HMIN318M: Imagerie médicale et 3D

Rendu TP : Segmentation

Tianning MA

M2 IMAGINA

28/09/2020

Table de matière

A. Introduction 2
B. Interaction possible 2
C. Rendu et explication des exercices 2
1. Visualisation L'image 2
2. Histogramme 3
3. Seuil (disscociation le cerveau / matière blanche) 5
4. Algorithme de segmentation 6
5. Programmation de l'algorithme (Code source en Annexe) 6
6. Testes 6
a. Partie 1 Testes sur « MR_head.Coronal.hdr » 6
b. Partie 2 Testes sur « Brainseg » 9
7. Automatisation des paramètres 10
Annexe 1 : Version lisible 11
Annexe 2 : Version copiable 14

A. Introduction

Ce compte rendu est dédié au TP segmentation du cour d'Imagerie médicale 3D (HMIN318M). Le travail est basé sur Tp précédent (TP0).

Toutes les questions sont finies.

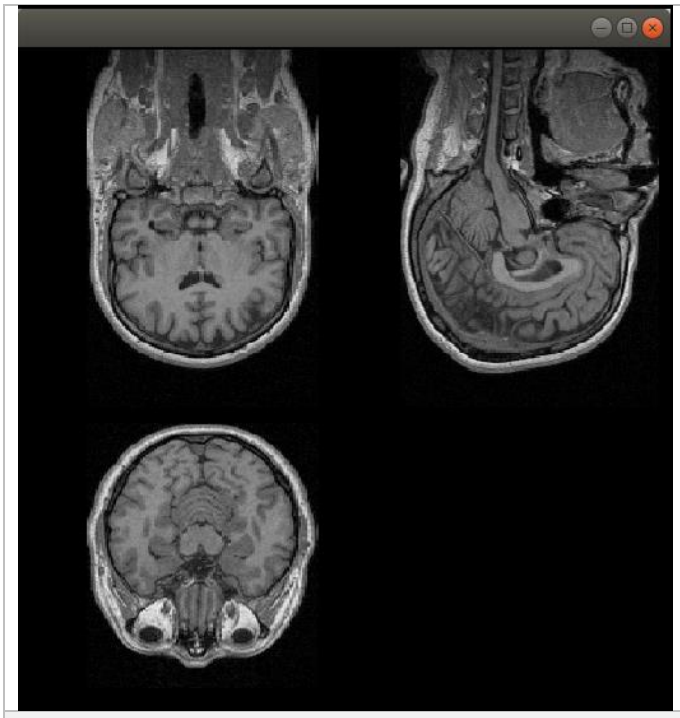
B. Interaction possible

Touche	Fonctionnement		
Molette de la souris	Changement de couches	M	Filtrage simple (blur)
+	Threshold(seuil) +=5	*	Nb fois d'érosion & dilatation ++
-	Threshold(seuil) -=5	/	Nb fois d'érosion & dilatation --

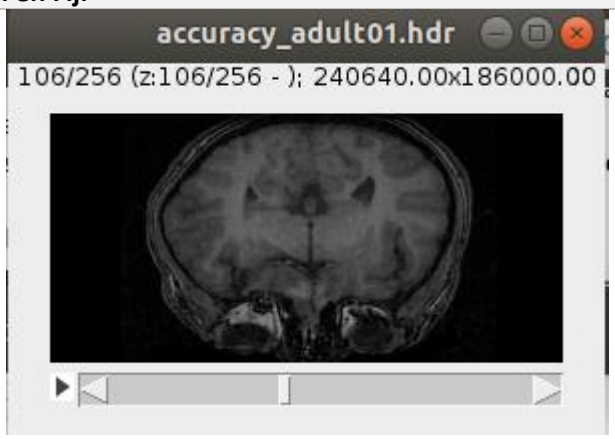
C. Rendu et explication des exercices

1. Visualisation L'image

Image « MR_head_Coronal »	**Image « brainseg »**
Visualisation sous linux avec le program	



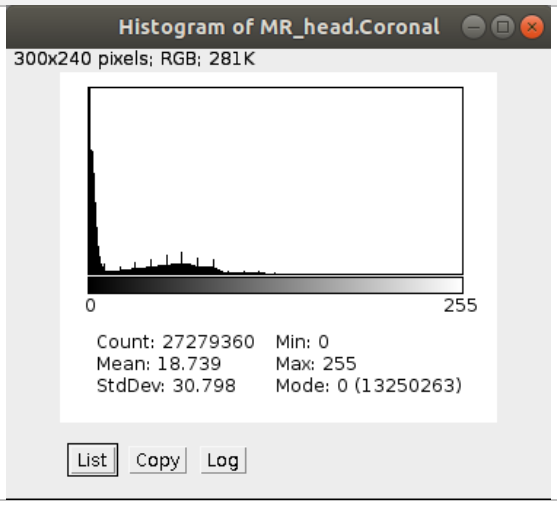
Visualisation en Fiji

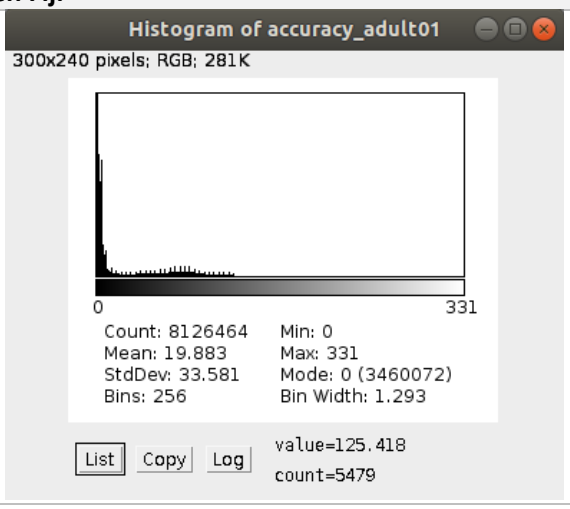


2. Histogramme

Image « MR_head_Coronal »	Image « brainseg »

