

Acquisition des mouvements d'une main humaine et restitution en environnement 3D

**Pascal LENOIS
Gilles MAIRE
Philippe PLAGNOL
Ludovic SMAL**

Sommaire

I.	La main humaine : sa biomécanique	4
II.	Les différents gants de saisie existants	5
1.	Les gants exo-squelettiques.....	5
2.	Les gants à encre conductrice (CyberGlove d’Immersion, PowerGlove de Matel)	6
3.	Les gants à fibre optique (Data Glove 5 de 5DT).....	7
4.	La réalisation du gant.....	8
III.	Première modélisation de la main	9
1.	Aperçu du modèle 3D	9
IV.	Seconde modélisation de la main	11
1.	Objet 3ds	11
2.	Interaction avec les touches.....	11
3.	Valeus limites des articulations	12
4.	Positionnement des angles	13
5.	Gestion de la caméra.....	15
6.	Menu sur clic droit.....	15
a)	WireFrame	16
b)	Lumière	16
c)	Faces cachées.....	17
7.	Les lumières.....	18
a)	Les modèles de lumière.....	18
b)	Création d'une lumière	19
8.	Fenêtre d’information sur les angles.....	22
9.	Rendu de la peau et des pliures	24
V.	Partie Electronique.....	26
1.	la carte de traitement du signal	26
2.	La carte de démultiplexage.....	27
VI.	Les DLL de communication.....	30
1.	La connexion au PC	30
2.	DLL de communication	31
3.	DLL de rendu graphique	32
VII.	Etude du FlexSensor	33
VIII.	Le programme Glove Manager.....	35
	Annexe : Programme du PIC	36

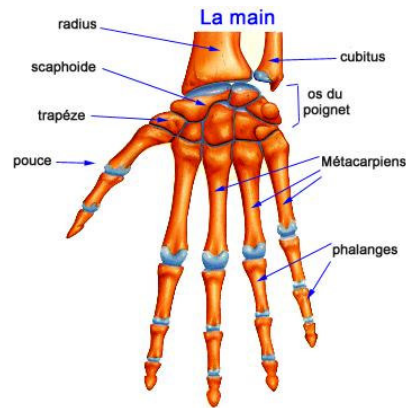
L'acquisition des mouvements de la main trouve actuellement de nombreuses applications dans des domaines tels que la réalité virtuelle, la chirurgie à distance, etc.

Nous avons choisi comme sujet de TPED la réalisation d'un gant de capture et la restitution virtuelle des mouvements relevés.

Notre étude se décompose en trois phases : dans en premier temps, nous devons réaliser une structure physique capable de détecter et de mesurer les différentes flexions de la main, puis créer une interface électronique faisant le lien entre les grandeurs physiques mesurées et le programme informatique, la dernière partie est une représentation visuelle de l'acquisition : nous comptons reproduire dans les grandes lignes la main humaine : le but étant qu'à tout moment, la configuration spatiale de la main virtuelle soit identique à celle de la main humaine sur laquelle se fera l'acquisition.

I. La main humaine : sa biomécanique

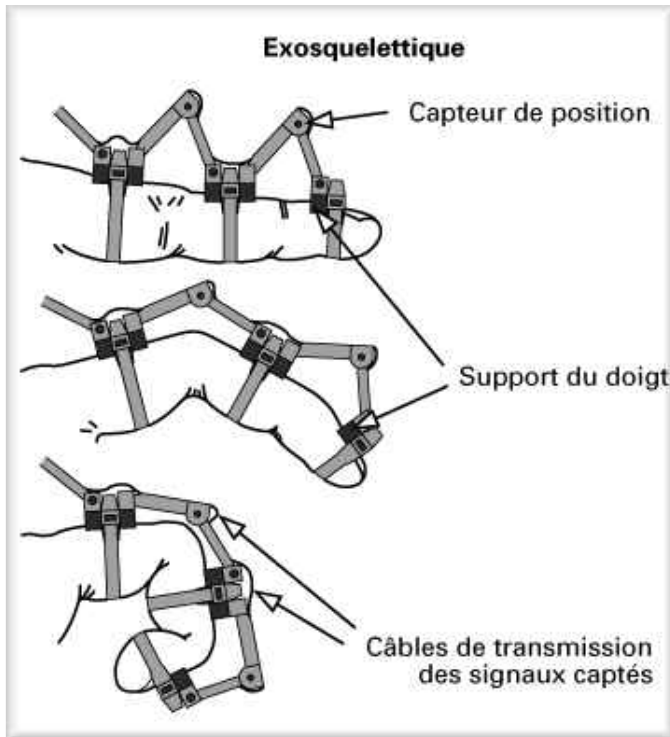
La main est un organe articulé possédant 22 degrés de liberté : deux degrés de liberté (abduction / adduction et flexion / extension) pour les articulations métacarpo-phalangiennes, une autre (flexion / extension) au niveau des articulations inter-phalangiennes de chaque doigt. L'articulation carpo-métacarpienne du pouce possède 3 degrés de liberté : abduction / adduction, flexion / extension et une pseudo-rotation due à l'incongruité entre les os du carpe et la base du métacarpe du pouce, et au relâchement des ligaments les reliant.



