HMIN318M: Imagerie médicale et 3D

Rendu TP : Segmentation (2)

Tianning MA

M2 IMAGINA

24/11/2020

Table de matière

A. Introduction

Ce compte rendu est dédié TP sur la deuxième partie de segmentation du cour d'Imagerie médicale 3D (HMIN318M). Le travail est basé sur la base de code du TP0.

Toutes les questions sont répondues. Vous trouverez également l'ensemble de mon code source : https://github.com/matianning/ImagerieMedicale3D/tree/main/TP%20Segmentation-2

B. Interaction possible

Touche	Fonctionnement		
Molette de la souris	Changement de couches	M	Filtrage simple (blur)
+	Threshold(seuil) +=5	Left Click	Séléctionner le voxel « graine »
-	Threshold(seuil) -=5		

C. Rendu et explication des exercices

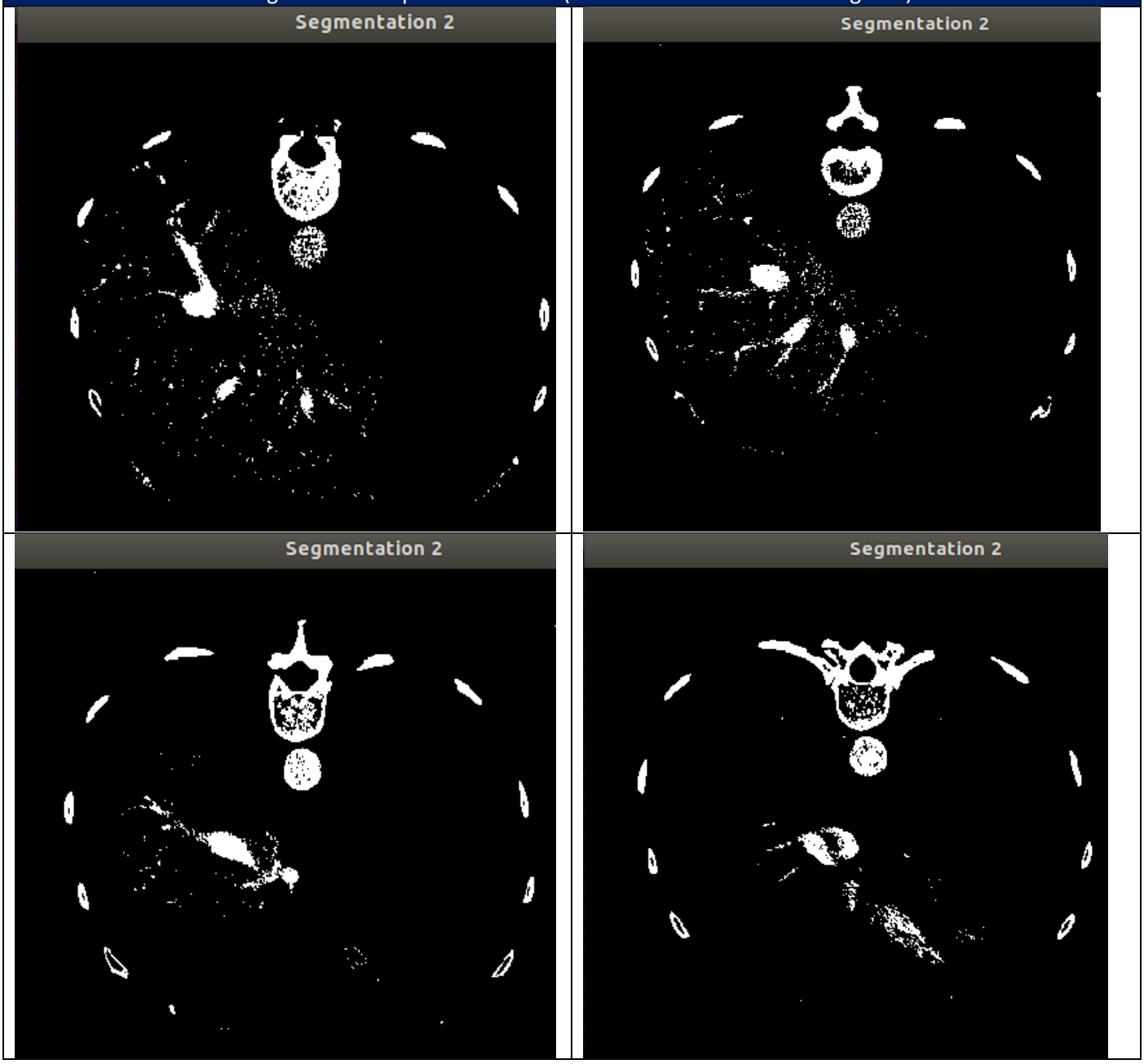
1. Visualisation d'image

Etudier l'image liver. Essayer de trouver empiriquement le seuil qui permet de segmenter au mieux le(s) réseau(x) vasculaire(s). Visualiser le résultat. Que se passe-t-il si on prend un seuil supérieur ou inférieur ?

L'image ci-contre est l'image originale.

Seuil choisi pour segmenter le réseau vasculaire : seuil = 200

Images obtenues pour seuil = 200 (couches différentes sur l'image 3D)



