Algorithmique géométrique – TP 5

Labrosse Alexandre – Henin Alexandre

Classification des triangles

Le but de ce TP est de proposer une classification(segmentation) des triangles qui composent un maillage.

Préambule

Dans un premier temps, nous décidons de baser notre segmentation suivant trois critères qui sont les suivants : le périmètre, l'aire et la somme des angles dièdres pour chacune des faces. Pour calculer le périmètre d'une face, nous utilisons les poids des demies-arêtes correspondant à la distance euclidienne séparant les deux sommets formant les demies-arêtes. Pour calculer l'aire d'un triangle à partir des coordonnées de ces sommets, nous utilisons la formule de Héron qui est la suivante :

$$S = \frac{1}{4}\sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)}$$

Figure 1 : Formule de Héron

Nous pouvons voir ci-dessous le calcul de chacunes de ces mesures :

```
for facet in facets:
    a=facet.halfedge.weight
    b=facet.halfedge.next.weight
    c=facet.halfedge.next.next.weight
    #Calcul du périmètre.
    facet.perimetre=round(a+b+c,2)
    #Calcul de l'aire.
    facet.aire=round(1/4*math.sqrt((a+b+c)*(-a+b+c)*(a-b+c)*(a+b-c)),3)
    #Calcul de la somme des angles diédraux.
    facet.sommeAngleDiedraux=facet.sommeAngleDietraux()
```

Figure 2 : Calcul des mesures de segmentation

Un angle dièdre correspond à l'angle entre deux faces. Il se calcule simplement en faisant le produit scalaire des normales des deux faces en question. La somme de ces angles pour chacune des faces nous permettra de déterminer si une surface est courbée ou plane. Pour ce faire, on prend la normale de cette face puis, pour chaque arête, on sélectionne la normale d'une face adjacente. Le produit scalaire de ces deux normales nous donne l'angle que l'on renvoie.

```
def sommeAngleDietraux(self):
    sommeAngle=0

    list_halfedge=self.get_halfedges()

    my_normal=self.get_normal()

    for i in list_halfedge:
        adjacent_normal=i.opposite.facet.get_normal()
        sommeAngle+=adjacent_normal[0]*my_normal[0]+adjacent_normal[1]*my_normal[1]+adjacent_normal[2]*my_normal[2]
    return round(sommeAngle,3)
```

Figure 3 : Calcul de la somme des angles dièdres

En fonction du périmètre

Grâce à la moyenne

Dans un premier temps, nous allons chercher à diviser les triangles en deux groupes, ceux avec un grand périmètre et ceux avec un petit. Nous définissons donc une valeur moyenne des périmètres de notre maillage. Si celui de notre face est inférieur, il sera classé comme étant un petit triangle, noté ici 0, et à l'inverse, il sera un grand triangle avec la valeur de 1.

```
def classification_faces_perimetre_moyenne(self,par_tab_classification_connexe,par_comp_connexe_en_cours):
    total=0

    for i in range(len(par_comp_connexe_en_cours)):
        total+=self.facets[list(par_comp_connexe_en_cours.keys())[i]].perimetre
    moyenne=total/len(par_comp_connexe_en_cours)

    for i in list(par_comp_connexe_en_cours.keys()) :
        if self.facets[i].perimetre<moyenne:
            par_tab_classification_connexe[i]=0
        else:
            par_tab_classification_connexe[i]=1
    return par_tab_classification_connexe
```

Figure 4 : Code représentant la segmentation d'une composante connexe en fonction de la moyenne des périmètres des faces

Cette méthode ne fonctionne que pour les maillages connexes. Pour ceux avec plusieurs composantes, nous devons répéter la classification sur chacune.

```
def classification_faces_perimetre_connexe_moyenne(self):
    tab_classification=self.parcours_largeur_composantes()[2]
    nbComposanante=max(tab_classification)+1
    comp_connexe_en_cours={}

    tab_classification_connexe={i:-1 for i in range(len(tab_classification))}

    for i in range(nbComposanante):
        for j in range(len(tab_classification)):
            if i==tab_classification[j]:
                comp_connexe_en_cours[j]=i

        tab_classification_connexe=self.classification_faces_perimetre_moyenne(tab_classification_connexe,comp_connexe_en_cours)
        comp_connexe_en_cours={}
    return tab_classification_connexe
```