II. Les différents gants de saisie existants

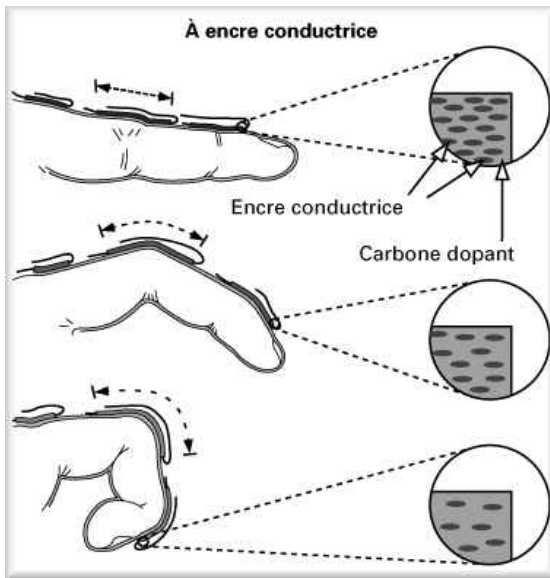
1. Les gants exo-squelettiques

La structure exo-squelettique est composée de segments articulés installés sur la main. Elle comporte des capteurs d'angles (potentiomètres, etc.) situés aux articulations de l'exosquelette qui retournent les données relatives à la flexion du doigt. Cette méthode peu onéreuse a l'inconvénient d'être encombrante et clairement ressentie par son utilisateur.



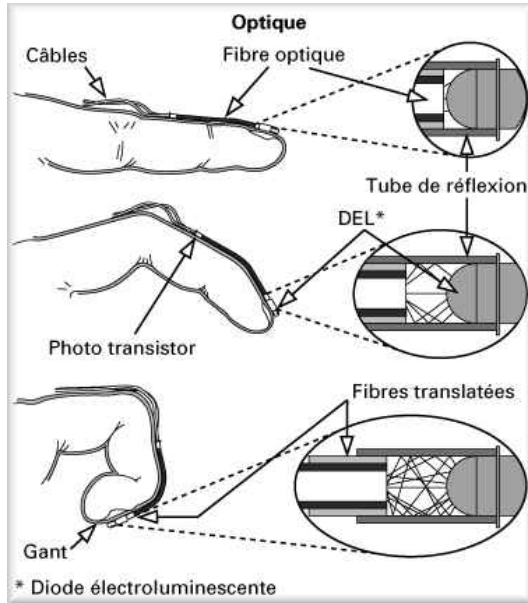
2. Les gants à encre conductrice (CyberGlove d'Immersion, PowerGlove de Matel)

Le principe de fonctionnement des gants contenant de l'encre repose sur des capteurs composés de deux couches d'encre conductrice d'électricité (contenant des particules de carbone) posés sur un substrat. L'augmentation de la distance entre les particules de carbone lors de la flexion des doigts correspond à un accroissement de la résistance qui se trouve corrélée par calibrage à une valeur angulaire déterminée : c'est actuellement la solution la plus utilisée et qui permet de mesurer jusqu'à 22 angles. Son principe permet de réellement coller à la peau et de se faire oublier par son utilisateur. Ce principe requiert cependant un calibrage avant utilisation (main ouverte / main fermée).



3. Les gants à fibre optique (Data Glove 5 de 5DT)

Lorsqu'on courbe une fibre optique, on constate une atténuation du faisceau de la lumière qui la traverse. Cette atténuation est proportionnelle à la flexion. On peut ainsi déterminer la flexion du doigt. C'est une solution à moindre coût mais dont la mise en pratique est complexe.



4. La réalisation du gant

Rappelons que notre étude vise à mettre en avant les solutions ayant le meilleur rapport qualité/prix (dans la limite de nos moyens et compétences). Nous excluons les structures exo-squelettiques pour leur configuration fondamentalement gênante et volumineuse et les structures à fibre optique trop complexes à mettre en place.

Lorsque nous avons débuté la recherche de composants permettant l'acquisition de flexions, nous nous sommes aperçu que ces derniers n'existaient pas sur le marché : les lamelles à encre conductrice utilisées dans le CyberGlove d'Immersion étaient brevetées et non commercialisées indépendamment. La solution que nous avons choisi d'adopter était de simuler les axes de liberté de la main par des potentiomètres que nous aurions pu par la suite monter en exosquelette autour de la main. C'est septembre que nous avons remarqué une démocratisation d'un capteur que nous avons exclu pour un motif de taille : le FlexSensor, ce composant développé par Spectra Symbol était utilisé dans le Power Glove de Matel et utilisait la technologie de l'encre conductrice (faisant varier selon la courbure la résistance globale).

Nous avons contacté Spectra Symbol qui nous a envoyé plusieurs échantillons : nous nous sommes aperçu de l'évolution technologique qu'avait subit ce composant, et de la possibilité de découpage et de modelage du composant. Le FlexSensor nous a pourtant posé un problème : la texture de la partie conductrice supérieure ne permettait pas de fixer de façon simple des fils conducteurs.

Voici la procédure que nous avons suivi pour rendre solidaires le Flex et le fils :

- percer deux trous dans la partie à connecter
- faire passer le fils dénudé (allonger sur la surface conductrice)
- placer de la colle d'argent Loctite 3880
- placer au four à 150° pendant 6 min

après avoir maîtriser cette technique de collage, nous avons trouvé un procédé permettant de simplifier la mise en place des Flexs sur les doigt : le fait de placer les deux fils de mesure aux extrémité et de placer le fils de référence au centre : un seul Flex permet ainsi de mesurer deux axes de liberté.

