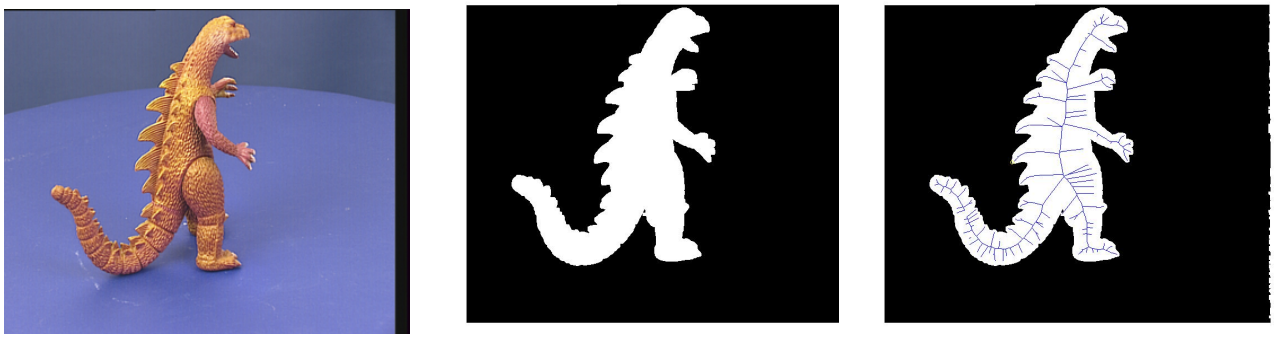
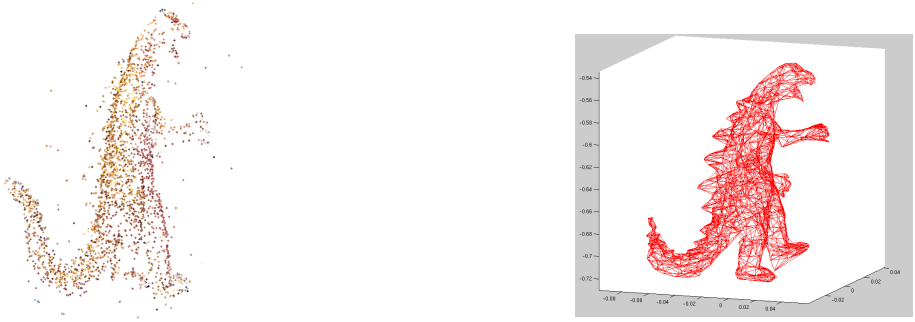
Extraction de l'axe médian d'un objet dans de multiples images de type fond/forme

Sylvie Chambon et Géraldine Morin



(a) (b) (c)



(d) (e)

Figure 1 – Un exemple d'image fournie en entrée, en (a). Nous visualisons en (b), le **résultat de segmentation attendu**, et en (c), celui de **l'extraction de l'axe médian attendu**. Puis, en (d), il s'agit de la visualisation de ce que l'on obtient lorsqu'on lance le script `TP_maillage.m` (en effectuant quelques manipulations de ce qui est affiché en 3D) et enfin, en (e), il s'agit du dernier résultat à obtenir : **le maillage surfacique de la forme**.

Objectifs

Le but de ce TP est de manipuler un ensemble d'images d'un même objet afin, d'une part, d'extraire l'axe médian, et, d'autre part, de construire le maillage surfacique de cet objet en 3D. Pour cela, nous proposons de :

(1) **Sur-segmenter** une image grâce à une technique de découpage en **superpixels** ;

(2) **Classer** simplement par un seuillage ou un critère de forme ces superpixels en régions extérieures (représentées en noir) et intérieures (représentées en blanc) à la forme ;

(3) **Appliquer un algorithme de calcul de l'axe médian** ;

(4) **Extraire un maillage surfacique de la forme**.

De plus, nous fournissons un code qui permet, d'une part, **de détecter et d'apparier des points 2D** dans les images et, d'autre part, à partir des matrices de calibrage des caméras, de construire un **nuage de points 3D** comme montré sur la figure 1.(d).

Données fournies

Pour ce sujet, nous vous fournissons une archive, cf. lien « Données à traiter » dans `moodle`, contenant un dossier `images` avec 36 images d'un dinosaure sur un fond bleu de nom `viff.0**.ppm` où `**` correspond au numéro de l'image, cf. figure 1. De plus, le dossier contient :

- `TP_maillage.m` : script à compléter permettant d'effectuer les étapes nécessaires pour retrouver le maillage ;
- `dino_Ps.mat` : matrices de projections des caméras associées à chaque prise de vue ;
- `viff.xy` : points 2D détectés et mis en correspondance dans la séquence.

