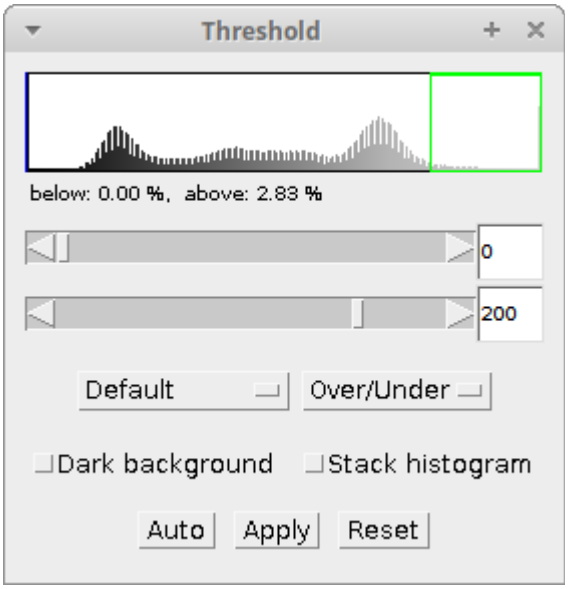
Compte-rendu TP2

Question 1

Le seuil qui permet le mieux de segmenter les réseaux vasculaires pour l'image *liver* me semble être aux alentours de 200, soit 97,18 %, si on est en black & white avec l'outil Fiji.



Si on choisit un seul supérieur on n'a plus autant de précision, on n'arrivera pas à distinguer, à bien segmenter les réseaux vasculaires du foie car on en manquera quelques réseaux importants.
Si, au contraire, on choisit un seuil inférieur, on prendra en compte trop d'informations pour la segmentation, cette dernière ne sera alors pas précise du tout.

Question 2

Algorithme :

- On sélectionne un voxel *source* correspondant à la partie qu'on souhaite segmenter.
- On analyse les 26 voxels adjacents au voxel sélectionné en regardant à chaque fois les voisins de ces derniers.

- Si l'intensité des voxels voisins est supérieure ou égale à celle du voxel *source*, alors on stocke ces derniers dans une liste et on répète le processus jusqu'à ce qu'on n'ait plus de voxels voisins avec une intensité supérieure ou égale à celle du voxel *source*.

Question 3

```
typedef struct voxel
{
    int x;
    int y;
    int z;
} Voxel;

// Supprime les voxels d'intensite inferieure pour la fonction draw_fill()
CImg<> delete_below_intensity(CImg<> img, int intensity){
    for(int i=0; i<img.width()*img.height()*img.depth(); i++){
        if(img[i] < intensity){
            img[i] = 0;
        }
    }
    return img;
}

// Extrait la region en fonction d'un voxel source
CImg<> extract_region(CImg<> img, Voxel voxel, int opacity, int tolerance){
    printf("Extracting given region with a tolerance of %d\n", tolerance);
    unsigned char color[] = { 255, 255, 255 };
    CImg<> region;
    img.draw_fill(voxel.x, voxel.y, voxel.z, color, opacity, region, tolerance);
    return region;
}

// Recupere le nombre de voxels d'une region
int get_voxel_amount(CImg<> region){
    int counter = 0;
    for(int i=0; i < region.width()*region.height()*region.depth(); i++) {
        if(region[i] != 0)
            counter++;
    }
    return counter;
}

// Construit l'image utilise par le graphe
CImg<> compute_graph(CImg<> img, Voxel voxel, int opacity, int nbOfSamples){
    CImg<> graph_img(1,nbOfSamples,1,1,0);
    CImg<> tmp;
    for(int i=0; i<graph_img.height(); i++){
        tmp = extract_region(img, voxel, opacity, i);
        graph_img(0,i) = get_voxel_amount(tmp);
    }
    return graph_img;
}

int get_max(CImg<> img){
    int max = 0;
    for(int i=0; i<img.height(); i++){
        if(img[i] > max)
            max = img[i];
    }
    return max;
}
```

```
int main(int argc, char** argv){

    if(argc != 3){
        printf("Usage: filename tolerance\n");
        exit(1);
    }

    char* filename = argv[1];
    int tolerance = atoi(argv[2]);
    int opacity = 1;
    float voxelSize[3];
    CImg<> baseImg, regionImg;
    int intensity;

    baseImg.load_analyze(filename, voxelSize);
    regionImg = baseImg;

    CImgDisplay main_display(baseImg, "Liver segmentation");
    CImg<> visu(500, 400, 1, 3, 0);
    int slice_index = 21;
    CImg<> slice = baseImg.get_slice(slice_index);
    main_display.display(slice);

    while (!main_display.is_closed()) {
        main_display.wait();
        if(main_display.wheel()){
            // On change la coupe en fonction de l'action de la molette
            int counter = main_display.wheel();
            slice_index += counter;
            if(slice_index < 0)
                slice_index = 0;
            if(slice_index >= regionImg.depth())
                slice_index = regionImg.depth()-1;

            cout << "Affichage de la coupe " << slice_index+1 << endl;

            // On affiche la nouvelle coupe
            slice = regionImg.get_slice(slice_index);
            main_display.display(slice);

            // On remet la molette a 0 pour faciliter sa lecture
            main_display.set_wheel();
        }
        // Si on clique avec le bouton gauche de la souris
        else if(main_display.button()){
            Voxel voxel;
            voxel.x = main_display.mouse_x() * regionImg.width() / main_display.width();
            voxel.y = main_display.mouse_y() * regionImg.height() / main_display.height();
            voxel.z = slice_index;
            intensity = baseImg(voxel.x, voxel.y, voxel.z);

            // Vu que draw_fill prend egalement la tolerance en-dessous, ce qu'on ne veut pas
            regionImg = delete_below_intensity(baseImg, intensity);
            regionImg = extract_region(baseImg, voxel, opacity, tolerance);
            // On affiche la region extraite choisie par l'utilisateur
            main_display.display(regionImg.get_slice(slice_index));

            // Le seuil varie de 0 a cette valeur pour le graphe
            int nbOfSamples = 100;
            printf("Computing best possible region...\n");
            // On calcule les differentes regions possible
            CImg<> graph_img = compute_graph(baseImg, voxel, opacity, nbOfSamples);
```

