TP3
Segmentation 3D par recalage

1. Profils d'intensité

On va récupérer les profils d'intensité pour chaque point du maillage. Pour chaque point, on récupère ses coordonnées, et sa normale. Ensuite, on va récupérer les valeurs associées à chaque point vers l'intérieur, puis vers l'extérieur du maillage.

Le code est le suivant :

```
CImg<unsigned char> prof(Ni+No,mesh.getNbPoints());
float p[3];
float n[3];

float pp[3];
for(int i = 0; i < mesh.getNbPoints(); ++i){
    mesh.getPoint(p, i);
    mesh.getNormal(n, i);
    int cpt = 0;
    for(int j = 0; j < Ni; ++j){
        for(int k = 0; k < 3; ++k){
            pp[k] = p[k] - n[k] * l * j;
        }
        prof(cpt++, i) = img.getValue(pp, interpolationType);
    }
    for(int j = 0; j < No; ++j){
        for(int k = 0; k < 3; ++k){
            pp[k] = p[k] + n[k] * l * j;
        }
        prof(cpt++, i) = img.getValue(pp, interpolationType);
    }
}

//prof.display();

return prof;
```

On observe le résultat suivant après le calcul des profils. La démarquation verticale correspond au ‘bord’ du maillage.

2. Calcul de la similarité

Seule la méthode SSD fonctionne, je n'ai pas réussi l'implémentation de NCC. Pour chaque point de l'image source, on va parcourir une région associée dans l'image cible, à partir de ce point, plus et moins un décalage donné.

Le code est le suivant :

```
float sum;

//parcours en hauteur
for(int h = 0; h < nbPoints; ++h){
    for(int j = -S; j < S; ++j){
        sum = 0;
        for(int w = 0; w < N; ++w){
            sum += pow((float)sourceProf(w, h) - (float)targetProf(w + S + j, h), 2);
        }
        sum /= (float)(2 * S);
        dist(j + S, h) = sum;
    }
}
```

Le résultat semble cohérent, en zoomant, on remarque que les lignes ne sont pas de la meme couleur tout le long.

3. Calcul des correspondances

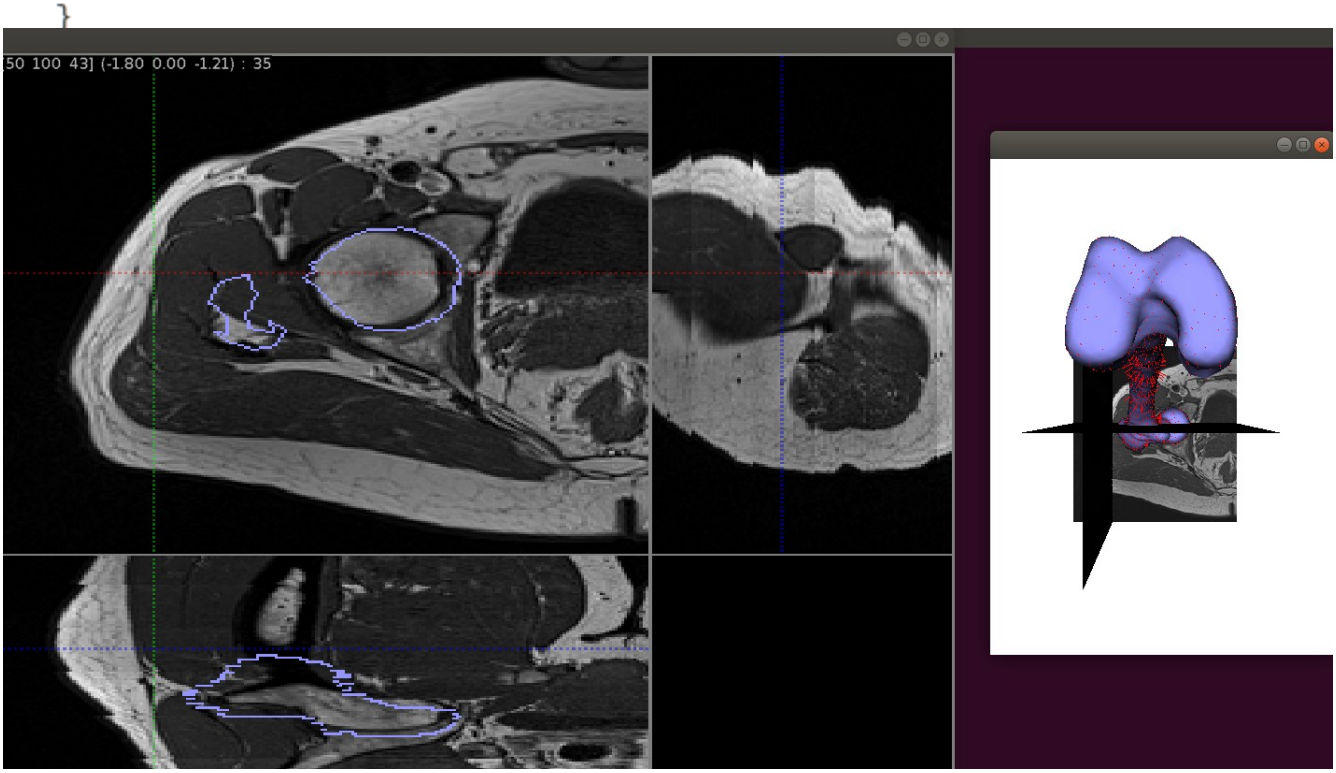
Pour chaque ligne issue du calcul de similarité, on récupère l'élément minimal. On va ensuite déplacer le point du maillage en fonction de cet élément, et de sa normale.

```
unsigned int nbpoints = dist.height();
unsigned int S = dist.width() / 2;

for(unsigned int i = 0; i < nbpoints; ++i){
    float min = dist(i, 0);
    float jMin = 0;
    for(int j = 1; j < dist.width(); ++j){
        if(dist(i, j) < min){
            min = dist(i, j);
            jMin = j;
        }
    }
    float p[3];
    float n[3];
    mesh.getPoint(p, i);
    mesh.getNormal(n, i);
    for(int k = 0; k < 3; ++k){
        p[k] = p[k] - l * jMin * n[k];
    }

    mesh.setCorrespondence(p,i);
}
```

Après le déroulement de l'algorithme, on obtient les résustats ci dessous qui me semblent corrects.



4. Tests effectués

En utilisant Rigid et Similitude, on obtient des résultats assez nets mais, jamais quasiment identique.
C'est en utilisant Affine que l'on obtient les meilleurs résultats pour un tour d'algorithme donné, avant que l'os ne retrécisse ou ne grandisse trop.

Pour la position initiale, l'algorithme ne va pas fonctionner si l'on se place au hasard, il faut garder des similitudes avec la réalité, sinon rien ne pourra être trouvé.

En faisant varier S, on observe deux comportements. Si S est trop petit, le recalage effectué ne bougera pas, ou très peu l'image, car il n'y aura pas assez de profils testés. Inversement, si S est trop grand, l'algorithme trouvera des similitudes qui ne sont pas présentes, à cause de la trop grande fourchette de valeurs testées.

Si on fait varier l, le problème d'un trop grand, ou trop petit décalage se pose aussi, étant donné que ce coefficient est utiliser pour calculer la nouvelle position de chaque point du maillage après le recalage.

Enfin, en faisant varier Ni et No, on peut obtenir des résultats incohérents, car l'os peut être 'traversé' si les valeurs sont trop grandes, et donc, obtenir des similitudes avec la matière de l'os là où il ne devrait pas y en avoir.