Travail noté

Vous avez quatre séances pour réaliser ce sujet en **binôme**. **Les binômes sont à enregistrer dès le début de la première séance, sur** `moodle`. Au cours d'une séance dédiée, le **jeudi 6 avril à 10h15**, en **présence**, vous serez évalué·e·s sur **deux aspects** :

1. **Individuellement**, vous aurez à répondre à des questions en ligne sur `moodle`. **Il faut être présent dans votre salle de TP indiqué sur votre emploi du temps.**
2. **En binôme**, vous présenterez, sur machine, votre travail et votre code. Vous déposerez, au préalable, sur `moodle`, une archive de nom `Nom1_Nom2.zip` contenant tous les fichiers nécessaires, sauf les images, et un `LISEZMOI.txt` expliquant comment exécuter vos programmes.

1 Segmentation en superpixels (1 séance et demi)

Nous vous proposons de mettre en œuvre l'agorithme de construction de superpixels proposé par Achanta, SLIC pour *Simple Linear Iterative Clustering*, cf. l'article ainsi qu'une présentation disponible sur `moodle`. L'approche qui a été partiellement décrite en cours se résume à suivre les 3 étapes suivantes :

1. Placement régulier de germes : pour compléter, lire la présentation sur `moodle` ;
2. Calcul des superpixels avec une **approche par k-moyenne** ;

3. Renforcement de la connexité **(optionnel)**: Pour coder simplement cette étape, il faut fusionner les petites régions avec leur plus grande région voisine. Plus précisément, en supposant que les régions ont une taille moyenne de $\frac{N_p}{K}$ avec N_p le nombre de pixels de l'image et K le nombre de superpixels à construire, une petite région correspond à une région de taille inférieure à un pourcentage de cette moyenne.

Vous avez déjà implanté l'algorithme des k-moyennes au cours des 4 séances de travaux pratiques en traitement d'images. Pour ces travaux pratiques, vous avez le choix :

(1) Reprendre et adapter vos code déjà réalisés en `opencv/C++` ;

(2) Réaliser une nouvelle implémentation en `Matlab`.

Pour réaliser ce choix, il est important de prendre en compte les éléments suivants : la seconde partie est uniquement proposée en `Matlab` et les codes déjà développés doivent être modifiés pour être utilisables dans ce nouveau sujet.

Nous vous rappelons que l'algorithme des k-moyennes ou *k-means* consiste à chercher la meilleure répartition des pixels en N classes, la valeur de N étant fixée par l'utilisateur. Il s'agit d'un algorithme itératif, dont les principales étapes sont :

a. Partition de l'image en N régions, suivant un critère c à définir ;

b. Estimation des centres C_k, $k \in [1, N]$, de ces régions ;

c. Tant que les centres sont modifiés :

   i. Pour chaque pixel de l'image, recherche du centre C_k le plus proche, au sens du critère c, et affectation du pixel considéré à la classe k ;

   ii. Mise à jour des centres C_k des N régions ainsi constituées.

Il n'existe pas de preuve de convergence de cet algorithme vers l'optimum global. D'ailleurs, le résultat dépend généralement de l'initialisation.

Dans la figure 2, nous illustrons le type de résultat que nous obtenons. Nous avons les paramètres d'entrée suivants :

- K, le nombre de superpixels ;
- m, la compacité, utilisée pour calculer le poids $(\frac{m}{V})^2$ donné au terme de distance en position par rapport à la distance en couleur, pour toutes les explications complémentaires voir la formule (4.4), page 52, dans le polycopié ici.

Dans cet exemple, nous avons fait varier le nombre de superpixels utilisés, en utilisons une compacité m constante.

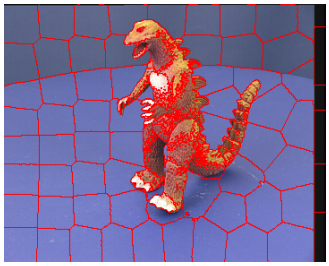
À faire

Nous vous demandons de coder cette approche. Vous pouvez utiliser la fonction `kmeans` de `matlab` mais ce n'est pas obligatoire.