On peut constater sur ces images, le réseau vasculaire sur le foie. Néanmoins, il y a quand même du bruit dessus (les voxels blancs isolés).

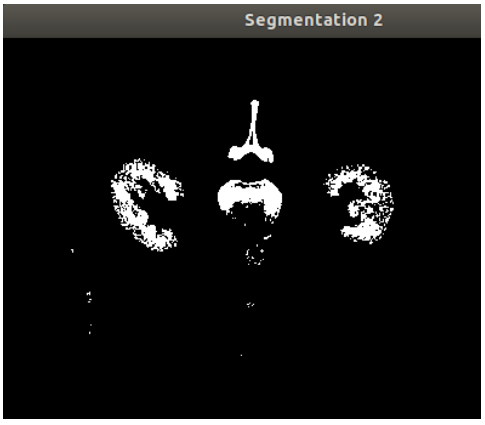


Figure 1 Image seuillée avec seuil = 220

Si on prend un seuil supérieur, on s'aperçoit que le réseau vasculaire (l'objet que l'on veut) sera filtré aussi. Donc, le réseau vasculaire ne sera plus visible sur l'image seuillée.

Si on prend un seuil inférieur, on voit sur l'image que non seulement le réseau vasculaire mais aussi par exemple la surface de foie. Donc, à cause de cela, le réseau vasculaire devient moins visible sur cette Image 3D.

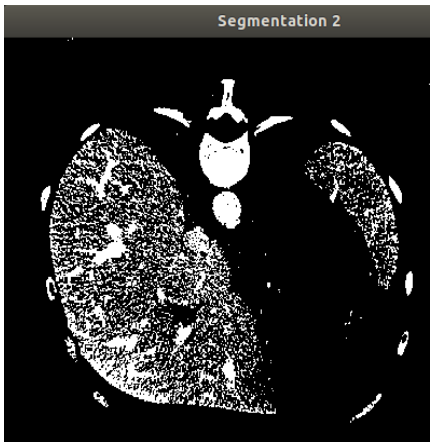


Figure 2 Image seuillée avec seuil = 180

2. Algorithme de segmentation du réseau vasculaire (selon l'article)

Selon l'article *« Analysis of Vasculature for Liver Surgical Planning »*, l'algrorithme de segmentation du réseau vasculaire (*Fast and Robust Vessel Segmentation*) nécessite tout d'abord des prétraitements d'image, puis une méthode de croissance par région pour identifier et dessiner le réseau vasculaire dans le foie.

- Phase de prétraitement :

les fonctions de filtre pourraient être utilisées pour la réduction de bruit

Le filtre Laplacien pour la compensation du fond.