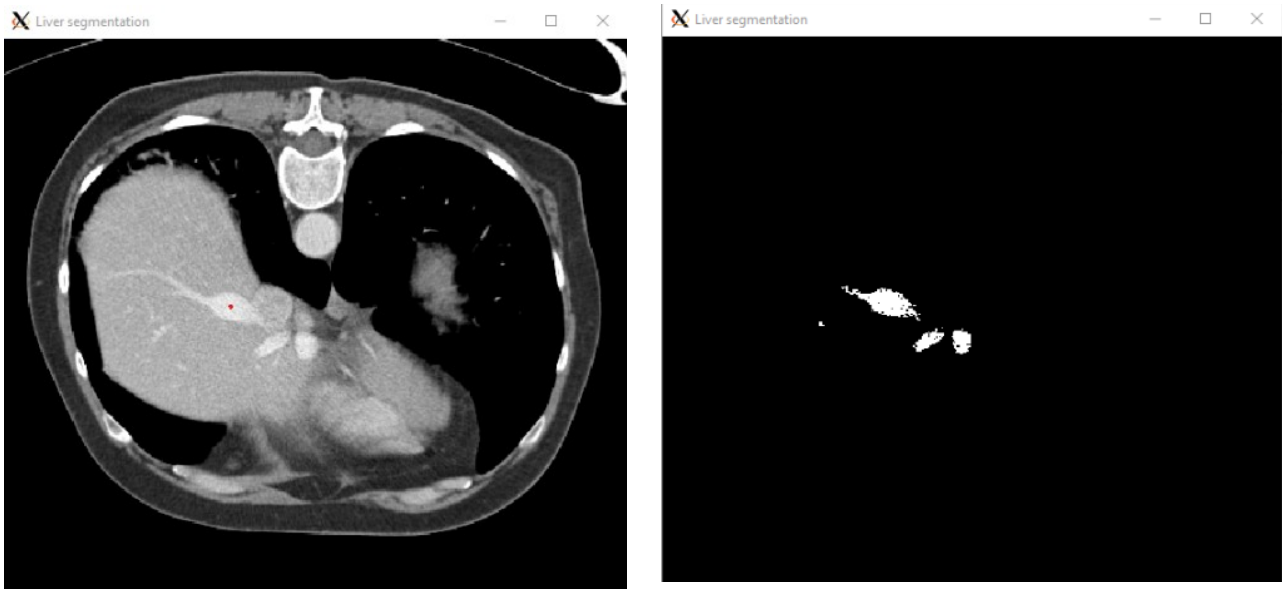
```
            CImgDisplay graph_display(visu, "Graph");
            // Construit le graphe avec la tolerance en abscisse et le nombre de voxels par region en ordonnee
            graph_img.display_graph(graph_display, 2, 1, "Tolerance", 0, nbOfSamples, "Amount", 0, get_max(graph_img), true);

            while(!graph_display.is_closed()){
                // En appuyant sur T on remplace la tolerance actuelle par le X du graphe de la position de la souris
                if(graph_display.is_keyT()){
                    tolerance = graph_display.mouse_x() * ((float)nbOfSamples/graph_display.width());
                    graph_display.close();
                }
            }

            // On recupere et affiche la nouvelle region
            regionImg = extract_region(baseImg, voxel, opacity, tolerance);
            main_display.display(regionImg.get_slice(slice_index));
        }
        // On affiche a nouveau l'image de base
        else if(main_display.is_keyR()){
            regionImg = baseImg;
            main_display.display(regionImg.get_slice(slice_index));
        }
    }

    return 0;
}
```