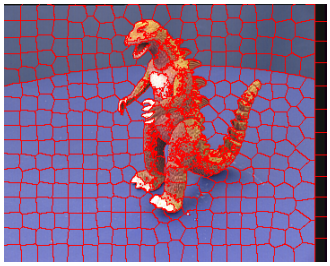
Nous vous demandons de respecter les deux points suivants :

(1) Vous devez générer l'affichage de l'image avec les germes et les superpixels (régions en couleur ou frontières entre superpixels) à chaque itération.

(2) Vous devez implanter une première version simple mais complète de SLIC. Ensuite, vous ferez évoluer votre code pour le rendre plus complet et respecter l'ensemble des éléments de SLIC.



100 superpixels



400 superpixels

FIGURE 2 – Exemple de sur-segmentations obtenues – Ici, pour une compacité m constante, nous avons fait varier le nombre de superpixels. Nous pouvons observer que les superpixels sont assez compactes. C'est-à-dire que leur forme se rapproche d'un diagramme de Voronoï. et il n'y a pas de variations brusques de la forme du contour. Cela signifie que m a une valeur assez faible qui donne peu de poids au critère de ressemblance colorimétrique. Dans cet exemple, nous favorisons donc la distance en position.

2 Segmentation binaire (1 demi-séance)

Pour classer les régions obtenues en régions extérieures ou intérieures, vous pouvez effectuer un seuillage sur la couleur des centres ou une sélection en fonction de la forme plus ou moins compacte des régions obtenues (cf. critère de compacité défini en cours, formule (4.3), page 51 du polycopié).

À faire Coder la segmentation binaire en choisissant l'approche que vous voulez.

3 Estimation de l'axe médian (1 séance)

Dans cette partie, nous vous demandons de calculer une discrétisation de l'axe médian associé à la forme binaire que vous avez obtenue.

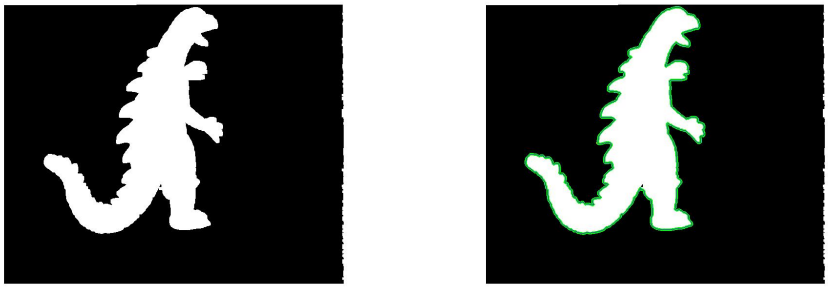
3.1 Estimation de la frontière

À faire A partir de l'image binaire, retrouvez les coordonnées dans l'image des points de la frontière de la forme, comme les points verts indiqués dans la figure 3.(b). L'ordre n'est pas important.

Vous pourrez utiliser la fonction `bwtraceboundary`. **Attent(Bonus) M.A.T et filtrageion** votre image binaire doit être *a priori* un objet blanc (1) sur un fond noir (0). De plus, vous pouvez avoir d'autres artefacts sur les segmentations utilisées donc, pensez à en tenir compte.

3.2 Estimation des points du squelette

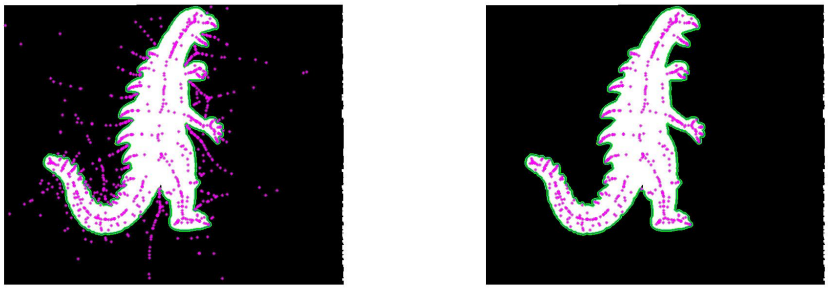
Nous avons vu en cours que les points du squelette sont donnés par les sommets du diagramme de Voronoï associé à un ensemble de points segmentant le bord, c'est-à-dire, les centres des cercles circonscrits aux triangles de la triangulation de Delaunay. Vous pourrez ajuster la densité des points du bord pour avoir un résultat correct, en gardant un temps de calcul raisonnable.



(a) (b)

FIGURE 3 – L'image binaire, (a), et la frontière trouvée (en vert), (b).