- Méthode de croissance par région

Pour poursuivre cette méthode, l'utilisateur doit sélectionner sur l'image, la veine porte à l'intéreur du foie et ce voxel choisi est considéré comme « graine ». Commencer par cette graine, l'algorithme va itérativement accumuler les 26 voxels voisins qui ont l'intensité supérieur ou égale à celle de graine, et sauvegarder dans une liste. Ensuite, l'algorithme va considérer cette liste comme nouvelle graine pour continuer à itérer jusqu'à le nombre max de voxel désiré (une valeur fixée). Pour finir, on dessine ces voxels dans l'image.

3. Implémentation (partielle) de l'aglorithme

Diffultés rencontrées :

- Problème avec la fonction draw_fill(...)

  J'ai beaucoup cherché sur Internet par rapport à l'utilisation de cette fonction, mais malheuresment, je n'ai toujours pas réussi à comprendre et comment le faire fonctionner.

  Donc pour dessiner sur l'image, j'ai utilisé la fonction draw_circle, qui consiste à dessiner un circle (rayon = 1) dans le voxel cliqué et sur la couche (z) courante. J'ai aussi testé pour changer l'intensité du voxel ciblé à 255 par exemple, mais cela a presque pareil que la fonction draw_circle.

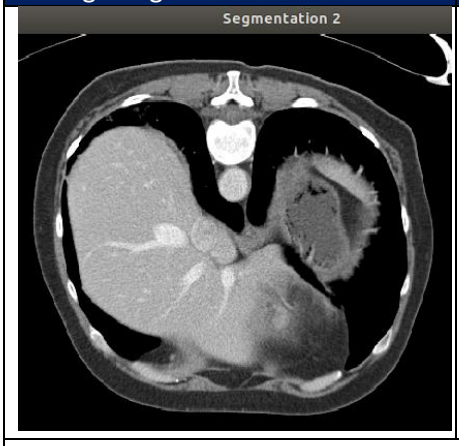
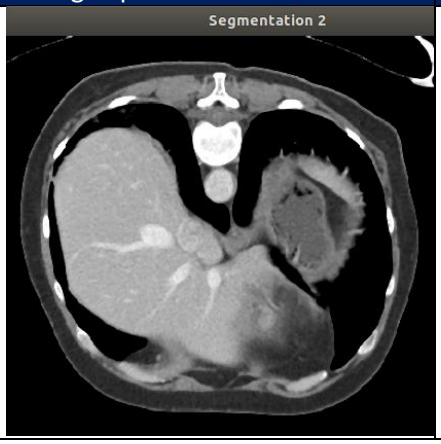
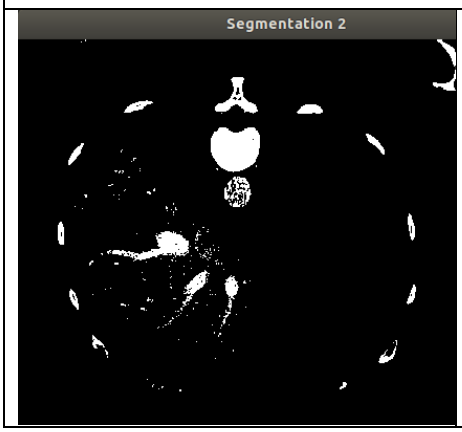
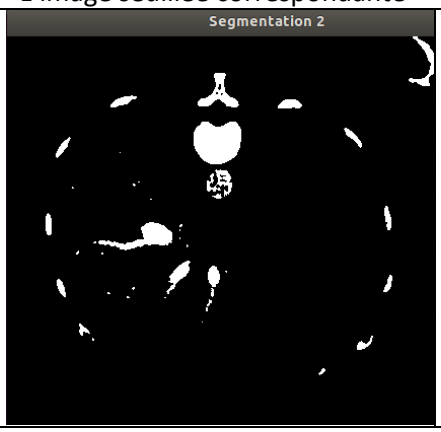
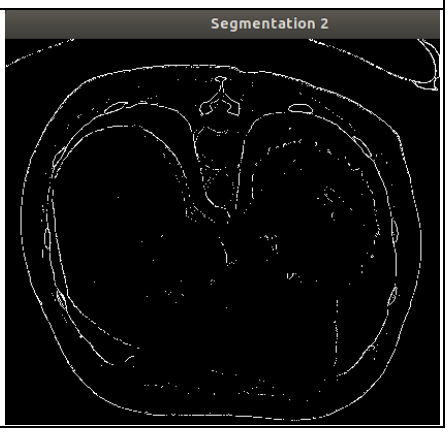
- La redondance de traitement.