Nous avons en tout placé 14 axes de liberté sur le gant, le composant et le gant que nous avons choisi posant des problèmes physiques (limites de taille, de flexion, etc...)

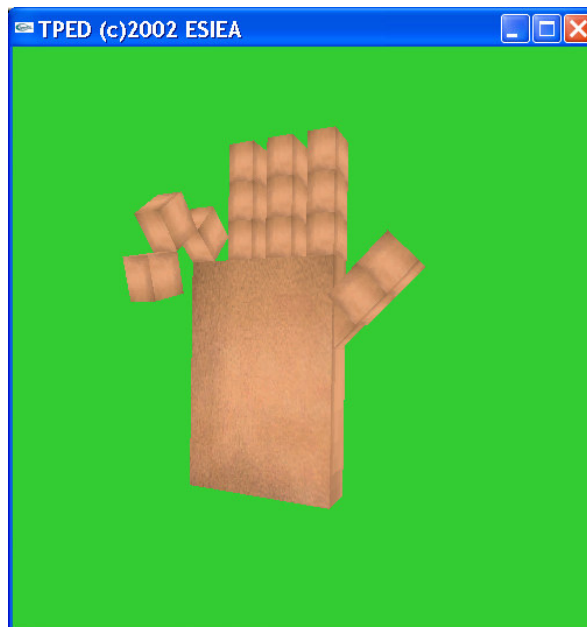


III . Première modélisation de la main

Nous utilisons pour des raisons pratiques le langage OPENGL pour la partie infographie et notamment les outils GLUT simplifiant la programmation.

Avant de vouloir modéliser une main 3D « réaliste » nous avons plutôt préféré prendre un modèle de base relativement simple. Celui-ci est composé de cubes, la représentation est éloignée de la main humaine mais cela nous a néanmoins permis d'apprendre à programmer en OPENGL et de comprendre les différentes transformations à appliquer aux objets pour réussir à restituer des mouvements réalistes.

Nous n'avons pas réalisé de lien entre la partie électronique et la partie infographie, les différents mouvements sont commandés à partir du clavier. Nous obtenons une main très schématique pour laquelle chaque doigt peut se plier ou se tendre.



1. Aperçu du modèle 3D

Les principales questions étaient :

- Comment réaliser un objet qui soit "animable" sous VC++6/Glut 3.7 ?
- Doit-on partir sur des polygones "liés" et les faire tourner suivant un axe de rotation ?

Pour modéliser la main, deux choix ont été faits :

- Créer avec les primitives de Glut ou OpenGL la main via des objets de base.
- Créer sous 3DS la main (ou Poser et importer sous 3DS) et l'exporter en ASE (format ASCII de 3DS) puis créer un programme en C++ pour stocker les objets dans une structure de données.

Les travaux ont donc commencé sur un objet basique composé de 3 cubes ayant 3 axes de rotation libre. L'évolution vers un modèle « paume+4 doigts » permet de mettre en place des contraintes sur les angles de rotation (angle bloqué à 90°). Le pouce a ensuite été ajouté au modèle.

Ce modèle permettait de tester différentes configurations dans l'espace mais manquait cruellement de réalisme. Il fallait un objet modélisé représentant une main.

Cette phase constitue la base de la seconde modélisation de la main.

IV. Seconde modélisation de la main

Notre travail s'est donc agencé de la façon suivante :

- Objet 3ds
- Interaction avec les touches
- Angles (positionnement, contraintes)
- Gestion de la caméra
- Menu sur clic droit
- Modèle de lumière
- Fenêtre d'information sur les angles

1. Objet 3ds

Pour obtenir une main réaliste, nous avons cherché un squelette dessiné sous 3DS dont chaque os est un objet.



Le rendu est plus que convainquant !

2. Interaction avec les touches

L'une des premières étapes a été d'implémenter l'utilisation du clavier pour pouvoir contrôler les mouvements de la main. Cela nous a permis de pouvoir tester la réactivité des objets sans avoir recours au gant.

Nous avons assigné 8 touches par doigt et 6 touches pour les mouvements du poignet. En effet, sur chaque doigt, il est possible d'agir sur 3 phalanges et de faire pivoter latéralement sa base. Les actions sur le poignet se traduisent par une rotation suivant les 3 axes du repère.

Nous avons donc défini la série de touches suivante :

Doigt	sens	Flexion 1	Flexion 2	Flexion 3	Pivot
poignet	-				y/h/n
poignet	+				Y/H/N
pouce	-	a	Q	w	,
pouce	+	A	Q	W	?
index	-	z	S	x	k
index	+	Z	S	X	K
majeur	-	e	D	c	o
majeur	+	E	D	C	O
annulaire	-	r	F	v	p
annulaire	+	R	F	V	P
auriculaire	-	t	G	b	m
auriculaire	+	T	G	B	M

Ces touches ont été choisies pour être le plus ergonomique possible, on remarquera que les touches correspondant à un doigt sont les unes au dessus des autres. De plus, la rotation dans le sens opposé se fait simplement en combinant la touche 'Shift' à la touche utilisée.