Figure 5 : Code représentant la segmentation d'un maillage en fonction de la moyenne des périmètres des faces

Grâce à la médiane

Les deux fonctions précédentes utilisent la moyenne comme valeur de référence. Selon le point de vue ou la méthode, il peut être préférable de prendre la médiane des périmètres. Elle est calculée à partir d'une liste de périmètres que l'on trie et que l'on coupe en deux (ou 2+1 si le nombre de valeurs dans la liste est paire) pour obtenir la valeur du milieu.

```
def calculMediane(self,par_list_comp_connexe_en_cours):

    par_list_comp_connexe_en_cours.sort()
    taille=len(par_list_comp_connexe_en_cours)

    if(taille%2==1):
        return par_list_comp_connexe_en_cours[ceil(taille/2)]

    else:
        return (par_list_comp_connexe_en_cours[int(taille/2)]+par_list_comp_connexe_en_cours[int(taille/2+1)])/2
```

Figure 6 : Code représentant le calcul de la médiane

Une fois la médiane calculée, on procède de la même manière qu'avec les moyennes, sur les maillages à une ou plusieurs composantes connexes.

```python
def classification_faces_perimetre_mediane(self,par_tab_classification_connexe,par_comp_connexe_en_cours):
    total=0
    liste_perimetre=[self.facets[list(par_comp_connexe_en_cours.keys())[i]].perimetre for i in range(len(par_comp_connexe_en_cours))]
    mediane=self.calculMediane(liste_perimetre)

    for i in list(par_comp_connexe_en_cours.keys()) :
        if self.facets[i].perimetre<mediane:
            par_tab_classification_connexe[i]=0
        else:
            par_tab_classification_connexe[i]=1
    return par_tab_classification_connexe
```

Figure 7 : Code représentant la segmentation d'un maillage en fonction de la médiane des périmètres des faces

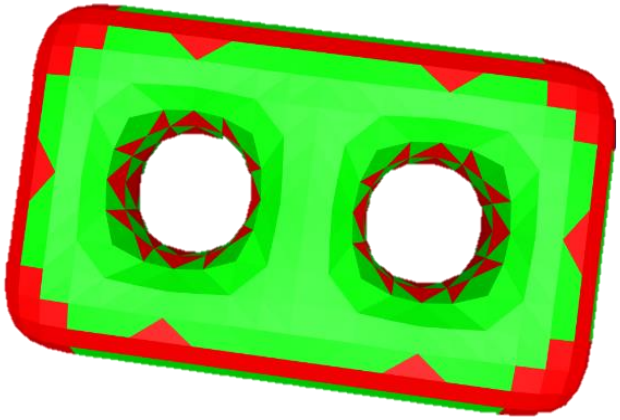

Figure 8 : segmentation de l'exemple Bitore en fonction

En appliquant cette méthode sur l'exemple Bitore, on remarque bien le liseré de « petits triangles » distingué des faces de la forme.

Grâce à la méthode d'Otsu

Enfin, on peut utiliser la méthode d'Otsu. Celle-ci, permet, à partir d'une valeur seuil, de classer nos triangles. L'objectif est donc de calculer le seuil qui sépare au mieux les deux classes, c'est-à-dire qui minimise la variance intra-classe et donc maximise la variance inter-classes.