  En effet, le problème principal est la redondance de traitement. Comment éviter de traiter un voxel déjà traité est important. Cela pourra avoir un impact important sur l'exécution. (le temps d'exécution sera très important)

4. Testes sur Les images

Prétraitement :

Le prétraitement de l'aglorithme consiste à réduire les bruits et compenser le fond. Pour cela, j'ai utilisé le filtre médian et un filtre laplacian.

L’image originale	L’image après le filtre médian	L’image après le filtre laplacian
		
L’image seuillée correspondante		
		

Comme montrée les images ci-dessous, on constate que après un filtre médian, le bruit est bien diminué, surtout sur l’image seuillée, on voit que les voxels isolés sont beaucoup moins. Et le réseau vasculaire semble plus évident que celle de l’image originale. Donc la réduction de bruit est bien réalisée.

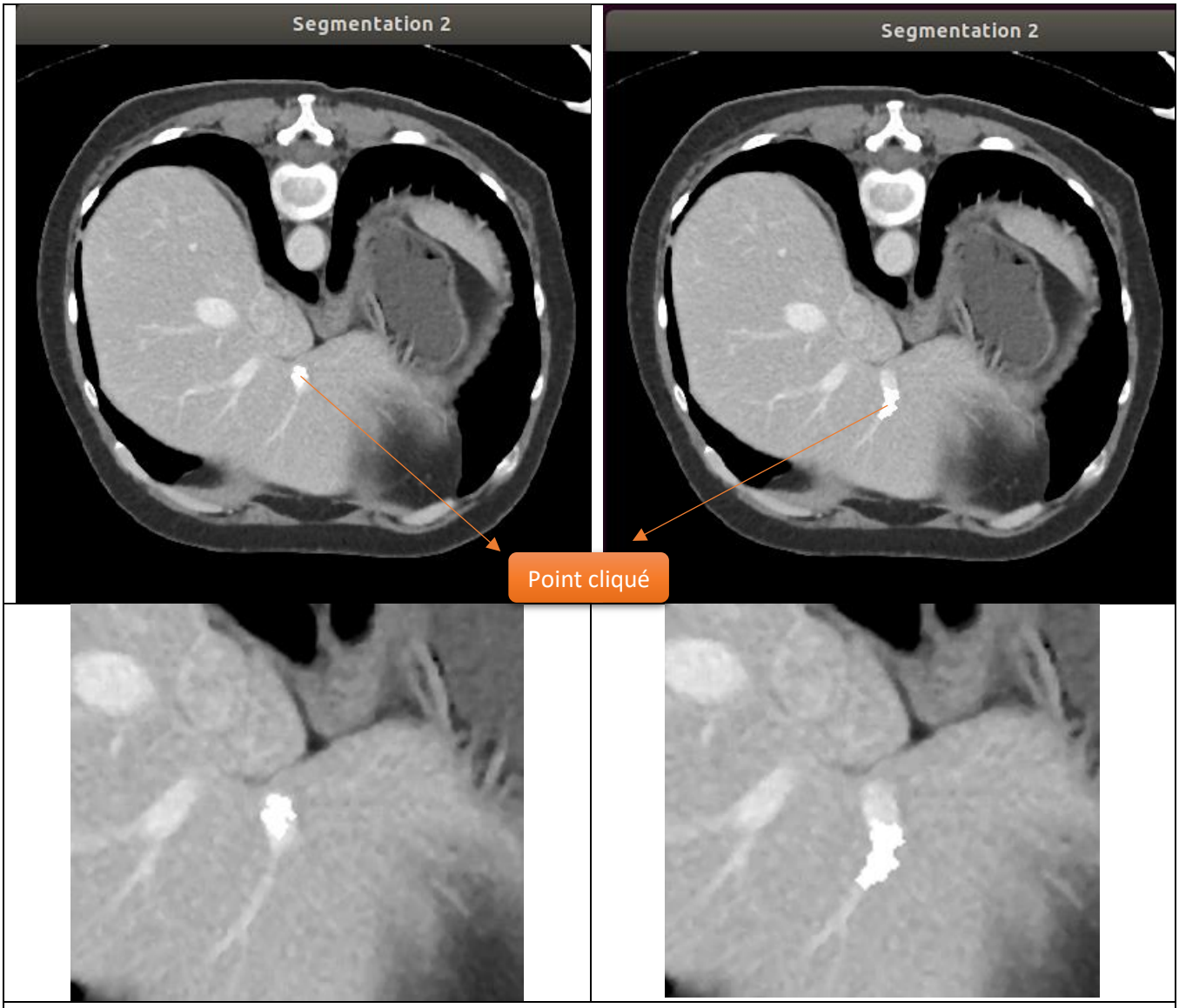
0	-1	0
-1	4	-1
0	-1	0

(Filtre Laplacian utilisé)

Par contre, après un filtre laplacien, on voit que le contour est bien accentué. Mais le réseau vasculaire dans le foie n’est plus aussi visible. Donc, je ne suis pas convaincu de cette opération ici. (ou peut-être j’ai mal utilisé dans mon cas). De ce fait, pour la suite, j’utiliserai seulement le filtre médian comme prétraitment de mon algorithme.

Croissance par région :

L’algorithme que j’ai implémenté est pour but de dessiner (fill) le réseau vasculaire au mieux (plus visible) après que l’utilisateur aurait choisi la veine porte à l’intérieur du foie par un simple click sur l’image. La suite d’algorithme consiste à utiliser le principe de croissance