Compte-rendu TP3

Calcul des profils d'intensité

```
CImg<unsigned char> computeProfiles(const MESH& mesh, const IMG<unsigned char,float>& img, const unsigned int Ni,
    const unsigned int No, const float l,const unsigned int interpolationType=1)
{
    CImg<unsigned char> prof(Ni+No,mesh.getNbPoints());

    float p[3];
    float n[3];
    float tmpP[3];

    for(int i=0; i<mesh.getNbPoints(); i++){
        mesh.getPoint(p, i);
        mesh.getNormal(n, i);
        // On descend la normale
        for(int j=0; j<Ni; j++){
            // Pour chaque coordonnée on descend la normale avec le pas L
            for(int a=0; a<3; a++){
                tmpP[a] = p[a] - n[a] * l * j;
            }
            // On assigne la valeur au bon index, le plus en bas sera le plus à gauche
            prof(Ni-j-1, i) = img.getValue(tmpP, interpolationType);
        }
        // On monte la normale
        for(int j=0; j<No; j++){
            // Pour chaque coordonnée on monte la normale avec le pas L
            for(int a=0; a<3; a++){
                tmpP[a] = p[a] - n[a] * l * j;
            }
            // On assigne la valeur au bon index, le plus en haut sera le plus à droite
            prof(Ni+j, i) = img.getValue(tmpP, interpolationType);
        }
    }

    prof.display();

    return prof;
}
```

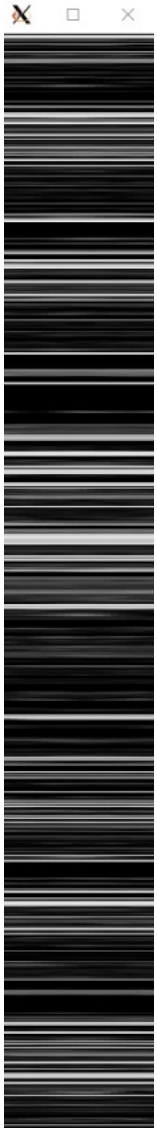
Pour récupérer les profil d'intensité on parcourt tous les pixels du maillage. Pour chaque pixel on récupère la normale, c'est elle qui va nous donner l'intensité, car on va la parcourir de bas en haut. Une ligne de l'image profil résultante correspond alors à l'intensité de la normale du pixel du maillage à cette position.

Voici le résultat

Calcul des profils de similarité

Voici mon implémentation de la mesure SSD :

```
if(metric == SSD)
{
    float sum;
    for(int y=0; y<nbpoints; y++){
        for(int i=-S; i<S; i++){
            sum = 0;
            // Calcul de la somme des differences au carre
            for(int x=0; x<N; x++){
                // +S dans target car l'image deborde de S a gauche et a droite
                sum += pow(targetProf(x+i+S, y) - sourceProf(x, y), 2);
            }
            // On veut avoir la moyenne de la somme
            dist(i+S, y) = sum / (2*S);
        }
    }
}
```



Voici mon implémentation de la mesure NCC :

```
else // NCC
{
    float aveS = 0;
    float aveT = 0;
    float upperCC = 0;
    float lowerLeftCC = 0;
    float lowerRightCC = 0;
    for(int y=0; y<nbpoints; y++){
        // Calcul de la moyenne pour la source
        aveS = 0;
        for(int z=0; z<sourceProf.width(); z++){
            aveS += sourceProf(z, y);
        }
        aveS = aveS / sourceProf.width();
        // Calcul de la moyenne pour la target
        aveT = 0;
        for(int z=0; z<targetProf.width(); z++){
            aveT +=  targetProf(z, y);
        }
        aveT = aveT / targetProf.width();
        for(int i=-S; i<S; i++){
            upperCC = 0;
            for(int x=0; x<N; x++){
                upperCC += (targetProf(x+i+S, y) - aveT) * (sourceProf(x, y) - aveS);
            }
            lowerLeftCC = 0;
            lowerRightCC = 0;
            for(int x=0; x<N; x++){
                lowerLeftCC += pow(targetProf(x+i+S, y) - aveT, 2);
                lowerRightCC += pow(sourceProf(x, y) - aveS, 2);
            }

            dist(i+S, y) = (upperCC) / sqrt(lowerLeftCC * lowerRightCC);
        }
    }
}
```

Pour chaque méthode de mesure j’ai donc simplement appliqué la formule correspondante pour chaque point du profil source.