Visualisation en Fiji



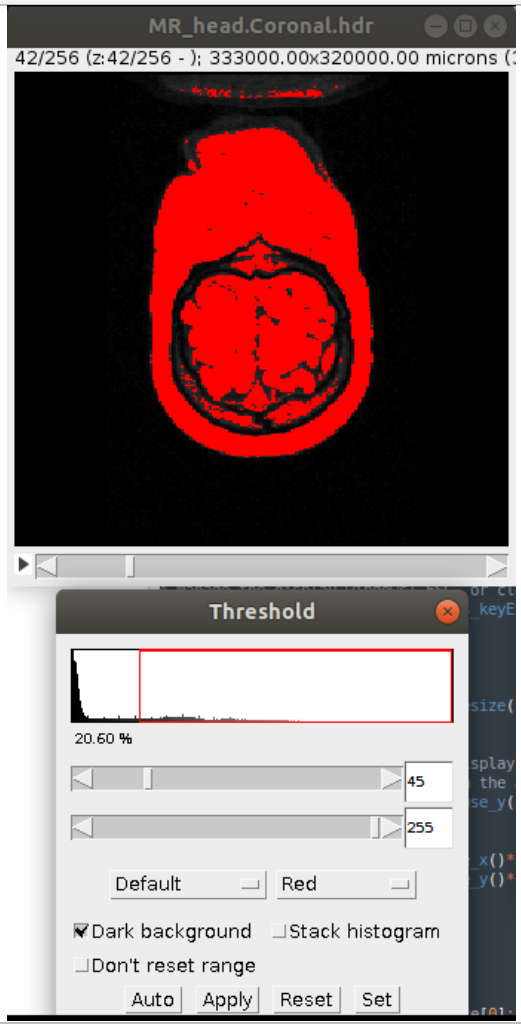
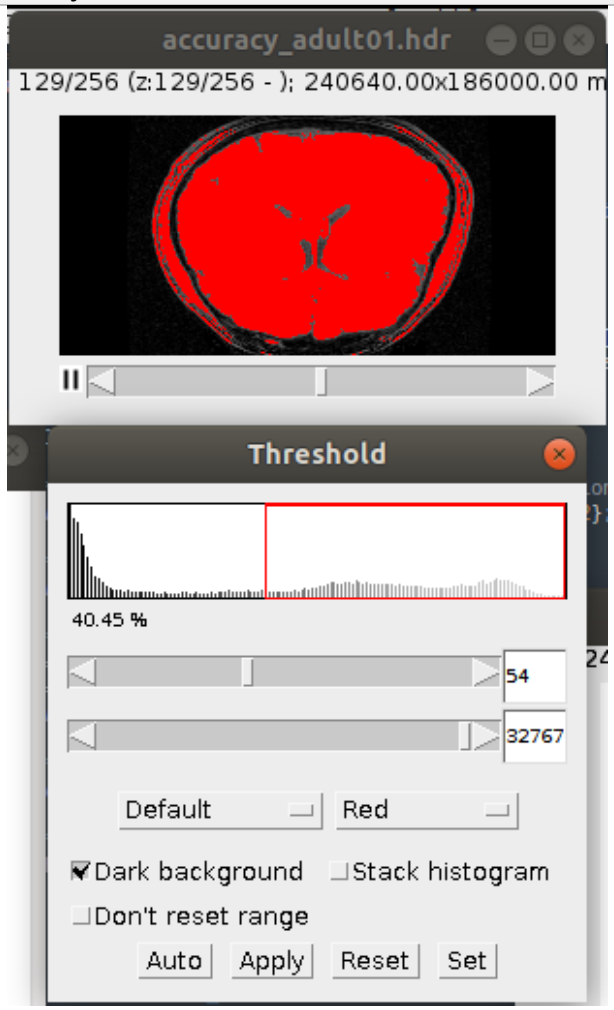


a. L'histogramme permet de représenter la répartition d'intensité des voxels dans une image 3D . Pour une donnée (x, y), la colonne du histogramme représente donc le nombre de voxels (y) dans l'image qui ont l'intensité x comme valeur.

Par exemple, pour histogramme de l'image « MR_head_Coronal », on peut constater que les valeurs de voxels se concentrent sur la partie gauche ([0,128]). L'image est constitué par une grande partie de l'air (en noir donc valeur proche de 0), une petite partie d'os (en gris clair, valeur plus proche de 255) et entre les deux, la matière grise. Sur l'histogramme, on peut clairement observer la répartition des différentes parties dans cette image 3D.