Nota : Les mouvements de pivot ne sont pas gérés par le gant.

3. Valeus limites des articulations

Nous avons ensuite déterminé pour plus de réalisme des contraintes sur les rotations des phalanges. Ces valeurs sont tirées d'articles médicaux de référence :

Doigt	Flexion 1	Flexion 2	Flexion 3	Pivot
pouce	$-10 \leq \alpha \leq 30$	$0 \leq \alpha \leq 60$	$0 \leq \alpha \leq 90$	$-10 \leq \alpha \leq 100$
index	$-30 \leq \alpha \leq 90$	$0 \leq \alpha \leq 110$	$0 \leq \alpha \leq 90$	$-20 \leq \alpha \leq 10$
majeur	$-30 \leq \alpha \leq 90$	$0 \leq \alpha \leq 110$	$0 \leq \alpha \leq 90$	$-15 \leq \alpha \leq 15$
annulaire	$-30 \leq \alpha \leq 90$	$0 \leq \alpha \leq 110$	$0 \leq \alpha \leq 90$	$-10 \leq \alpha \leq 20$
auriculaire	$-30 \leq \alpha \leq 90$	$0 \leq \alpha \leq 110$	$0 \leq \alpha \leq 90$	$-10 \leq \alpha \leq 40$

Exemple du code associé à la flexion 1 de l'index :

```
//Flexion -1
    case 'z':
        if (g_3DModel.pObject[5].angle.x > -90)
            g_3DModel.pObject[5].angle.x -= 5;
        break;
//Flexion +1
    case 'Z':
```

```

if (g_3DModel.pObject[5].angle.x < 30)
    g_3DModel.pObject[5].angle.x += 5;
break;

```

Nota : On remarque que les valeurs sont opposées aux contraintes de référence à cause du changement de repère (Cf. 'positionnement des angles')

4. Positionnement des angles

La main modélisée sous 3ds est constituée de 20 objets correspondant aux os du poignet et à chaque phalange de chaque doigt. Pour information, nous avons extrait les cotes de chaque objet ainsi que le nombre de sommets et de faces.

Ensuite, il a fallu déterminer les positions des points de rotation de chaque objet avec le logiciel 3ds max dans le repère (x, y, z).

Nous nous sommes aperçut que le repère que nous avons en C++ ne correspondait pas au repère sous 3ds. La transformation effectuée est la suivante :

3ds	C++
x	x
y	-z
z	y

Le tableau contenant l'ensemble des résultats est le suivant :

Doigt	Num. objet	Nom	Taille x	Taille y	Taille z	Nb. Sommets	Nb. Faces	x 3ds	y 3ds	z 3ds	x C++	y C++	z C++
poignet	0	Carpals	1,916	1,25	1,185	338	644	0	-4	0	0	0	4
pouce	1	Mcarpal1	1,31	1,707	0,614	80	156	-0,9	-3	-0,2	-0,9	-0,2	3
pouce	2	ProxPhal1	0,515	1,223	0,552	66	128	-1,86	-1,45	-0,35	-1,86	-0,35	1,45
pouce	3	DistPhal1	0,293	0,959	0,433	49	94	-2,09	-0,27	-0,43	-2,09	-0,43	0,27
index	4	Mcarpal2	0,786	2,726	0,579	106	208	*	*	*	*	*	*
index	5	ProxPhal2	0,543	1,554	0,486	78	152	-0,7	0	0	-0,7	0	0
index	6	MidPhal2	0,49	0,96	0,535	42	80	-0,9	1,46	0	-0,9	0	-1,46
index	7	DistPhal2	0,348	0,618	0,398	27	50	-0,97	2,27	-0,3	-0,97	-0,3	-2,27
majeur	8	Mcarpal3	0,617	2,732	0,579	106	208	*	*	*	*	*	*
majeur	9	ProxPhal3	0,526	1,768	0,486	78	152	0	0	0	0	0	0
majeur	10	MidPhal3	0,446	1,1	0,535	42	80	0	1,85	0	0	0	-1,85
majeur	11	DistPhal3	0,347	0,725	0,398	27	50	-0,1	2,86	-0,27	-0,1	-0,27	-2,86
annulaire	12	Mcarpal4	0,738	2,637	0,579	106	208	*	*	*	*	*	*
annulaire	13	ProxPhal4	0,546	1,72	0,486	78	152	0,65	0	0	0,65	0	0
annulaire	14	MidPhal4	0,435	1,054	0,535	42	80	0,86	1,65	0	0,86	0	-1,65
annulaire	15	DistPhal4	0,335	0,707	0,398	27	50	0,78	2,65	-0,27	0,78	-0,27	-2,65
auriculaire	16	Mcarpal5	0,769	2,167	0,474	106	208	*	*	*	*	*	*
auriculaire	17	ProxPhal5	0,553	1,417	0,398	78	152	1,22	-0,45	0	1,22	0	0,45
auriculaire	18	MidPhal5	0,37	0,861	0,438	42	80	1,5	0,77	0	1,5	0	-0,77
auriculaire	19	DistPhal5	0,275	0,584	0,326	27	50	1,48	1,58	-0,23	1,48	-0,23	-1,58

Exemple du code associé à l'index :