par région. A partir un voxel graine, on regardera ses voisins, si ses voisins ont l’intesité supérieure ou égale à celle de graine, on les ajoutera dans la liste de graine et continuer la itération suivante (jusqu’à maxItération). Donc, théoriquement, la région d’intérêt va étendre à partir de voxel « graine » et afficher tout le réseau vasculaire.



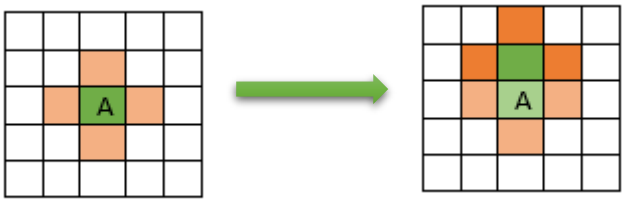
On voit que la forme de remplissage suit bien la forme du réseau vasculaire, mais cela ne remplit pas tous le réseau. C'est parce qu'ici, j'ai mis le nombre d'itération = 10. Le nombre d'itération est l'itération pour étendre « le voxel graine », selon l'intensité et celle de ses voisins.

Néanmoins, en faisant la méthode de cette expension à partir de voxel graine, le program rame beaucoup, et je pense mon program est très couteux en terme de temps. En effet, pour chaque frame, à partir de la liste de graine, il faut regarder tous ses voisins puis comparer et mettre à jours la liste, cela est très couteux.

5. Méthode d'optimisation

Selon l'implémentation ci-dessus, on s'aperçoit que le program pour remplir et étendre la région intérêt est très couteux en terme de temps et calcul. Le problème principal est de traitement répétitif sur certaines voxels. En effet, par exemple (en 2D ici), si on veut étendre le voxel graine « A », on a besoin de regarder ses voisins, mais il faut faire attention à ne pas ajouter le voxel dessus le voxel A, sinon ce serait redondant (on va avoir un traitement sur un voxel déjà traité). Et ça augmente considérablement la complexité et le temps d'exécution.