Pareil pour l'image brainseg. On voit la répartition des valeurs de voxels et elle est concentrée sur la partie gauche, donc l'image est sous exposée. Elle apparaît donc comme la première image plus sombre. De plus, on peut aussi remarquer les histogrammes commencent à se hachurer (plus évident que celui de première image.) Il est possible que ce soient des informations disparues. Enfin, on peut savoir aussi que les images ont besoins plus de contrasts

b. La méthode d'Otsu est utilisée pour effectuer un seuillage automatique à partir de la forme de l'histogramme de l'image.

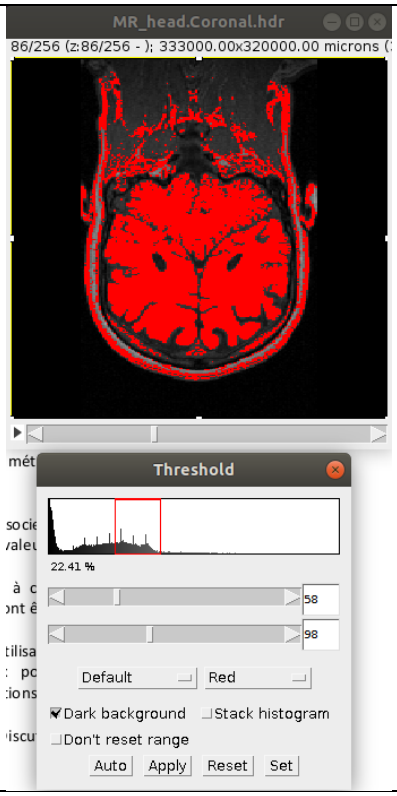
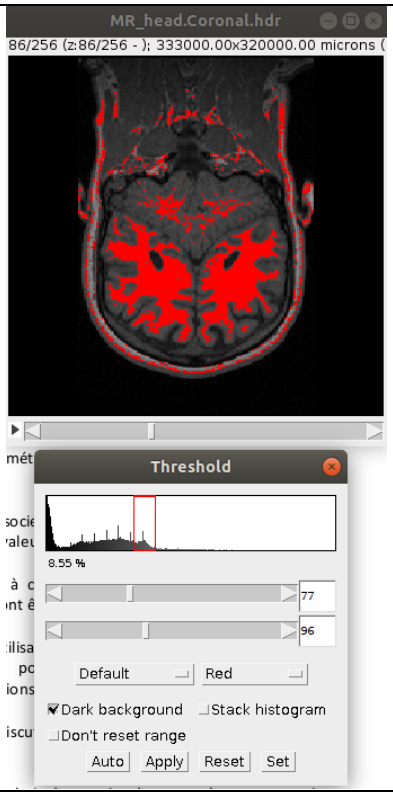
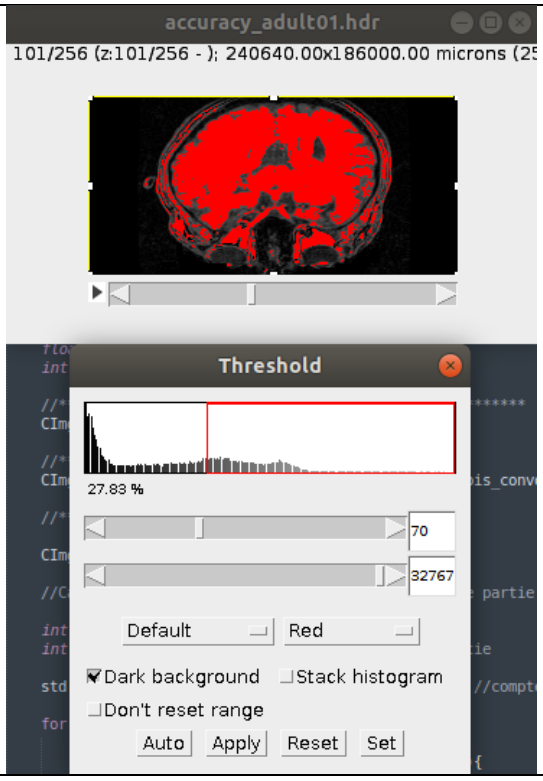
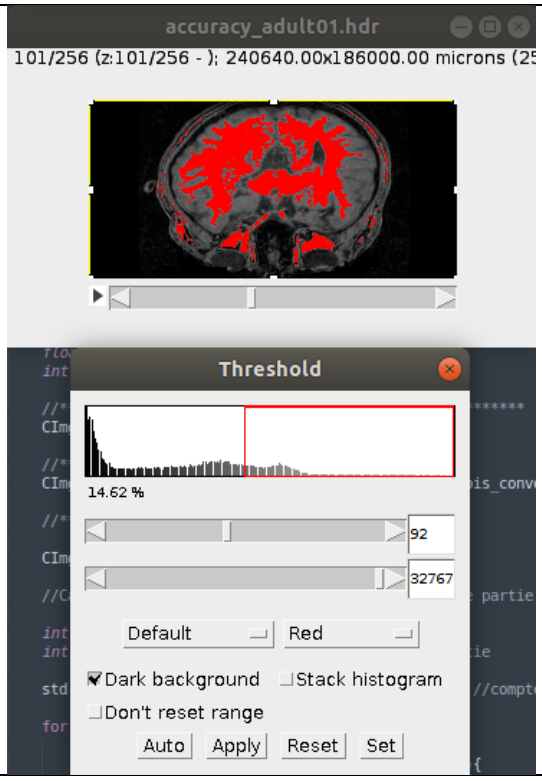
Image « MR_head_Coronal »	**Image « brainseg »**
Visualisation en Fiji	
	

La méthode d'Otsu consiste à trouver le seuil en minimisant la variance intra-classe. Cette valeur est définie comme une somme pondérée de la variance des deux classes (le premier plan et l'arrière-plan).