Elle a besoin pour cela d'un histogramme des périmètres. Celui-ci est simplement défini en calculant la fréquence d'apparition de chacun des périmètres comme nous le faisons ci-dessous au travers du dictionnaire « histogramme ».

```python
def calcul_histogramme_otsu(self,par_comp_connexe_en_cours):

    liste_perimetre=[self.facets[list(par_comp_connexe_en_cours.keys())[i]].perimetre for i in range(len(par_comp_connexe_en_cours))]
    mini=0.01
    maxi=max(liste_perimetre)

    histogramme={round(i,2):0 for i in np.arange(mini,maxi+0.01,0.01)}

    for i in liste_perimetre:
        histogramme[i]+=1
    taille=len(liste_perimetre)
    for k,v in histogramme.items():
        histogramme[k]= v/taille
    return histogramme
```

Figure 9 : Code représentant le calcul de l'histogramme utilisé pour la méthode d'Otsu

Ensuite nous déterminons le seuil optimal qui sera utilisé pour la classification. Ce seuil correspond à la valeur k qui maximisera la variance interclasse. Nous calculons donc, pour chaque classe C1 et C2 possibles, la variance interclasse en appliquant la formule suivante :

$$var_{C1\,interC\,2} = \frac{(Moy \times Proba_{C1}(k) - Moy_{C1}(k))^2}{Proba_{C1}(k) \times Proba_{C2}(k)}$$

Ainsi, nous calculons le seuil optimal en conservant la valeur k de périmètre pour lequel la variance interclasse a été maximale. Nous obtenons donc le seuil permettant de segmenter nos deux classes de faces. Le code suivant permet le calcul du seuil optimal. Les commentaires du code donnent plus de détails sur l'implémentation précise.

```python
def calcul_seuil_optimal_otsu(self,histogramme,par_comp_connexe_en_cours):
    maxi=max(histogramme.keys())
    maxVariance=0
    kMaxVariance=0
    mini=0.01
    total=0

    for i in range(len(par_comp_connexe_en_cours)):
        total+=self.facets[list(par_comp_connexe_en_cours.keys())[i]].perimetre
    moyenne=total/len(par_comp_connexe_en_cours)

    for i in np.arange(mini,maxi+0.01,0.01):
        probC1=0
        probC2=0
        moyenneC1=0
        totalPerimetreC1=0
        variance_inter_classe=0

        for j in np.arange(mini,round(i,2)+0.01,0.01):
            probC1+=histogramme[round(j,2)]

        probC2=1-probC1

        moyenneC1=self.calculMoyenneSeuil(par_comp_connexe_en_cours,round(i,2))
        if(probC1!=0 and probC2!=0):
            variance_inter_classe=((moyenne*probC1-moyenneC1)**2)/(probC1*probC2)

            if variance_inter_classe>=maxVariance:
                maxVariance=variance_inter_classe
                kMaxVariance=round(i,2)
    return kMaxVariance,maxVariance
```

Figure 10 : Code représentant le calcul du seuil optimal d'Otsu

Une fois la valeur seuil définie, il ne reste plus qu'à classer les périmètres, s'ils sont inférieurs ou supérieurs au seuil. L'opération est répétée pour les maillages à plusieurs composantes connexes.

```python
def classification_faces_perimetre_otsu(self,par_tab_classification_connexe,par_comp_connexe_en_cours):
    total=0
    histogramme=self.calcul_histogramme_otsu(par_comp_connexe_en_cours)

    seuil,variance=self.calcul_seuil_optimal_otsu(histogramme,par_comp_connexe_en_cours)

    for i in list(par_comp_connexe_en_cours.keys()) :
        if self.facets[i].perimetre<=seuil:
            par_tab_classification_connexe[i]=0
        else:
            par_tab_classification_connexe[i]=1
    return par_tab_classification_connexe
```

Figure 11 : Code représentant la segmentation d'une composante connexe en utilisant la méthode d'Otsu sur les périmètres des faces

Ainsi, nous obtenons des résultats intéressants sur les maillages avec des classes identifiées. Par exemple pour la pièce de Tetris, la méthode d'Otsu sur les périmètres nous permet d'identifier les arêtes de la pièce comme nous pouvons le voir ci-dessous :