L'index est constitué de quatre phalanges dont trois possèdent des contraintes de rotation. On dessine donc les objets (4, 5, 6 et 7) via la fonction DessinePhalange(). De plus, on utilise les fonctions glPushMatrix() et glPopMatrix() pour lier les points de rotations les uns aux autres. Dans la pratique, cela permet par exemple de faire bouger la dernière phalange du doigt par rapport à la position de la précédente.

```
DessinePhalange(4);
glPushMatrix();
    // Faît tourner les doigts autour de leurs axes
    glTranslatef(-0.7, 0, 0);
    glRotatef(g_3DModel.pObject[5].angle.x, 1.0, 0, 0);
    glRotatef(g_3DModel.pObject[5].angle.y, 0, 1.0, 0);
    glTranslatef(0.7, 0, 0);
    glPushMatrix();
        DessinePhalange(5);
    glPopMatrix();

    glTranslatef(-0.9, 0, -1.46);
    glRotatef(g_3DModel.pObject[6].angle.x, 1.0, 0, 0);
    glRotatef(g_3DModel.pObject[6].angle.y, 0, 1.0, 0);
    glTranslatef(0.9, 0, 1.46);
    glPushMatrix();
        DessinePhalange(6);
    glPopMatrix();

    glTranslatef(-0.97, -0.3, -2.27);
    glRotatef(g_3DModel.pObject[7].angle.x, 1.0, 0, 0);
    glRotatef(g_3DModel.pObject[7].angle.y, 0, 1.0, 0);
    glTranslatef(0.97, 0.3, 2.27);
    glPushMatrix();
        DessinePhalange(7);
    glPopMatrix();
glPopMatrix();
```

Nota : La fonction DessinePhalange() se charge de l'affichage de l'objet 3ds.

5. Gestion de la caméra

Pour naviguer dans la scène, nous avons implémenté la fonction `gluLookAt()`. Cette fonction positionne une caméra dans la scène qui vise un point défini dans l'espace. Elle prend 9 paramètres en entrée qui sont des triplets de valeurs (x, y, z). Le premier triplet correspond à la position de la caméra, le second correspond à la position du point à viser et le dernier au vecteur directeur de la scène.

Nous avons variabilisé la position de la caméra pour pouvoir agir à tout moment sur sa position et nous obtenons la fonction suivante :

```
gluLookAt(CamX, CamY, CamZ, 0, 0, 0, 0, 1, 0);
```

Les touches de contrôle de la caméra sont les suivantes :

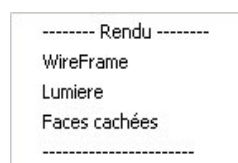
Touche	Fonction
1	--
2	bas
3	zoom out
4	gauche
5	init
6	droite
7	--
8	haut
9	zoom in
+	rotation +
-	rotation -

Il est important d'avoir toujours à l'esprit que la caméra **vise** toujours le point (0, 0, 0) ce qui engendre des mouvements parfois illogique. Pour exemple, si l'on zoom sur l'objet, ce dernier grossit plus on approche du centre du repère mais dès que l'on dépasse le centre du repère, la caméra se retourne pour voir l'objet tout en s'éloignant. De même, lorsque l'on effectue une translation, l'objet est toujours visé par la caméra ce qui nous donne l'impression de tourner autour. Une touche dite d'initialisation (touche 5) est donc à la disposition de l'utilisateur pour repositionner la caméra à sa position d'origine.

Enfin, les touches + et - permettent de faire tourner le repère lié à l'objet.

6. Menu sur clic droit

Afin de pouvoir paramétrer simplement l'application en cours d'exécution nous avons mis en place un système de menu, visible lors d'un clic droit sur la souris.