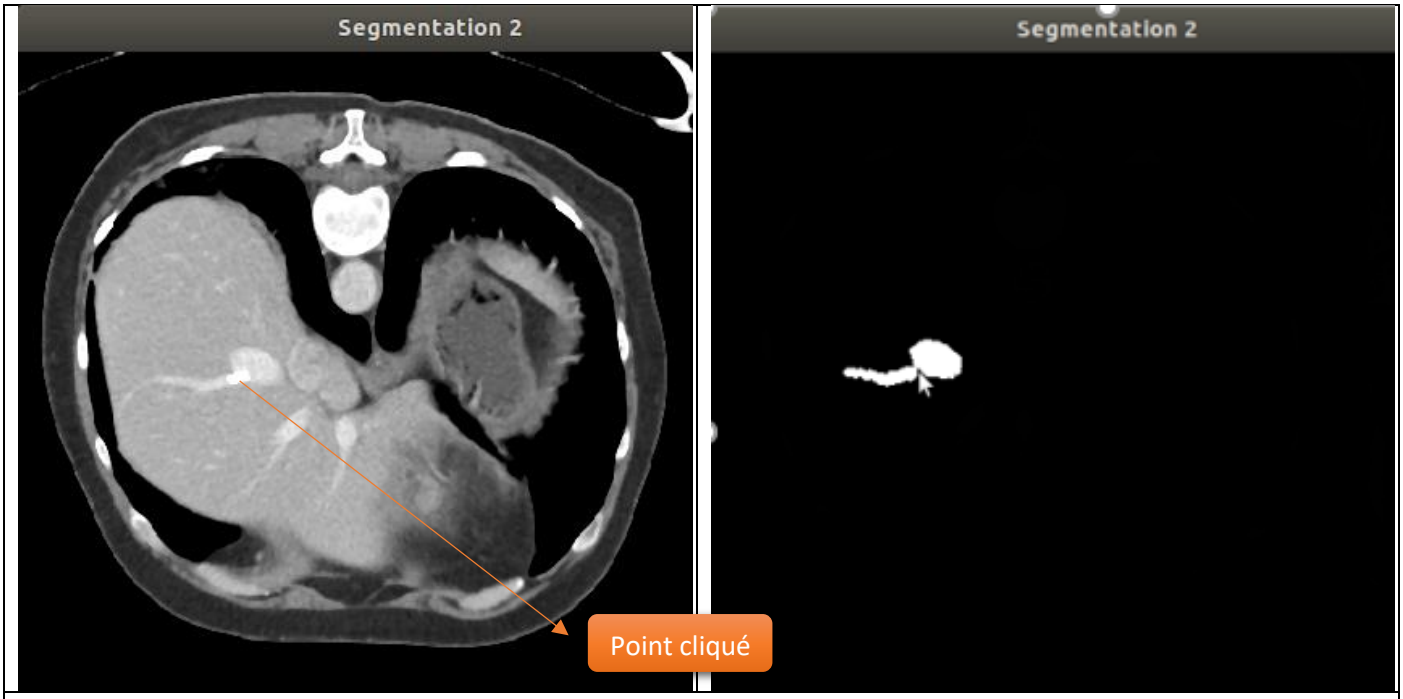
C'est pour cela, mon implémentation n'a pas réussi à remplir tous le réseau que l'on veut mais aussi limité à une partie de réseau vasculaire (à cause de la limite de nombre d'itération).



Donc, après la réflexion, je pense à au lieu de étendre chaque voxel en comparant avec ses voisins, on pourra se servir directement la forme connexe via la labélisation. Le principe est au lieu de étendre la graine, on regarde directement sur les parties qui ont le même label que le voxel sélectionné.