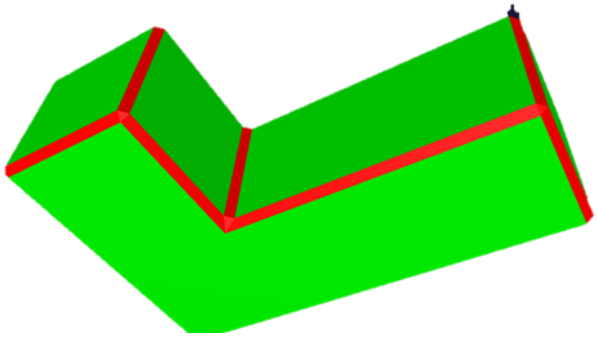

Figure 12 : segmentation de la pièce de Tetris suivant la méthode d'Otsu sur les périmètres des faces

En fonction de l'aire

En plus des périmètres, nous avions défini une aire sur chacune des faces. Cette aire peut être utilisée pour classer nos triangles. La méthode est identique aux calculs utilisant les périmètres. Nous avons donc utilisé le critère de la moyenne, de la médiane et la méthode d'Otsu pour les aires.

```
def classification_faces_aire_moyenne(self,par_tab_classification_connexe,par_comp_connexe_en_cours):
    total=0

    for i in range(len(par_comp_connexe_en_cours)):
        total+=self.facets[list(par_comp_connexe_en_cours.keys())[i]].aire
    moyenne=total/len(par_comp_connexe_en_cours)

    for i in list(par_comp_connexe_en_cours.keys()) :
        if self.facets[i].aire<=moyenne:
            par_tab_classification_connexe[i]=0
        else:
            par_tab_classification_connexe[i]=1
    return par_tab_classification_connexe
```

Figure 13 : Code représentant la segmentation d'une composante connexe en fonction de la moyenne des aires des faces

En appliquant cette classification avec l'aire moyenne, nous obtenons les résultats suivants sur les exemples Bonhomme et Bitore. Comme pour le précédent maillage, on remarque la distinction entre les gros triangles qui composent principalement les faces planes de notre figure quand les petits triangles se retrouvent dans les zones plus arrondies.

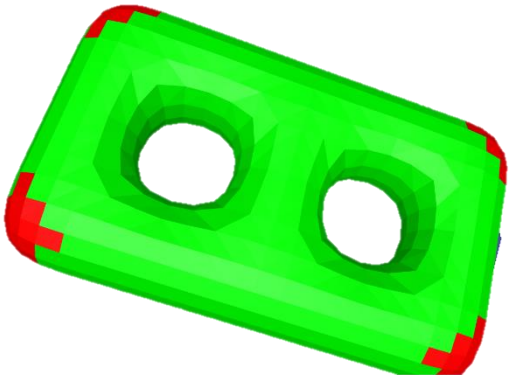

Figure 14 : segmentation du Bitore en fonction de la moyenne des aires de ces faces

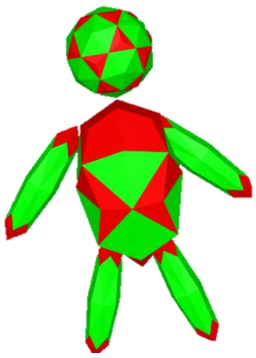

Figure 15 : segmentation du bonhomme en fonction de la moyenne des aires de ces faces

Les captures d'écran des autres segmentations sont disponibles dans le dossier Captures.

En fonction des angles dièdres

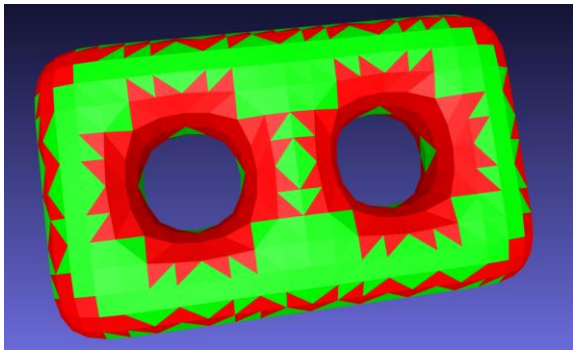

Figure 16 : segmentation du Bitore en fonction des angles dièdres via la méthode d'Otsu

Enfin, nous pouvons utiliser les angles dièdres définis précédemment. Si on applique directement sur un exemple en utilisant la méthode Otsu, on comprend plus précisément comment fonctionne cette classification.