Les commentaires décrivent mon procédé mais je vais quand même le détailler.
Quand on lance le programme on passe en ligne de commande une tolérance.
Au lancement, l'image de base s'affiche, l'utilisateur sélectionne alors un certain pixel d'une certaine coupe de l'image 3D.
S'affiche alors la région correspondante et un calcul est fait pour afficher le graphe correspondant à la région en question extraite avec des différentes tolérances.
S'affiche ensuite une fenêtre avec un graphe sur lequel apparaît la courbe comprenant le nombre de voxels par région, en ordonnée, en fonction de la tolérance, en abscisse.
L'utilisateur peut alors placer la souris à l'endroit voulu de la courbe et appuyer sur T pour répéter l'extraction de région avec la nouvelle tolérance sélectionnée, le résultat s'affiche alors à l'écran.
En appuyant sur R on revient à l'image de base, sans extraction de région.

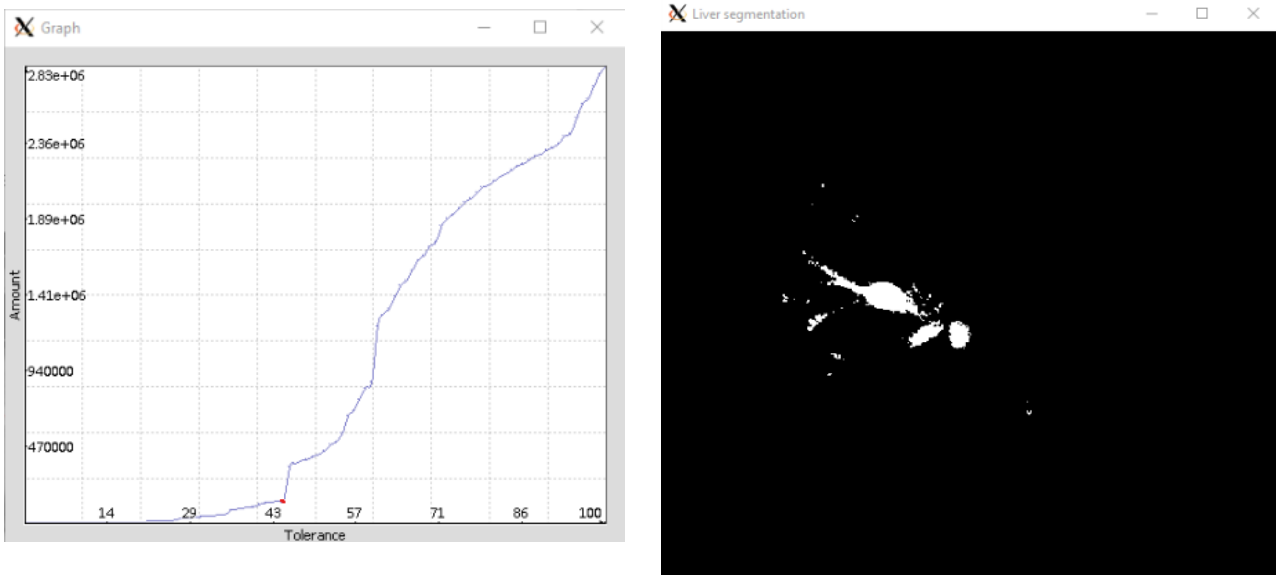
Question 4

Liver.hdr

J’ai cliqué au niveau du point rouge, sachant que ma tolérance de base est de 30.



A droite on voit la coupe de la région avec une tolérance de 30.