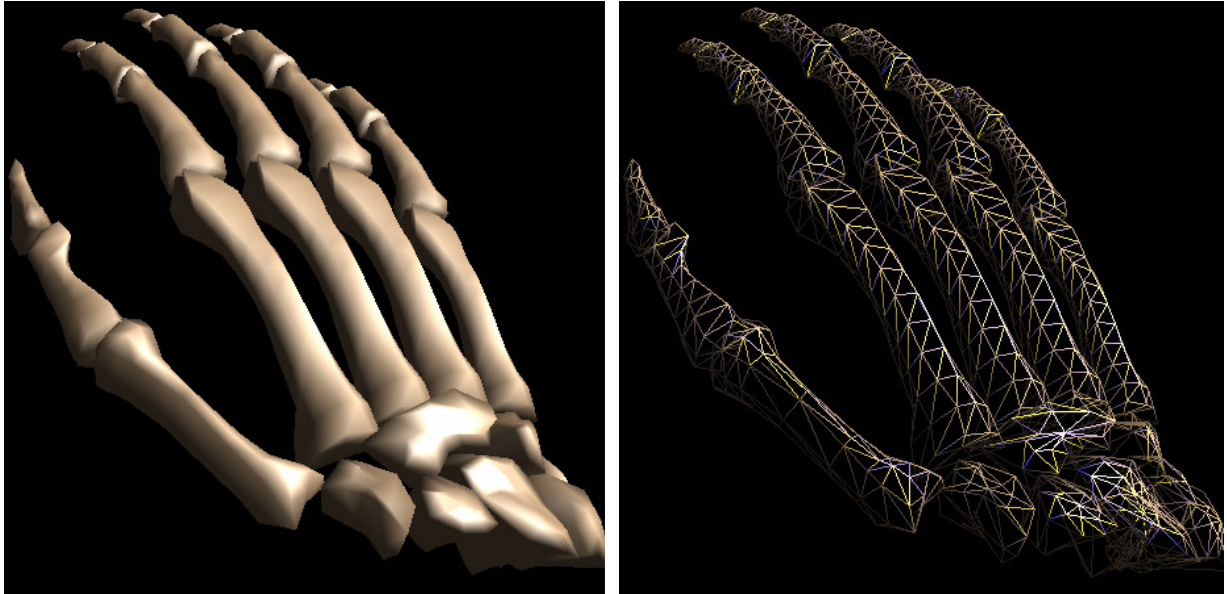


Le menu qui s'affiche permet de gérer trois options de rendu :

a) WireFrame

Cette option permet d'afficher la main en mode fil de fer, c'est-à-dire que les polygones sont remplis ou non.

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    ou
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```



b) Lumière

Cette option active ou désactive la source lumineuse.

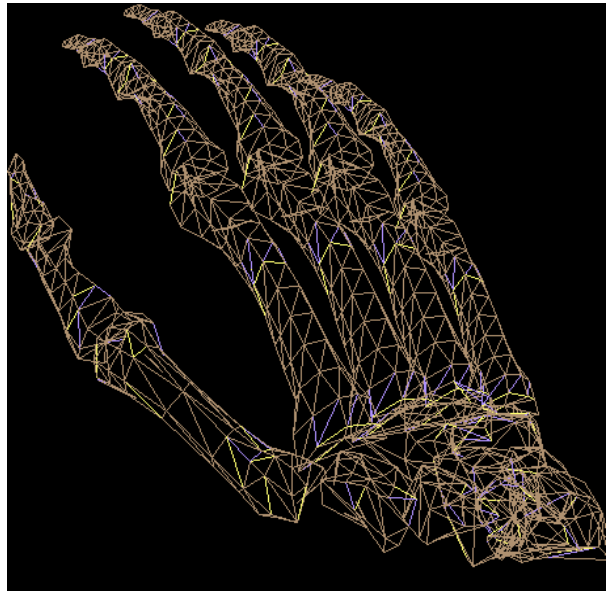
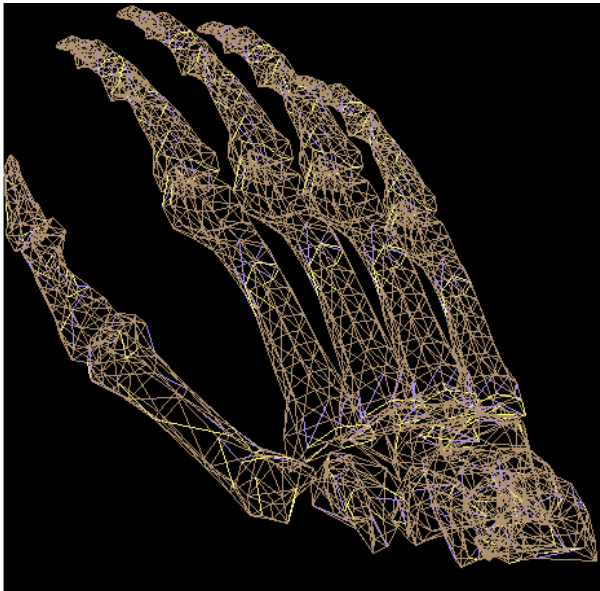
```
glDisable(GL_LIGHTING);
glDisable(GL_LIGHT0);
glDisable(GL_COLOR_MATERIAL);
    ou
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
```



c) Faces cachées

Cette option active ou désactive la gestion des faces cachées. Si elle est activée seule les faces visibles sont affichées ce qui est un gain de temps pour les traitements.

```
glDisable(GL_CULL_FACE);  
ou  
glEnable(GL_CULL_FACE);
```



7. Les lumières

L'aspect 3D des objets est souvent visible que lorsqu'ils sont éclairés par une source de lumière. Par exemple, une sphère non éclairée n'est qu'un disque.

a) Les modèles de lumière

OpenGL nous offre la possibilité d'utiliser 8 lumières individuelles qui seront représentées par `GL_LIGHT0`, `GL_LIGHT1`, `GL_LIGHT2`...`GL_LIGHT7`. Il est possible de changer la couleur de chaque lumière, les déplacer, changer leur intensité, leurs propriétés réfléchissantes.

Voici les différents types d'éclairages :

Ambiante

L'éclairage ambiant correspond à une lumière dispersée par l'environnement. Sa source est impossible à déterminer. Quand celle-ci frappe une surface elle est dispersée de manière égale dans toutes les directions.