Les zones à l'intérieur du « donut », étant courbes, elles sont représentées en rouge. Les faces planes en vert.

Nous sommes arrivés à ce résultat en remplaçant l'histogramme des périmètres ou de l'aire par celui des angles.

```
def calcul_histogramme_otsu_angleD(self,par_comp_connexe_en_cours):
    liste_angleD=[self.facets[list(par_comp_connexe_en_cours.keys())[i]].sommeAngleDiedraux for i in range(len(par_comp_connexe_en_cours))]
    mini=0
    maxi=3

    histogramme={round(i,3):0 for i in np.arange(mini,maxi+0.001,0.001)}

    for i in liste_angleD:
        histogramme[i]+=1
    taille=len(liste_angleD)
    for k,v in histogramme.items():
        histogramme[k]= v/taille
    return histogramme
```

Figure 17 : code représentant le calcul de l'histogramme pour les angles dièdres

Conclusion

Ainsi, nous avons implémenté dans le cadre de ce TP trois types de segmentation : la segmentation en fonction des périmètres, des aires et des angles dièdres. Nous avons utilisé trois critères qui sont les suivants : la moyenne, la médiane et le seuil d'Otsu.

À partir du calcul du périmètre ou celui de l'aire, on remarque que les résultats qu'ils soient fait avec la moyenne (à gauche) ou la valeur médiane (à droite) sont très similaires voir identiques.

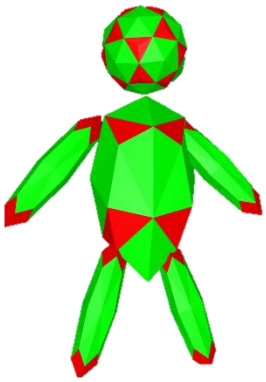

Figure 18 : segmentation du bonhomme en fonction de la moyenne des aires (ou périmètre car identique)

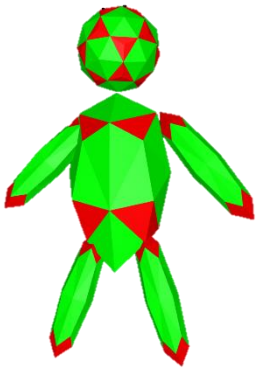

Figure 19 : segmentation du bonhomme en fonction de la médiane des aires (ou périmètre car identique)

La méthode d'Otsu permet d'obtenir des résultats très intéressants lorsqu'il s'agit de distinguer des parties clairement identifiées dans un maillage, comme c'est le cas ici avec les arêtes de la pièce de Tetris. On remarque que les résultats sont sensiblement différents si la méthode est utilisée avec l'histogramme des périmètres (gauche) ou celui des aires (droite).

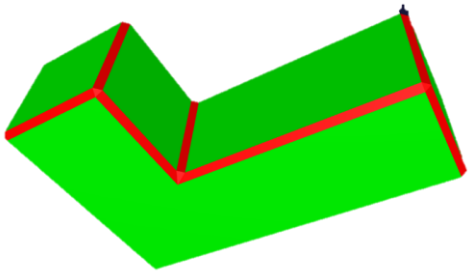

Figure 20 : segmentation de la pièce de Tetris par rapport à la méthode d'Otsu sur les périmètres

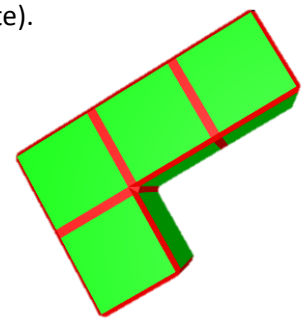

Figure 21 : segmentation de la pièce de Tetris par rapport à la méthode d'Otsu sur les aires

On pourrait également imaginer une segmentation multi-classes séparant les faces d'un maillage en plusieurs classes. Une extension de la méthode d'Otsu permet la détection de plusieurs classes. On peut également penser au partitionnement en k-moyennes…