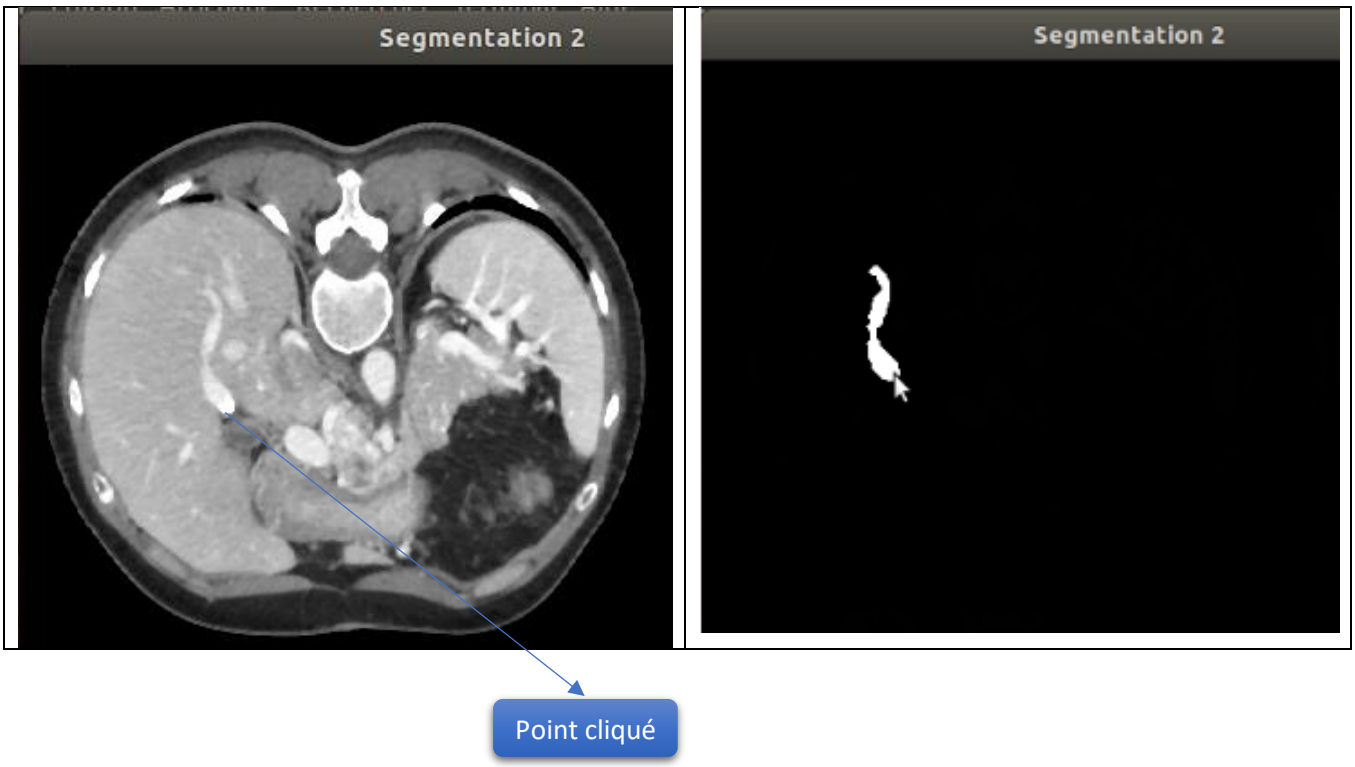
Problème :

- La sélection du voxel de départ a besoin un peu de réflexion. La croissance de région est pour étendre la région d'intérêt selon l'intensité, mais celle avec la labélisation étendra la région selon la connexité. Mais les objets pourraient se connectent, mais il se peut que l'intensité d'un objet connecté n'est pas la partie que l'on veut. Cela est problématique.
- Le remplissage est limité à la partie qui se connecte avec le voxel graine. Par exemple pour le réseau vasculaire, seul la partie de réseau qui connecte avec le voxel sélectionné sera affichée. Donc l'affichage n'est toujours pas complète.



On constate que seulement les voxels qui connectent avec le voxel « graine » seront affiché. Donc le réseau vasculaire n'est pas complètement apparaît.