À faire Trouver les points appartenant au squelette, soit en utilisant la triangulation de Delaunay, soit avec le diagramme de Voronoi. `Matlab` propose des fonctions `delaunay` et `voronoi`, et variantes, permettant de calculer l'une ou l'autre. Pour chaque point de l'axe, on stockera l'information de rayon. On ne s'intéresse qu'à l'axe médian interne, il faut donc filtrer les points pour ne garder que ceux qui sont à l'intérieur de la forme.



(a) (b)

FIGURE 4 – Les points de l'axe médian, (a), et les points de l'axe médian interne, (b).

3.3 Estimation de la topologie du squelette

Trouver une condition pour qu'il existe une arête entre deux points du squelette.

À faire Tracer l'axe médian. Vous devez trouver un résultat ressemblant à celui de la figure 5.

Remarque: vous pouvez utiliser une matrice d'adjacence. L'axe médian connecté peut dans ce cas être tracé avec la fonction `gplot`.

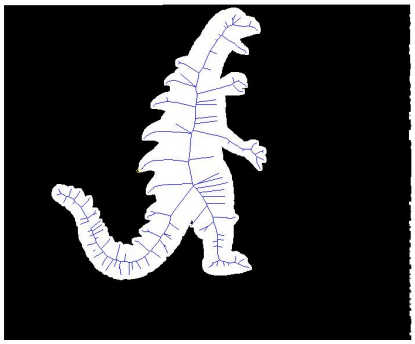


FIGURE 5 – L'axe médian interne.

3.4 (Bonus) M.A.T et filtrage

Vous pouvez dessiner les cercles autour de chaque point de l'axe médian pour vérifier que l'axe médian avec les rayons associés modélisent effectivement la forme binaire.

Vous pouvez aussi proposer une méthode de filtrage des données, pour avoir un axe médian un peu plus propre (moins de branches).

4 Extraction du maillage (1 séance)

4.1 Triangulation des points reconstruits

L'étape suivante du travail est de retrouver la tétraèdrisation de Delaunay à partir du nuage de points 3D reconstruits. Pour cela, on utilisera la fonction `DelaunayTri` de `matlab`.

Vérification On affichera ensuite le résultat trouvé qui doit être similaire à celui vu dans la figure 6.

4.2 Élimination des tétraèdres superflus

Comme vous pouvez le remarquer, la tétraèdrisation de Delaunay décrit l'enveloppe convexe des points 3D mais tous les tétraèdres n'appartiennent pas à l'objet 3D. Nous allons maintenant éliminer les tétraèdres qui sont hors de l'objet (parties concaves ou points aberrants). Pour cela, on vérifie que chaque tétraèdre inclus dans l'objet 3D se projette à l'intérieur de l'image d'un objet. La figure 7

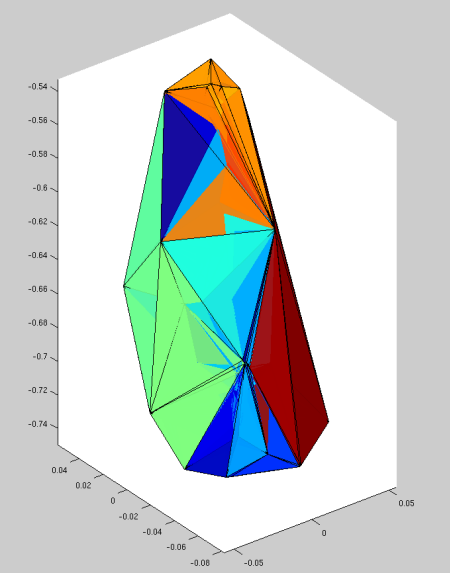


FIGURE 6 – Tétraèdrisation de Delaunay.

illustre ce propos. Le tétraèdre dont la projection est le triangle vert doit être gardé alors que celui dont la projection est le triangle rouge doit être retiré.

FIGURE 7 – En vert, la projection d'un tétraèdre appartenant à l'objet 3D. En rouge, la projection d'un tétraèdre appartenant à l'enveloppe convexe de l'objet 3D mais n'appartenant pas à cet objet 3D.

