**TP Imagerie médicale 3D 3 (2 x 1,5 heures)**

**Benjamin Gilles – 13 novembre 2013**

*Ce TP a pour objectif de vous faire programmer un algorithme de recalage automatique de maillages surfaciques basé sur l'algorithme ICP.*

***Il est à rendre AVANT le jeudi 27 novembre par courrier électronique à :*** [***benjamin.gilles@lirmm.fr***](mailto:benjamin.gilles@lirmm.fr)

*Ce TP peut être effectué seul ou en binôme.*

*TP a priori sous* ***Linux****.*

*La bibliothèque CIMG (* [*http://cimg.sourceforge.net/*](http://cimg.sourceforge.net/) *) est déployée sur le parc informatique. Elle se trouve dans le répertoire : /net/local/CImg-1.5.1. Un sous-répertoire nommé "subsol" contient le fichier CImg.h*

*Le logiciel Fiji (* [*http://fiji.sc/*](http://fiji.sc/) *) est installé sur les machines.*

1. Visualiser les maillages avec Fiji (installé sur les machines) Plugins/3D Viewers/File/ImportWavefront (lit les .obj). Permet de voir les différences de morphologie.
2. Pour les Master 2 Informatique et les Master 2 STIC/santé qui le souhaitent, compléter le programme de recalage icp.cpp en utilisant l'algorithme ICP (obligatoire pour les Master 2 Informatique).

**A rendre** : compte-rendu d'une ou deux pages + sources

1. Pour les autres Master STIC/Santé, utiliser des programmes domaine public (exemple MeshLab cf. <http://sourceforge.net/apps/mediawiki/meshlab/index.php?title=Alignment>) ou MATLAB pour recaler les deux maillages avec une méthode fondée sur l'ICP.

**A rendre :** compte-rendu de minimum 3 pages expliquant précisément la procédure suivie, l'algorithme et le choix des paramètres.

**Annexe 1: description de l'algorithme ICP**

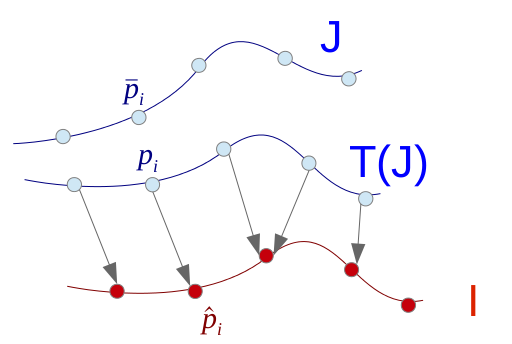
Cet algorithme sera vu en détail lors du cours du 8 novembre.

Le but est d 'aligner (de recaler) une surface **J** sur une surface **I** au moyen d'une transformation globale (rigide, affine ou similitude). Les pointsde **J** sont transformés selon la relation :

où **A** est une matrice 3x3 et t un vecteur 3x1, en 3d.

A chaque itération, une nouvelle transformation est calculée de façon à approximer au mieux des *correspondances* entre **T(J)** et **I**.

On note, le point de **I** correspondant au point(par exemple son point le plus proche). L'importance de chaque correspondance est pondéré par(typiquement entre 0 et 1).



Dans un premier temps, il s'agit d’implémenter le calcul des correspondancespar point le plus proche : est défini comme le point de **I** le plus proche du point de **T(J)**.

Dans un deuxième temps, le but du TP est de calculer la transformation optimale cherchant à minimiser la distance entre les points transformés et leur point correspondant:

Pour cela, on a besoin des deux matrices 3x3 de covariance suivantes :

calculée à partir des centroides et

En fonction du type de transformation (rigide, affine ou similitude), on calcule A :

- transformation affine :

- transformation rigide:

- similitude: , (note : la trace d'une matrice est la somme des termes diagonaux)

La translation optimale est obtenue par :

Après avoir établi les correspondances (1ere partie à compléter dans la fonction *Registration*), il reste a implémenter le calcul de , , Q, K et de s.

Dans un premier temps, les pondérations wi peuvent être laissées à 1. Après avoir fait marché le recalage, on vous propose d'implementer différentes pondérations vues en cours afin d'améliorer la qualité du recalage (notamment sur le modèle partiel femur\_f\_partial.obj).

Le programme se compile depuis un terminal avec la commande *Make* (vérifier que la librairie Cimg.h est présente dans le répertoire ou en dessous dans ../cimg/)

Le programme se lance avec la commande :

./icp modèlesource.obj modèlecible.obj

par default (en exécutant uniquement la commande .icp) , les modèles utilisés sont respectivement ./data/femur\_m.obj et ./data/femur\_f.obj.