On considère d'abord l'image ne contient que ces deux classes, puis calcule le seuil optimal qui peut séparer ces deux classes pour que ses variances (dans les deux classes respectivement) soient minimales. Donc, on peut observer sur les images ci-dessus. Cette valeur corresponds la valeur des voxels (le seuil) et qui pourrait (idéalement) séparer l'objet et le fond.

(sur la première image, cette valeur est 45. La deuxième 54.)

3. Seuil (disscociation le cerveau / matière blanche)

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Dissocier au mieux le cerveau</td>
<td>Dissocier au mieux la matière blanche</td>
</tr>
<tr>
<td colspan="2">Après les deux images résultats ci-dessus, on constate que ce n’est pas facile de dissocier une partie. Par exemple si on veut disscocier la matière blanche de l’image, quand on modifie le seuil, il se peut d’avoir les « troux » dans la matière blanche. De plus, les parties (comme des taches) en dehors de matière blanche peut être récupérées aussi, à cause de ses valeurs qui sont proche de la matière blanche. Enfin, les liasons des parties peuvent aussi avoir des problèmes.</td>
</tr>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Dissocier au mieux le cerveau</td>
<td>Dissocier au mieux la matière blanche</td>
</tr>
<tr>
<td colspan="2">Pareil comme l’image précédente. On voit que non seulement la partie souhaitée est capturée, mais aussi plus ou moins les parties autours (soit l’os, soit les autres matières) sont aussi sélectionnées.</td>
</tr>
</table>

4. Algorithme de segmentation

Après avoir lu l'article, l'algorithme est principalement en 4 étapes :

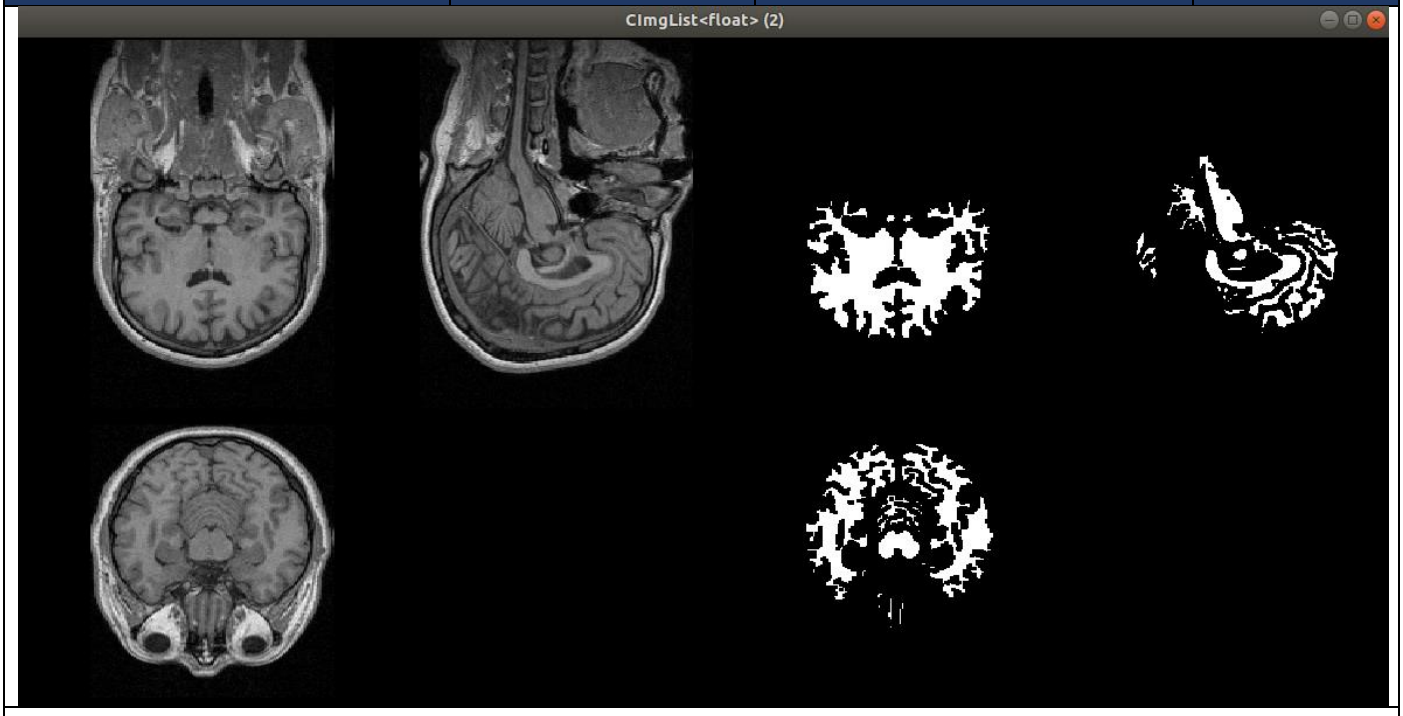
1. En utilisant les threshold, il faut faire un premier seuillage binaire.
2. Effectuer une opération érosion, et mettre à jour les voxels (enlever les taches isolés)
3. Labéliser les différentes composantes et ne conserver que (size) la plus grande (sauf le fond de l'image)
4. Effectuer une opération dilatation, enlever les effets d'érosion et obtenir l'image résultat.