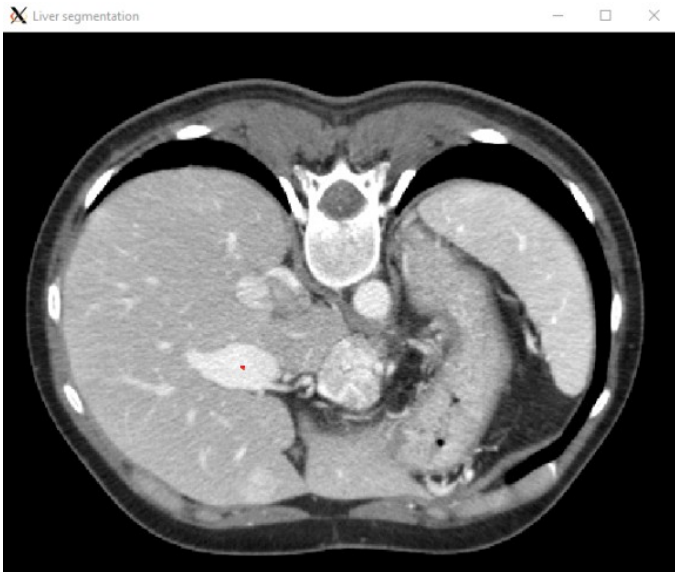


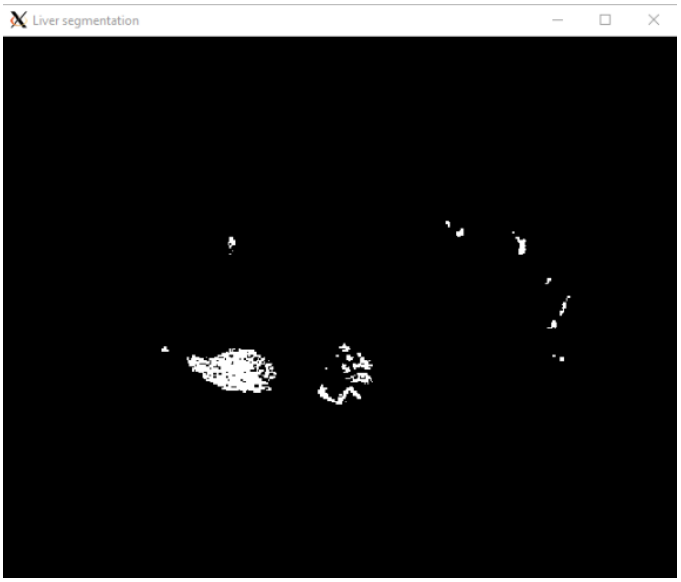
A droite la même coupe que précédemment mais cette fois-ci avec une tolérance de 44, la tolérance adéquate d’après le graphe.

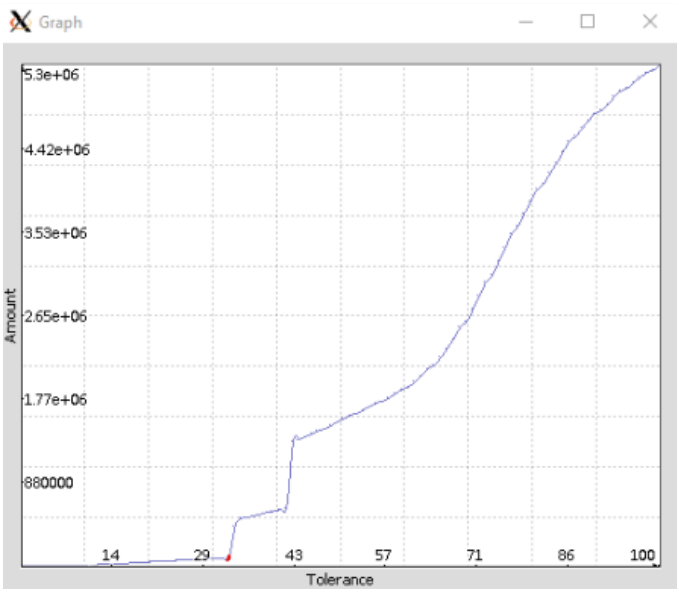
Voici l’image surfacique résultante.

BREBIX.hdr

J’ai également cliqué au niveau du point rouge avec une tolérance de base de 30.