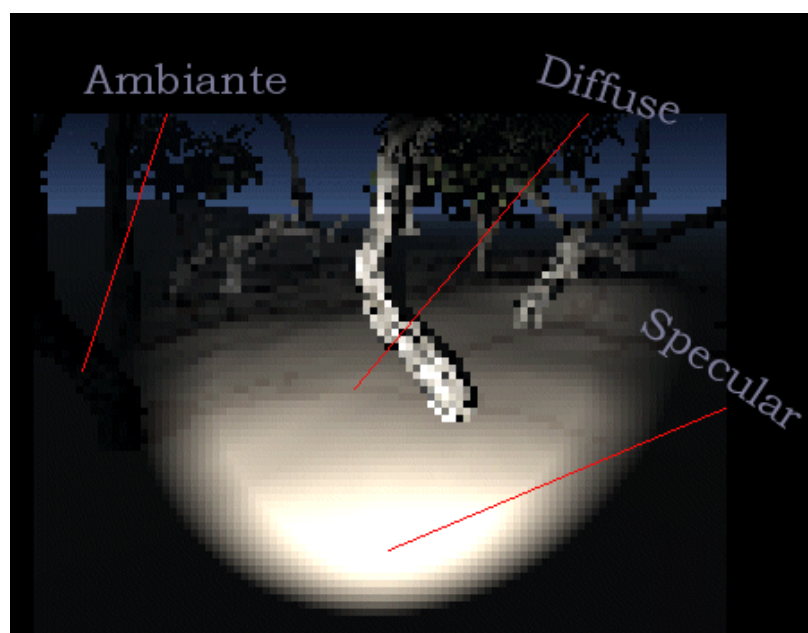
Diffuse

Cette lumière provient d'une source particulière et est aussi dispersée de manière égale dans toutes les directions.

Spéculaire

Comparable à la brillance d'une surface comme le métal, un miroir ou autres surfaces ayant des propriétés très réfléchissantes.

Démonstration graphique :

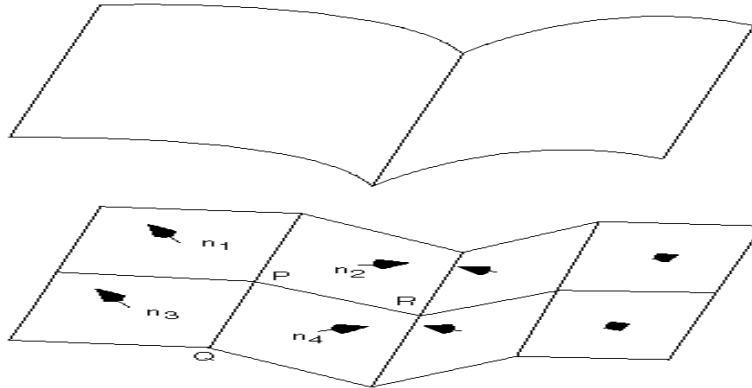


b) Création d'une lumière

Les normales de surfaces

Le plus complexe à comprendre dans l'utilisation des lumières en 3D est de déterminer la normale des facettes d'un objet. Qu'est-ce qu'une NORMALE de surface ?

Une normale est en faite un vecteur déterminant la position de l'objet par rapport aux sources de lumière lui étant soumis :



Les flèches noires sont toutes des normales de surfaces donc elles sont perpendiculaires à leur facette. Donc, pour que la lumière soit bien réfléchi sur chaque facette de vos primitives, il ne faut jamais oublier de spécifier chaque normales de ces facettes. Si on ne les spécifie pas alors on obtient un éclairage non-plausible et nous dirons même qu'elle n'aura aucun sens.

De plus il ne faut jamais oublier que la Normale doit être donnée sous forme unitaire donc sa norme doit être égal à 1. Si on ne veut pas les normaliser soi même OpenGL peut le faire mais au dépend de la rapidité de l'application : `glEnable(GL_NORMALIZE)` ou `glEnable(GL_RESCALE_NORMAL)`.

Créer et positionner la ou les sources de lumières

La commande `glLightfv()` permet entre autre de spécifier l'emplacement de la lumière. Comme c'est la première lumière, elle sera représentée par la constante `GL_LIGHT0`. Une fois que les caractéristiques de la lumière sont définies, il faut l'activer avec la commande `glEnable(GL_LIGHT0)`. Il ne faut pas oublier que plus on ajoute de lumière (`GL_LIGHT0`, `GL_LIGHT1`, `GL_LIGHT2...`) à notre scène et plus les performances de l'application seront diminuées.

Propriétés de la matière réfléchissant à la lumière

La lumière ce n'est pas tout, car nous devons spécifier qu'elle couleur est absorbée par l'objet recevant la source de lumière. C'est la manière dont la lumière réfléchi sur l'objet.

Chaque objet réfléchi une couleur dont l'ambiante, la diffuse et la spéculaire. Il est possible de définir la brillance de cette réflectivité. Pour la spécification de réflectivité de la matière, la commande est la suivante :

glMaterialf(GLenum face, GLenum pname, GLfloat param);

- Le paramètre face spécifie la ou lesquelles des facettes d'un objet seront touchées par la lumière, il peut prendre une des trois valeurs suivantes : GL_FRONT, GL_BACK, ou GL_FRONT_AND_BACK.
- Le paramètre pname sera expliqué dans le tableau ci-bas.
- Le paramètre param est un pointeur sur un tableau contenant les informations sur les modifications.

glMaterialf
Paramètre pname

Paramètre	Description
GL_AMBIENT	Couleur ambiante de la matière
GL_DIFFUSE	Couleur diffuse de la matière.
GL_AMBIENT_AND_GL_DIFFUSE	Couleur ambiante et diffuse de la matière.
GL_SPECULAR	Couleur spéculaire de la matière.
GL_SHININESS	Exposant spéculaire [0..128].
GL_EMISSION	Couleur émissive de la matière.
GL_COLOR_INDEXES	Index des couleurs ambiante, diffuse et spéculaire.