5. Programmation de l'algorithme (Code source en Annexe)

6. Testes

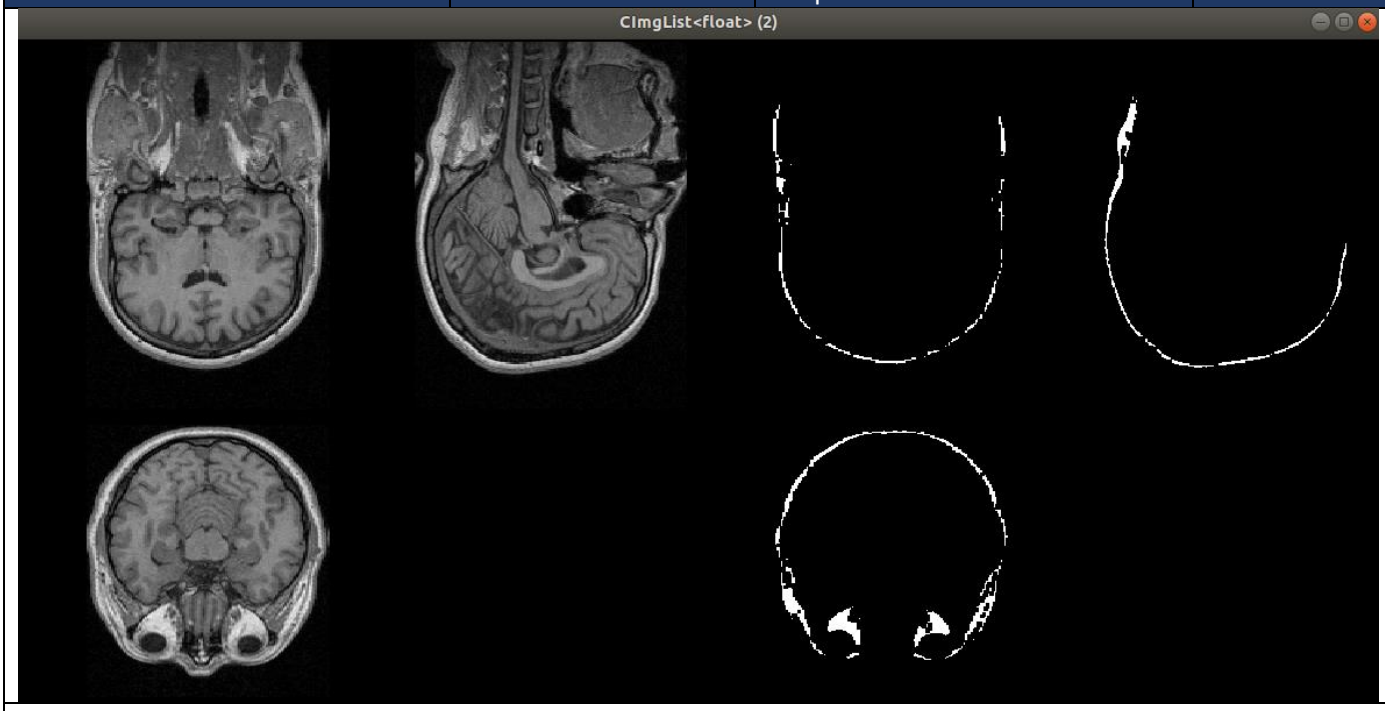
a. Partie 1 Testes sur « MR_head.Coronal.hdr »

Valeur de Threshold :	60	Nombre de fois pour érosion et dilatation :	3



Prenons le seuil de 60 pour isoler bien le cerveau dans l'image, puis j'applique 3 fois érosion pour éliminer les taches. Ensuite, j'ai fais la labélisation sur les différentes parties, et sélectionné la plus grande partie pour bien isoler l'objet que l'on veut. Enfin, j'ai appliqué 3 fois dilatation pour annuler les effets d'érosion et obtenir le résultat final.
Selon l'image ci-dessus, on s'aperçoit que on a bien isolé le cerveau.

Valeur de Threshold :	75	Nombre de fois pour érosion et dilatation :	3



Ensuite, j'ai aussi fait les essaies pour les autres valeurs de threshold. Par exemple, avec le seuil de 80, on obtient finalement l'image ci-dessus. On peut constater que le contour d'objet est bien isolé.

Valeur de Threshold :	40	Nombre de fois pour érosion et dilatation :	3

Enfin, avec un seuil de 40. J'ai obtenu le résultat ci-dessus. On s'aperçoit que le cerveau n'est pas encore bien isolé. Néanmoins, dans l'ensemble de l'image 3D, on n'a pas autant de « taches » isolés grâce aux opération fermeture (érosion - dilatation) sur les voxels.

