorations mais pour la mesure géodésique le parcours ne peut pas contourner le bord donc les distances ne correspondent pas à la distance réelle.



On peut le voir sur cette image le sommet noir est le sommet de départ et le sommet turquoise à droite qui est le sommet le plus éloigné d'après la fonction car le parcours passe par le "haut" du cube pour atteindre ce point et non le contour du bord.

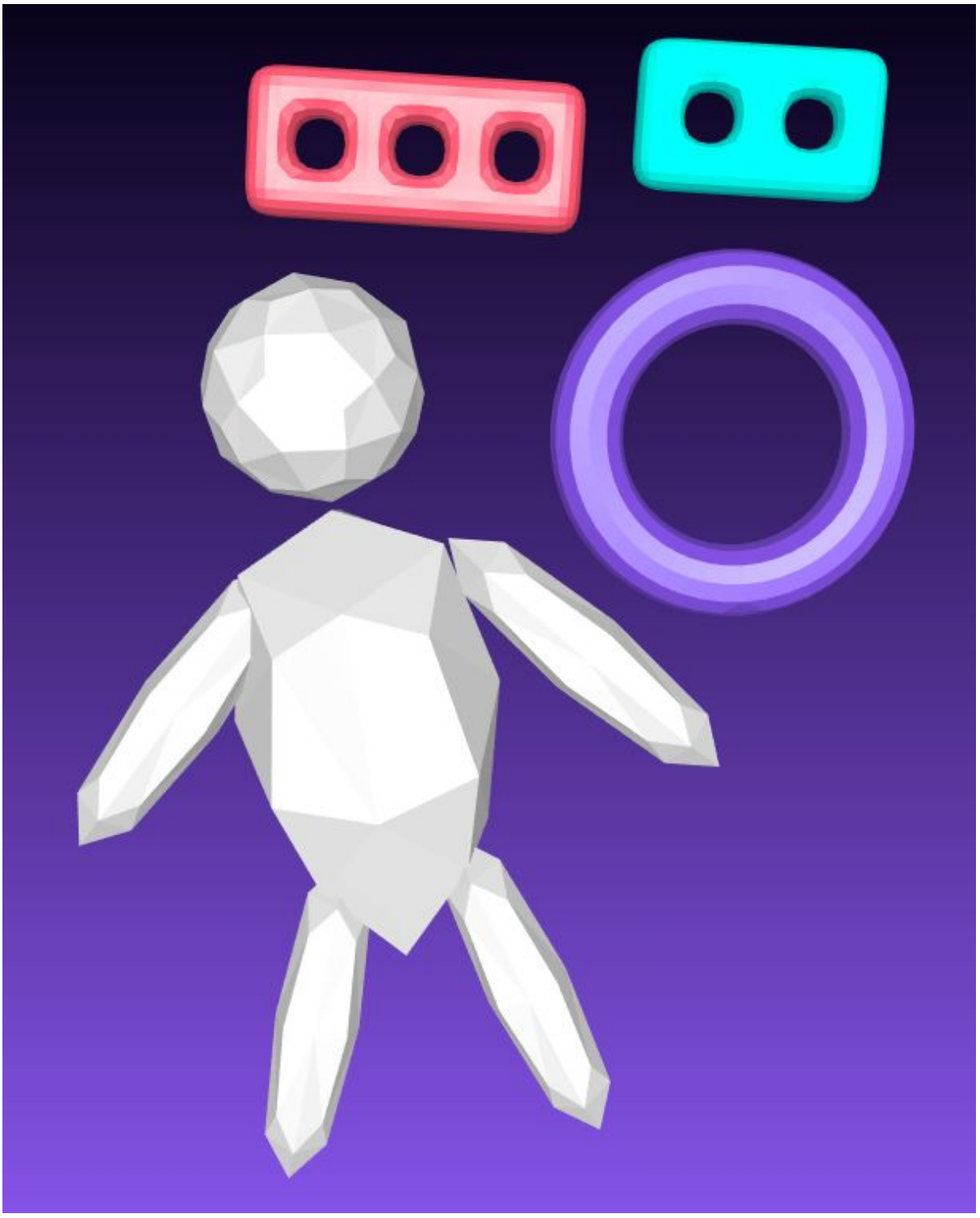
Dans le but d'améliorer la librairie nous avons modifié la limitation de 3 sommets pour une face, elle peut donc en avoir autant que possible maintenant. pour cela chaque face possède la liste des indices des sommets qui la composent.



On peut voir que le cube avec deux bords possède 4 composantes connexes chaque arête qui relie les deux bords.

```python
def adjacent_halfedges(self):

    continu = True
    first = self.halfedge
    tab = [first]
    next =  first.next.opposite

    if next != None:
        tab.append(next)
        while next != first and next != None:
            next = next.next.opposite
            if next == None:
                break
            tab.append(next)

    if first == next:
        continu = False

    if continu:
        if first.opposite != None:
            prev =  first.opposite.prev
            if prev != None:
                tab.append(prev)
                while prev != first and prev != None:

                    if prev.opposite == None:
                        break
                    prev = prev.opposite.prev
                    tab.append(prev)
    return tab
```

Voici une partie de l'implémentation des bords, on parcourt les demi-arêtes adjacentes dans les deux sens jusqu'à croiser un bord, ou dans un seul sens si on a réussi à faire le tour la première fois.



On peut voir les différentes colorations pour les genres 0, 1, 2 et 3.