Pour le recalage partiel, vous pouvez exécuter :

./icp ./data/femur\_m.obj ./data/femur\_f\_partial.obj

Pour récupérer les coordonnées des points lors du calcul des correspondances et des matrices, il faut utiliser les fonctions fournies dans la classe **Mesh**

Par exemple, il est possible de récupérer les coordonnées du point i Î**I** dans un vecteur (par exemple le vecteur *float P[3]*) en utilisant la fonction target.GetPoint(P,i).

De même, on récupère les coordonnée du point transformé grâce à source.GetPoint(P,i).

Pour les points non transformés (utilisés dans les matrices), on utilise source.GetPoint0(P,i).

**Annexe 2: description de l'API**

1. Mesh.h :

Fonctions de base de gestion de maillages :

struct MESH

{

CImgList<unsigned int> faces;

CImg<float> vertices0;

CImg<float> vertices;

CImg<float> correspondences;

CImg<float> normals;

CImg<float> weights;

inline unsigned int getNbPoints() ;

void getPoint0(float p[3], const unsigned int index) ;

void getPoint(float p[3], const unsigned int index) ;

void setPoint(const float p[3], const unsigned int index) ;

void getCorrespondence(float p[3], const unsigned int index) ;

void setCorrespondence(const float p[3], const unsigned int index) ;

void getNormal(float n[3], const unsigned int index) ;

float getWeight(const unsigned int index) ;

void setWeight(const float w, const unsigned int index) ;

bool LoadObj(const char\* filename);

void updateNormals();

void drawMesh(..) ;

void drawCorrespondences(.. );

void drawNormals(..);

void getTrace(..) ;

};

1. Mathematics.h :

Fonctions mathématiques de base pour les calculs ICP :

void Invert(float M\_inv[3][3],const float M[3][3]) ;

void Mult(float C[3][3],const float A[3][3],const float B[3][3]) ;

void Mult(float p[3],const float A[3][3],const float p0[3]) ;

int Jacobi(float M[3][3],float e[3],float E[3][3]) ; // compute M=E.e.E^T where e is diagonal

int ClosestRigid(float K[3][3],float R[3][3]) ; // compute R=K(K^TK)^{-1/2}

1. ICP.cpp

Boucle de visualisation :

int main(int argc,char \*\*argv)

{

{..} // load meshes from obj files

{..} // init 3d view parameters and rendering parameters

{..} // init display

while (!disp.is\_closed()) // main loop

{

if (init\_pose) {..} // Init camera pose

if (redraw) {..} // Rotate and draw 3d object

// Handle user interaction

if (disp.button() ..) { } // Mouse interaction

switch ( disp.key()) // keyboard interaction

{

case cimg::keyF.. : {..} break; // switch rendering mode

case cimg::keyR : {..} break; // reinit camera

case cimg::keyC : {..} break; // enable/disable visu of correspondences

case cimg::keyN : {..} break; // enable/disable visu of normals

case cimg::keyPADADD : {..} break; // change transformation type

case cimg::keySPACE : // one step of icp

{

// update global transformation

float A[3][3]={{1,0,0},{0,1,0},{0,0,1}}; float t[3]={0,0,0};

Registration(A,t,source,target,transformType);

// update points

for(unsigned int i=0;i<source.getNbPoints();i++)

{

float p0[3]; source.getPoint0(p0,i);

float p[3]; Mult(p,A,p0);

for(unsigned int j=0;j<3;j++) p[j]+=t[j];

source.setPoint(p,i);

}

disp.set\_key(); redraw = true;

break;

}

}

}

return 0;

}

1. Geometric registration :

Algorithme ICP à implémenter :

void Registration(float A[3][3], float t[3], MESH& source, MESH& target, const transformationType &transformType)

{

target.updateNormals();

source.updateNormals();

// compute correspondences

for(unsigned int i=0;i<source.getNbPoints();i++)

{

float p[3]; source.getPoint(p,i);

float cp[3]={p[0],p[1],p[2]};

**///\*\*\*\*\*\*\*\* TO BE COMPLETED \*\*\*\*\*\*\*\*\*\*\*\*///**

source.setCorrespondence(cp,i);

}

// weight correspondences and reject outliers

for(unsigned int i=0;i<source.getNbPoints();i++)

{

source.setWeight(1,i);

**///\*\*\*\*\*\*\*\* TO BE COMPLETED \*\*\*\*\*\*\*\*\*\*\*\*///**

}

// compute centroids

float c0[]={0,0,0}, c[]={0,0,0}, N=0;

**///\*\*\*\*\*\*\*\* TO BE COMPLETED \*\*\*\*\*\*\*\*\*\*\*\*///**

// fill matrices

float Q[][3]={{1,0,0},{0,1,0},{0,0,1}}, K[][3]={{1,0,0},{0,1,0},{0,0,1}},sx=3;

**///\*\*\*\*\*\*\*\* TO BE COMPLETED \*\*\*\*\*\*\*\*\*\*\*\*///**

// compute solution for affine part

{..}

// compute solution for translation

{..}

}