Essaies sur les nombres de fois pour ***l'opération érosion – dilatation***

Valeur de Threshold :	60	Nombre de fois pour érosion et dilatation :	2

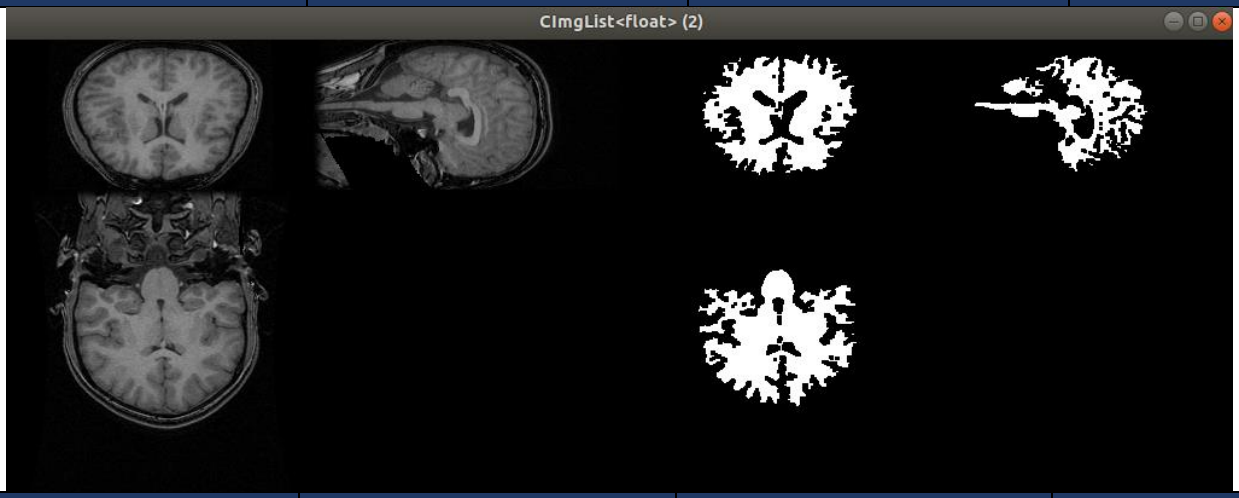
Valeur de Threshold :	60	Nombre de fois pour érosion et dilatation :	5

Si on diminue le nombre de fois pour l'érosion et dilatation, on s'aperçoit que la partie souhaitée n'est pas idéalement isolée. Néanmoins, si on augement trop le nombre de fois pour ces opération, on constate aussi que la partie isolée n'est pas assez précise. Donc, il faut faire un compromis et trouver le nombre de fois idéal pour ces opérations.

b. Partie 2 Testes sur « Brainseg »

Comme pour l'image précédante, voici les images résultats pour celle de « Brainseg ». On constate que pour avoir un résultat idéal, pour une différente image, on doit modifier la valeur de seuil ou le nombre de fois pour l'érosion/dilatation. Par exemple, pour l'image de brainseg, on s'aperçoit que avec une valeur de 75 comme threshold, le rendu est plus idéale. (alors que sur l'image précédente, le threshold était 60)

Valeur de Threshold :	75	Nombre de fois pour érosion et dilatation :	3



Valeur de Threshold :	60	Nombre de fois pour érosion et dilatation :	3

Valeur de Threshold :	100	Nombre de fois pour érosion et dilatation :	3

7. Automatisation des paramètres

Pour rendre interactif le choix des paramètres, j'ai ajouté certaines intéractions des touches qui permettront de visualiser un rendu correspondant en appuyant sur les touches.

```
/***** Intéraction des touches *****/
if (disp.is_keyM()){ img.blur(2.0,2.0,2.0,true,false); redraw = true; }
if (disp.is_keyN()){ img.sharpen(20.0,20.0,20.0,true,false); redraw = true;}

if(disp.is_keyPADADD()){ seuil+=5; redraw = true; }
if(disp.is_keyPADSUB()){ seuil-=5; redraw = true; }
if(disp.is_keyPADMUL()){ nb_fois_convolution++; redraw = true; }
if(disp.is_keyPADDIV()){ nb_fois_convolution--; redraw = true;}
```

Le paramètre « seuil » sera passé dans la fonction threshold pour le premier seuillage binaire.

Le paramètre « nb_fois_convolution » sera passé dans les fonctions erode et dilate, pour effectuer ces opérations.