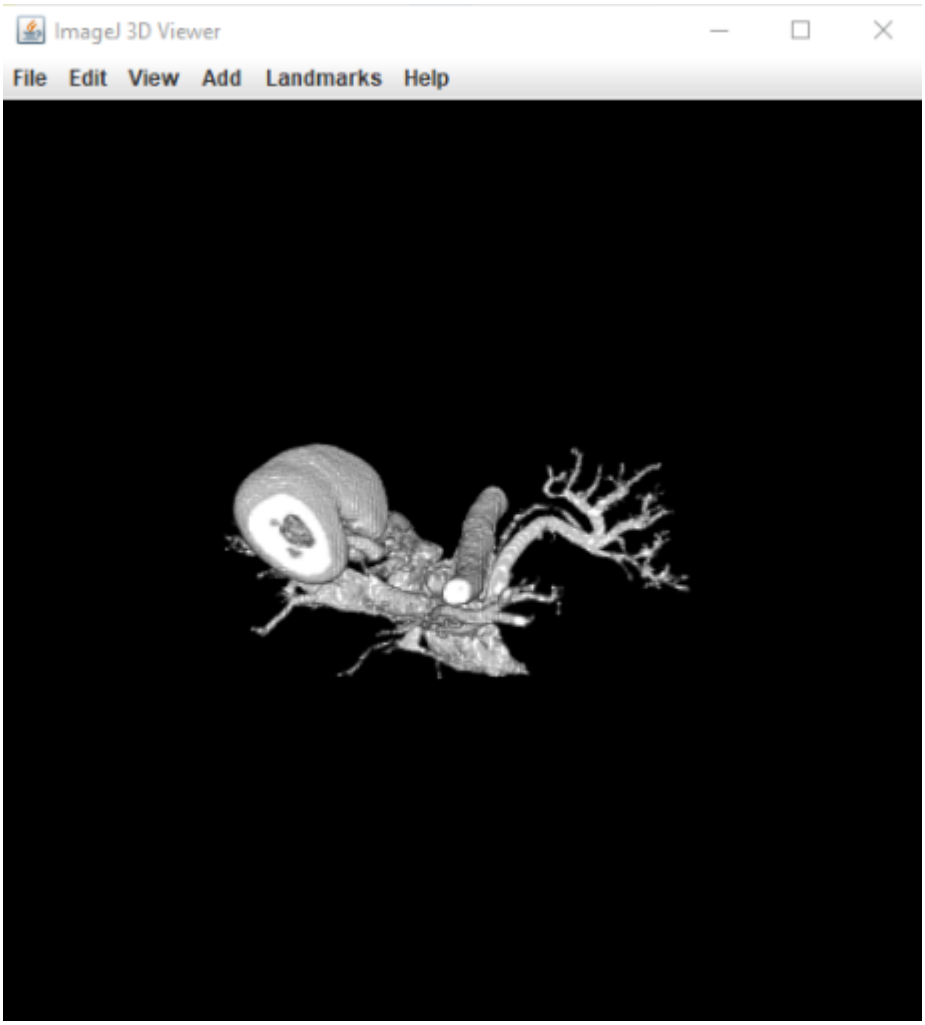






Je ne savais pas laquelle des deux tolérances choisir, j’ai donc pris la plus petite, qui est de 33. Le résultat ne change donc pas beaucoup par rapport au 30.

Voici l’image surfacique résultante.



Question 5.1

Une amélioration serait d’utiliser le même principe que draw_fill sauf qu’au lieu de renvoyer une image on renvoie directement le nombre de voxels de la région extraite. Pour construire la courbe on n’a pas besoin d’avoir les différentes régions extraites mais uniquement leur taille. Cependant cela ne change rien au niveau de la complexité de l’algorithme.
D’un autre côté si le seul but est de construire la courbe pour en déduire la valeur de tolérance la plus optimale pour la région en question, on peut aussi directement le faire sans avoir à construire toute la courbe, et donc, sans tester toutes les tolérances possible.
Il suffirait de commencer avec une tolérance égale à 0 et d’augmenter cette dernière à chaque itération. On regarde donc, à chaque itération, le nombre de voxels correspondant à la région, à la tolérance donnée. Dès que le nombre de voxels augmente drastiquement par rapport aux tolérances précédentes on peut

dire que la tolérance optimale pour la région sélectionnée est la tolérance précédent la courante.
On n’aurait donc pas a testé toutes les tolérances, ce qui voudrait dire qu’il ne faudrait pas extraire toutes les différentes régions de l’image 3D ce qui ferait un gain considérable.
Dans les exemples de la question 4 on se serait donc arrêté à 44 itérations et 33 itérations au lieu de faire la totalité des 100 itérations.