Pour chaque image i, on a à estimer le masque binaire, cf. section 2. Vous devez utiliser ces masques à présent. point 3D $\begin{pmatrix} X & Y & Z & 1 \end{pmatrix}^T$ écrit en coordonnées homogènes se projette dans la *ième* image en un point 2D $\begin{pmatrix} u & v & 1 \end{pmatrix}^T$ via la matrice de projection $P\{i\}$ qui vous est fournie :

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} wu \\ wv \\ w \end{pmatrix} = P\{i\} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Lors de la projection d'un point 3D, on trouve le point 2D à un facteur d'échelle près. Il est donc nécessaire de normaliser le point trouvé par la troisième coordonnée. Pour retirer les tétraèdres, nous proposons de suivre les étapes suivantes, **pour chaque tétraèdre issu de la triangulation de Delaunay** :

(1) Calculer le barycentre de ses 4 sommets (on pourra utiliser plusieurs barycentres avec différents poids de manière à ce que le barycentre se rapproche successivement de chaque sommet) ;

(2) Projeter chaque barycentre dans les 36 masques ;

(3) Si un barycentre ne se projette pas dans la région du dinosaure dans au moins une des 36 images (`im_mask` vaut 1), retirer le tétraèdre. **Attention !** il est nécessaire de vérifier que le barycentre est bien projeté dans l'image avant de tester si `im_mask` vaut 1.

Vérification (Bonus) M.A.T et filtrage Vous devriez retrouver un résultat similaire à celui présenté dans la figure 8.

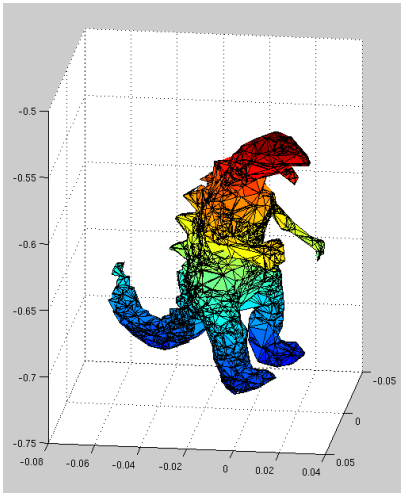


FIGURE 8 – Objet 3D composé de plusieurs tétraèdres.

Conseil une fois cette étape terminée, on pourra sauvegarder les variables d'environnement grâce à la fonction `save` de `matlab`. Elles pourront ensuite être chargées grâce à la fonction `load` (en utilisant un deuxième script `TP_maillage_suite`), ce qui évitera de relancer tous les calculs à chaque fois.

4.3 Maillage

Cette étape consiste à retrouver le maillage 3D, c'est-à-dire retrouver les triangles/faces sur la surface de l'objet. Pour cela, il est nécessaire de parcourir les tétraèdres issus de l'étape précédente et de ne garder une face que si elle n'appartient qu'à **un seul tétraèdre** (elle est alors nécessairement à la surface de l'objet, les autres faces/triangles, elles/eux, appartiennent automatiquement à deux tétraèdres).

Une façon de procéder consiste à :

(1) Construire une matrice de taille $N_f \times 3$ avec les N_f faces (4 par tétraèdre) ;

(2) Trier dans l'ordre croissant de son premier sommet, puis son deuxième et enfin de son troisième en utilisant la fonction `sortrows` de `matlab` ;

(3) Parcourir cette matrice en supprimant les faces en double qui sont obligatoirement l'une à la suite de l'autre si le tri a été bien fait.

Cet algorithme est en $O(n \log(n))$ (à cause du tri) ce qui nous permettra d'avoir un temps de calcul plus raisonnable par rapport à une implémentation naïve quadratique.

Vérification Un résultat est montré dans la figure 9.

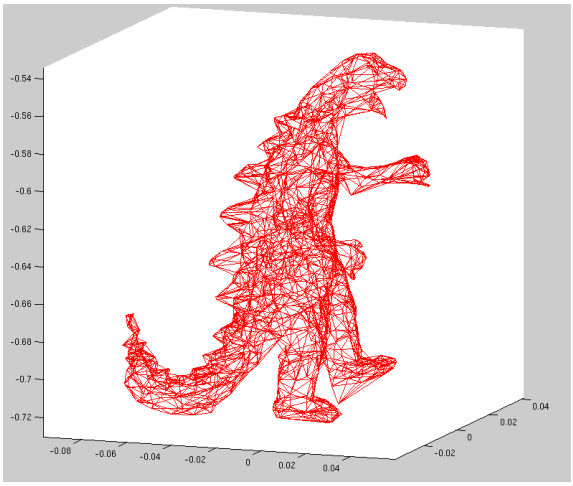


FIGURE 9 – Maillage final de l'objet en 3D.