Annexe 1 : Version lisible

```cpp
/*  HMIN318 Tianning MA
    M2 IMAGINA - TP Segmentation
    Compilation:
    (Linux) g++  -o visu4.exe visu4.cpp -O2 -L/usr/X11R6/lib  -lm  -lpthread -lX11
    Execution : ./ visu4.exe nom_fichier.hdr
*/

#include "CImg.h"
#include<iostream>
#include <vector>
using namespace cimg_library;

float seuil = 80.0;
int nb_fois_convolution = 3;

/* Main program */
int main(int argc,char **argv)
{
    /* Create and load the 3D image */
    CImg<> img;
    float voxelsize[3];
    /* Load in Analyze format and get the voxel size in an array */
    img.load_analyze(argv[1],voxelsize);


    /* Get the image dimensions */
    int dim[]={img.width(),img.height(),img.depth()};
    printf("Reading %s. Dimensions=%d %d %d\n",argv[1],dim[0],dim[1],dim[2]);
    printf("Voxel size=%f %f %f\n",voxelsize[0],voxelsize[1],voxelsize[2]);

    /* Create the display window of size 512x512 */
    CImgDisplay disp(512*2,512,"Segmentation");

    /* The 3 displayed slices of the MPR visualisation */
    int displayedSlice[]={dim[0]/2,dim[1]/2,dim[2]/2};

    /* Slice corresponding to mouse position: */
    unsigned int coord[]={0,0,0};

    /* The display window corresponds to a MPR view which is decomposed into the following 4 quadrants:
    2=original slice size=x y        0 size=z y
    1= size=x z                     -1 corresponds to the 4th quarter where there is nothing displayed */
    int plane=2;

    /* For a first drawing, activate the redrawing flag */
    bool redraw=true;

    /* Manage the display windows: ESC, or closed -> close the main window */
    while (!disp.is_closed() && !disp.is_keyESC()) // Main loop
    {
        /* List of events */
        /***** Intéraction des touches *****/
        if (disp.is_keyM()){ img.blur(2.0,2.0,2.0,true,false); redraw = true; }
        if (disp.is_keyN()){ img.sharpen(20.0,20.0,20.0,true,false); redraw = true;}

        if(disp.is_keyPADADD()){ seuil+=5; redraw = true; }
        if(disp.is_keyPADSUB()){ seuil-=5; redraw = true; }
        if(disp.is_keyPADMUL()){ nb_fois_convolution++; redraw = true; }
        if(disp.is_keyPADDIV()){ nb_fois_convolution--; redraw = true;}


        /* Resizing */
        if (disp.is_resized())
        {
            disp.resize();
        }
        /* Movement of the mouse */

        /* If the mouse is inside the display window, find the active quadrant
        and the relative position within the active quadrant */
        if(disp.mouse_x()>=0 && disp.mouse_y()>=0)
        {

            unsigned int mX = disp.mouse_x()*(dim[0]+dim[2])/disp.width();
            unsigned int mY = disp.mouse_y()*(dim[1]+dim[2])/disp.height();

```

```
            if (mX>=dim[0] && mY<dim[1])
            {
                plane = 0;
                coord[1] = mY;
                coord[2] = mX - dim[0];
                coord[0] = displayedSlice[0];
            }
            else
            {
                if (mX<dim[0] && mY>=dim[1])
                {
                    plane = 1;
                    coord[0] = mX;
                    coord[2] = mY - dim[1];
                    coord[1] = displayedSlice[1];
                }
                else
                {
                    if (mX<dim[0] && mY<dim[1])
                    {
                        plane = 2;
                        coord[0] = mX;
                        coord[1] = mY;
                        coord[2] = displayedSlice[2];
                    }
                    else
                    {
                        plane = -1;
                        coord[0] = 0;
                        coord[1] = 0;
                        coord[2] = 0;
                    }
                }
            }
            redraw = true;
        }

        /* Click Right button to get a position */
        if (disp.button()&2  && (plane!=-1))
        {
            for(unsigned int i=0;i<3;i++)
            {
                displayedSlice[i]=coord[i];
            }
            redraw = true;
        }
```

```
        /* Wheel interaction */
        if (disp.wheel())
        {
            displayedSlice[plane]=displayedSlice[plane]+disp.wheel();

            if (displayedSlice[plane]<0)
            {
                displayedSlice[plane] = 0;
            }
            else
            {
                if (displayedSlice[plane]>=(int)dim[plane])
                {
                    displayedSlice[plane] = (int)dim[plane]-1;
                }
            }

        /* Flush all mouse wheel events in order to not repeat the wheel event */
        disp.set_wheel();
        redraw = true;
        }
```

```
        if (redraw)
        {
            /* Create a 2D image based on the MPR projections given by a projection point
            which is the intersection of the displayed slices */
            CImg<> mpr_img=img.get_projections2d(displayedSlice[0],displayedSlice[1],displayedSlice[2]);

            /* The MPR image has a given size. It needs to be resized in order to fit at best in the display window */
            mpr_img.resize(512,512);

            //******* Etape 1 : Seuillage sur l'image *******
            CImg<> res = img.get_threshold(seuil);

            //******* Etape 2 - Opération - Erosion *******
            CImg<> erode = res.erode(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);

            //******* Etape 3 - Labélisation *******
            CImg<> labels = res.label(false);

            //Calcul la labélisation et choisir le plus grande partie
            int max_size = 0;
            int max_label = 0; // Label de La Plus Grande Partie

            std::vector<int> compteur(labels.max() + 1 , 0 ); //compteur pour chaque labels de l'image

            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) > 0) {   // Enlever label du fond

                            compteur[(int)labels.atXYZ(x,y,z)]++;

                            if(compteur[labels.atXYZ(x, y, z)] > max_size) {
                                max_size = compteur[labels.atXYZ(x, y, z)];
                                max_label = labels.atXYZ(x, y, z);
                            }
                        }
                    }
                }
            }

            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) != max_label)
                                res(x,y,z) = 0;
                    }
                }
            }

            //******* Etape 4 dilatation (Enlever les effets d'érosion d'avant) *******
            erode.dilate(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);

            disp.display((mpr_img,res.normalize(0,255)));


            /* To avoid repetitive continuous redrawing */
            redraw=false;
        }
    }
    return 0;
}
```