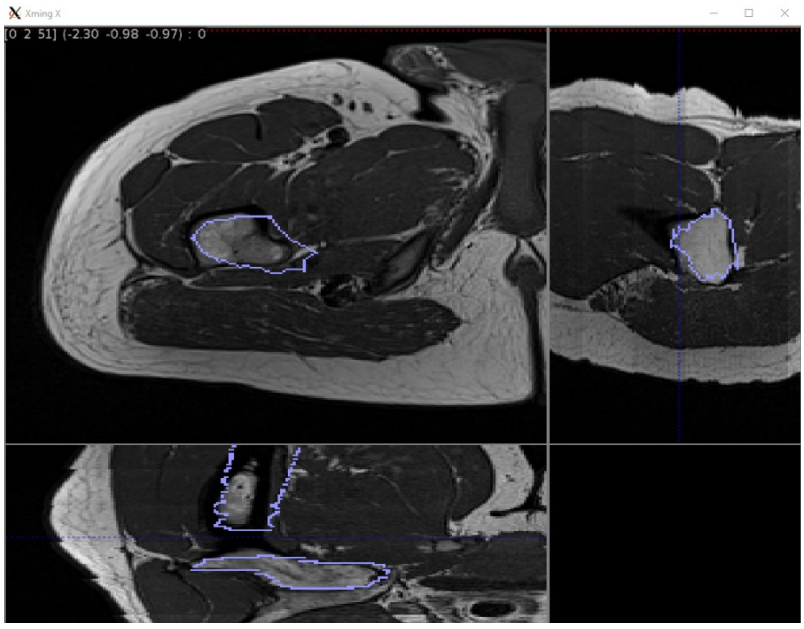
Calcul des correspondances

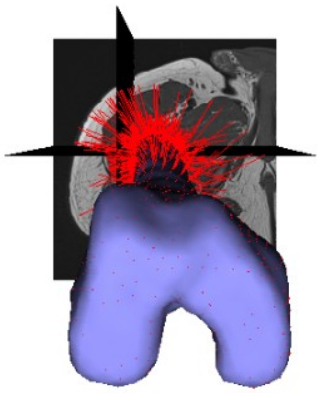
```
void computeCorrespondences(MESH& mesh, const CImg<float> &dist,const float l)
{
    unsigned int nbpoints=dist.height();
    unsigned int S=dist.width()/2;

    for(unsigned int i=0;i<nbpoints;i++)
    {
        float p[3];
        float tmpP[3];
        float n[3];
        mesh.getPoint(p, i);
        mesh.getNormal(n, i);
        int x;
        float min = 1000;
        // On cherche la distance minimale
        for(int j=0; j<dist.width(); j++){
            if(dist(j, i) < min){
                min = dist(j, i);
                x = j;
            }
        }
        // On applique la transformation au point
        for(int a=0; a<3; a++){
            tmpP[a] = p[a] + x * l * n[a];
        }
        mesh.setCorrespondence(tmpP, i);
    }

}
```

J’ai eu un résultat correct avec la méthode SSD, le voici :





En revanche, le résultat, si on peut parler de résultat, avec la méthode NCC ne correspondant à rien, le maillage disparaît dès la première itération.

Tests

Je n’ai pas pu tester suivant la mesure de similarité car ma mesure NCC ne semble pas être correcte.

Suivant le type de transformation :

Similitude et Rigide me donne les meilleurs résultats, avec Rigide le résultat est quasi parfait. Avec Affine ça ne marche pas trop.

Suivant la position initiale de l’image cible :

La position initiale ne peut pas être aléatoire, il faut placer l’image cible avec un minimum de cohérence. Il ne suffit que de quelques dixièmes de plus ou de moins sur un des axes pour que le recalage ne fonctionne plus du tout. Cela s’explique par le fait que pour pouvoir recaler efficacement il faut trouver des similitudes, par conséquent, si les similitudes sont trop éloignées elles ne seront jamais repérées par l’algorithme, mis à part si la zone testée pour chaque point est super grande, mais d’autres problèmes viendront alors empêcher un recalage idéal. Il faut donc placer l’image cible de la meilleure façon possible pour un œil humain si on veut avoir le meilleur recalage possible.

Suivant les paramètres :

- Si on diminue la valeur de S le recalage ne bouge presque pas, ce qui semble logique vu que la zone testée pour le recalage n’est potentiellement pas assez grande pour trouver le bon recalage.
- Au contraire, si on augmente la valeur de S le recalage bouge beaucoup trop, la zone testée est trop grande ce qui fait que l’algorithme trouve des similarités là où il ne devrait pas y en avoir.
- Si on diminue la valeur de l le recalage effectué grâce aux distances dans la fonction *computeCorrespondences* sera moins fort. Si on l’augmente il sera beaucoup plus conséquent, il vaut donc mieux garder l entre 0,02 et 0,04.
- Le résultat observé avec les paramètres Ni et No est assez similaire à celui de S. Augmenter ces valeurs revient à augmenter la zone d’intensité prise sur le maillage dans la fonction *computeProfiles*. Ce qui revient a effectué des mesures SSD et NCC sur beaucoup plus de valeurs, vu que chaque calcul de somme prend en compte le nombre de valeurs déterminé par Ni et No. L’opposé est également vrai.