Testes sur l'image *BREBIX.liver.norm.hdr :*



Annexe :

Vous trouverez l'intégralité de mon code dans mon espace git (lien écrit dans l'introduction de mon rapport)

Intéraction left Click pour choisir le voxel graine

```
/* Left Click pour choisir la position de graine*/
if (disp.button()&1  && (plane!=-1))
{

    pos_x = disp.mouse_x();
    pos_y = disp.mouse_y();
    pos_z = displayedSlice[plane];
    //std::cout<<"Graine choisie : ["<<pos_x<<","<<pos_y<<","<<pos_z<<"]"<<std::endl;

    redraw = true;
    seuiller = true;
}
```

Prétraitement :

```
CImg<> mpr_img=img.get_projections2d(displayedSlice[0],displayedSlice[1],displayedSlice[2]);
mpr_img.resize(512,512);



//******* Prétraitement *******

//******* Filtre médian --- Réduction bruit ******
mpr_img = mpr_img.get_blur_median(3.0);

//******* - Opération - Erosion *******
/*
mpr_img = mpr_img.erode(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);
mpr_img = mpr_img.dilate(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);
*/

//******** Filtre laplacien --- compensation du fond ********
CImg<> filtre_laplacien = CImg<>::matrix(0,-1,0,-1,4,-1,0,-1,0);
//mpr_img.convolve(filtre_laplacien);
CImg<> result = mpr_img;
```

Croissance par région :

```
    if(pos_x!=0 || pos_y!=0){
        point graine;
        graine.x = pos_x; graine.y = pos_y; graine.z = pos_z;
        graines.clear(); graines.push_back(graine);

        int compteur = 0;
        //***Croissance par région***
        for(int i = 0; i < maxIter; i++){
            int current_size = graines.size();
            for(int j = compteur; j < current_size; j++){
                compteur++;
                int current_x = graines[j].x;
                int current_y = graines[j].y;
                int current_z = graines[j].z;
                int current_value = mpr_img(current_x, current_y);

                if(current_x - 1 >= 0){
                    if(mpr_img(current_x - 1, current_y) >= current_value -tolerance){
                        point newGraine; newGraine.x = current_x - 1; newGraine.y = current_y; newGraine.z = current_z;
                        graines.push_back(newGraine);
                    }
                }
                if(mpr_img(current_x + 1, current_y) >= current_value-tolerance){
                    point newGraine; newGraine.x = current_x + 1; newGraine.y = current_y; newGraine.z = current_z;
                    graines.push_back(newGraine);
                }
                if(current_y - 1 >= 0){
                    if(mpr_img(current_x, current_y - 1) >= current_value -tolerance){
                        point newGraine; newGraine.x = current_x; newGraine.y = current_y - 1; newGraine.z = current_z;
                        graines.push_back(newGraine);
                    }
                }
                if(mpr_img(current_x, current_y + 1) >= current_value-tolerance){
                    point newGraine; newGraine.x = current_x; newGraine.y = current_y + 1; newGraine.z = current_z;
                    graines.push_back(newGraine);
                }

            }

            //std::sort(graines.begin(), graines.end());
            //graines.erase(std::unique(graines.begin(), graines.end()), graines.end());

        }
    }
```

```
//*****dessiner dans l'image de sortie*******
for(int i = 0; i < graines.size(); i++){
    result.draw_circle(graines[i].x, graines[i].y, 1, color);
}

//*****Seuiller l'image de sortie********
if(seuiller){
    result = result.get_threshold(seuil);
    CImg<> labels = result.get_label(false);
    for(size_t x = 0; x < labels.width(); x++){
        for(size_t y = 0; y < labels.height(); y++){
            for(size_t z = 0; z < labels.depth(); z++){
                if(labels.atXYZ(x,y,z) == labels.atXYZ(pos_x, pos_y, pos_z)){   //même label que la graine
                    result.draw_circle(x, y, 1, color);
                }
            }
        }
    }
}

disp.display(result.abs().normalize(0,255));

redraw=false;
```

Remarque :

Car ici j'ai utilisé la fonction draw_circle au lieu de draw_fill, et la fonction prends seulement les coordonnées de x et y. Donc j'ai fait la croissance par région seulement par x et y. Normalement, la meillure façon de faire est de faire la croissance par x, y et z (3 axes) avec la fonction draw_fill, donc le code concernant la croissance de région changera. Non seulement il faut comparer ses 4 voisins, mais 26 voisins pour chaque voxel graine.