Annexe 2 : Version copiable

```
/* HMIN318 Tianning MA
M2 IMAGINA - TP Segmentation
Compilation:
(Linux) g++  -o visu4.exe visu4.cpp -O2 -L/usr/X11R6/lib  -lm  -lpthread -lX11
Execution : ./ visu4.exe nom_fichier.hdr
*/
#include "CImg.h"
#include<iostream>
#include <vector>
using namespace cimg_library;

float seuil = 80.0;
int nb_fois_convolution = 3;

/* Main program */
int main(int argc,char **argv)
{
/* Create and load the 3D image */
CImg<> img;
float voxelsize[3];
/* Load in Analyze format and get the voxel size in an array */
img.load_analyze(argv[1],voxelsize);

/* Get the image dimensions */
   int dim[]={img.width(),img.height(),img.depth()};
printf("Reading %s. Dimensions=%d %d %d\n",argv[1],dim[0],dim[1],dim[2]);
printf("Voxel size=%f %f %f\n",voxelsize[0],voxelsize[1],voxelsize[2]);
/* Create the display window of size 512x512 */
CImgDisplay disp(512*2,512,"Segmentation");
/* The 3 displayed slices of the MPR visualisation */
int displayedSlice[]={dim[0]/2,dim[1]/2,dim[2]/2};
/* Slice corresponding to mouse position: */
unsigned int coord[]={0,0,0};
/* The display window corresponds to a MPR view which is decomposed into the following 4 quadrants:
2=original slice size=x y       0 size=z y
1= size=x z                  -1 corresponds to the 4th quarter where there is nothing displayed */
int plane=2;
/* For a first drawing, activate the redrawing flag */
bool redraw=true;
/* Manage the display windows: ESC, or closed -> close the main window */
while (!disp.is_closed() && !disp.is_keyESC()) // Main loop
{
/* List of events */
/***** Intéraction des touches *****/
if (disp.is_keyM()){ img.blur(2.0,2.0,2.0,true,false); redraw = true; }
if (disp.is_keyN()){ img.sharpen(20.0,20.0,20.0,true,false); redraw = true;}

if(disp.is_keyPADADD()){ seuil+=5; redraw = true; }
if(disp.is_keyPADSUB()){ seuil-=5; redraw = true; }
if(disp.is_keyPADMUL()){ nb_fois_convolution++; redraw = true; }
if(disp.is_keyPADDIV()){ nb_fois_convolution--; redraw = true;}


/* Resizing */
if (disp.is_resized())
```

```
{
disp.resize();
}
/* Movement of the mouse */
/* If the mouse is inside the display window, find the active quadrant
and the relative position within the active quadrant */
if(disp.mouse_x()>=0 && disp.mouse_y()>=0)
{
unsigned int mX = disp.mouse_x()*(dim[0]+dim[2])/disp.width();
unsigned int mY = disp.mouse_y()*(dim[1]+dim[2])/disp.height();
if (mX>=dim[0] && mY<dim[1])
{
plane = 0;
coord[1] = mY;
coord[2] = mX - dim[0];
coord[0] = displayedSlice[0];
}
else
{
if (mX<dim[0] && mY>=dim[1])
{
plane = 1;
coord[0] = mX;
coord[2] = mY - dim[1];
coord[1] = displayedSlice[1];
}
else
{
if (mX<dim[0] && mY<dim[1])
{
plane = 2;
coord[0] = mX;
coord[1] = mY;
coord[2] = displayedSlice[2];
}
else
{
plane = -1;
coord[0] = 0;
coord[1] = 0;
coord[2] = 0;
}
}
}
redraw = true;
}
/* Click Right button to get a position */
if (disp.button()&2  && (plane!=-1))
{
for(unsigned int i=0;i<3;i++)
{
displayedSlice[i]=coord[i];
}
redraw = true;
}

/* Wheel interaction */
```

```
if (disp.wheel())
{
displayedSlice[plane]=displayedSlice[plane]+disp.wheel();
if (displayedSlice[plane]<0)
{
displayedSlice[plane] = 0;
}
else
{
if (displayedSlice[plane]>=(int)dim[plane])
{
displayedSlice[plane] = (int)dim[plane]-1;
}
}

/* Flush all mouse wheel events in order to not repeat the wheel event */
disp.set_wheel();
redraw = true;
}



if (redraw)
{
/* Create a 2D image based on the MPR projections given by a projection point
which is the intersection of the displayed slices */
CImg<> mpr_img=img.get_projections2d(displayedSlice[0],displayedSlice[1],displayedSlice[2]);
/* The MPR image has a given size. It needs to be resized in order to fit at best in the display window */
mpr_img.resize(512,512);
//******* Etape 1 : Seuillage sur l'image *******
CImg<> res = img.get_threshold(seuil);
//******* Etape 2 - Opération - Erosion *******
CImg<> erode = res.erode(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);

//******* Etape 3 - Labélisation *******
CImg<> labels = res.label(false);

//Calcul la labélisation et choisir le plus grande partie
int max_size = 0;
int max_label = 0; // Label de La Plus Grande Partie

std::vector<int> compteur(labels.max() + 1 , 0 ); //compteur pour chaque labels de l'image

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) > 0) { // Enlever label du fond

compteur[(int)labels.atXYZ(x,y,z)]++;

if(compteur[labels.atXYZ(x, y, z)] > max_size) {
max_size = compteur[labels.atXYZ(x, y, z)];
                max_label = labels.atXYZ(x, y, z);
}
}
}
}
```

```
}

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) != max_label)
res(x,y,z) = 0;
}
}
}

//******* Etape 4 dilatation (Enlever les effets d'érosion d'avant) *******
erode.dilate(nb_fois_convolution,nb_fois_convolution,nb_fois_convolution);

disp.display((mpr_img,res.normalize(0,255)));


/* To avoid repetitive continuous redrawing */
redraw=false;
}
}
return 0;
}
```