Création des sources de lumière

Une lumière comporte plusieurs propriétés comme : sa couleur, sa position, sa direction... Pour modifier les propriétés d'une lumière quelconque, on utilise la commande suivante :

*glLightfv(GLenum light, GLenum pname, const GLfloat *params)*

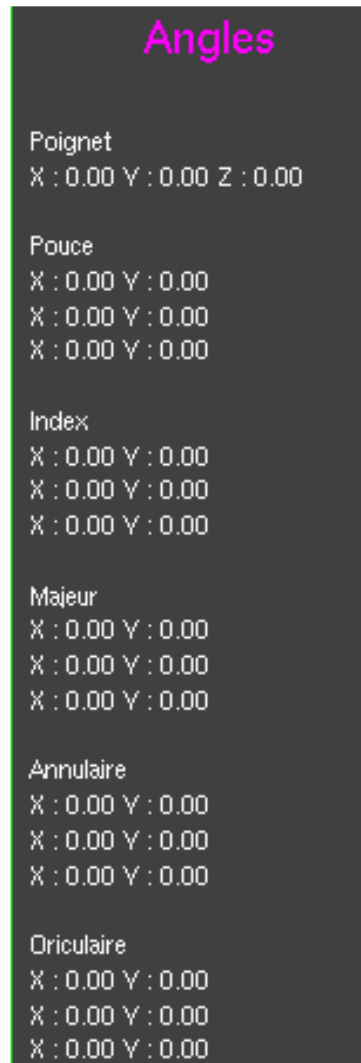
- Le paramètre light spécifie la lumière à modifier (GL_LIGHT0...GL_LIGHT7).
- Le deuxième paramètre pname est expliqué dans le tableau ci-bas.
- Le dernier paramètre params est un pointeur vers un tableau, contenant des valeurs, sur le paramètre de la lumière que l'on désire modifier.

glLightfv
Paramètre pname

Paramètre	Description
GL_AMBIENT	Intensité ambiante de la lumière
GL_DIFFUSE	Intensité diffuse de la lumière.
GL_SPECULAR	Intensité spéculaire de la lumière.
GL_POSITION	(X, Y, Z, W) position de la lumière dans l'espace.
GL_SPOT_DIRECTION	(X, Y, Z) Direction du faisceau lumineux de la lumière. La direction est spécifiée par un vecteur.
GL_SPOT_EXPONENT	Exposant du faisceau lumineux.
GL_SPOT_CUTOFF	Angle de coupure du faisceau lumineux.
GL_CONSTANT_ATTENUATION	Facteur d'atténuation constant.
GL_LINEAR_ATTENUATION	Facteur d'atténuation linéaire.
GL_QUADRATIC_ATTENUATION	Facteur d'atténuation quadratique.

8. Fenêtre d'information sur les angles

Afin de valider le bon fonctionnement des mouvements liés au gant ou à l'action des touches du clavier il est nécessaire d'afficher les valeurs des angles des différents os de la main.

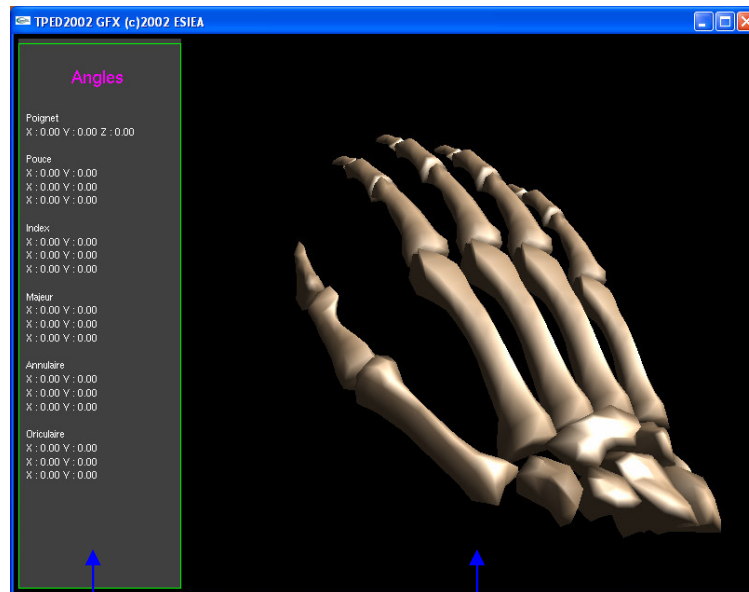


Le problème principal que nous avons rencontré lors de la mise en place de cette partie est la possibilité d'afficher les informations dans la même fenêtre que celle contenant la main. Et sans que ces informations ne soient soumises aux diverses rotations de la caméra.

Nous avons en fait créé une seconde fenêtre intégrée à la première et ayant ses propres procédures de gestion de l'affichage :

```
winIdSub = glutCreateSubWindow (winIdMain, 5, 5, 175, SCREEN_HEIGHT - 10);
glutDisplayFunc (subDisplay);
glutReshapeFunc (subReshape);
```

① WinIdMain correspond au handle¹ de la fenêtre principale.



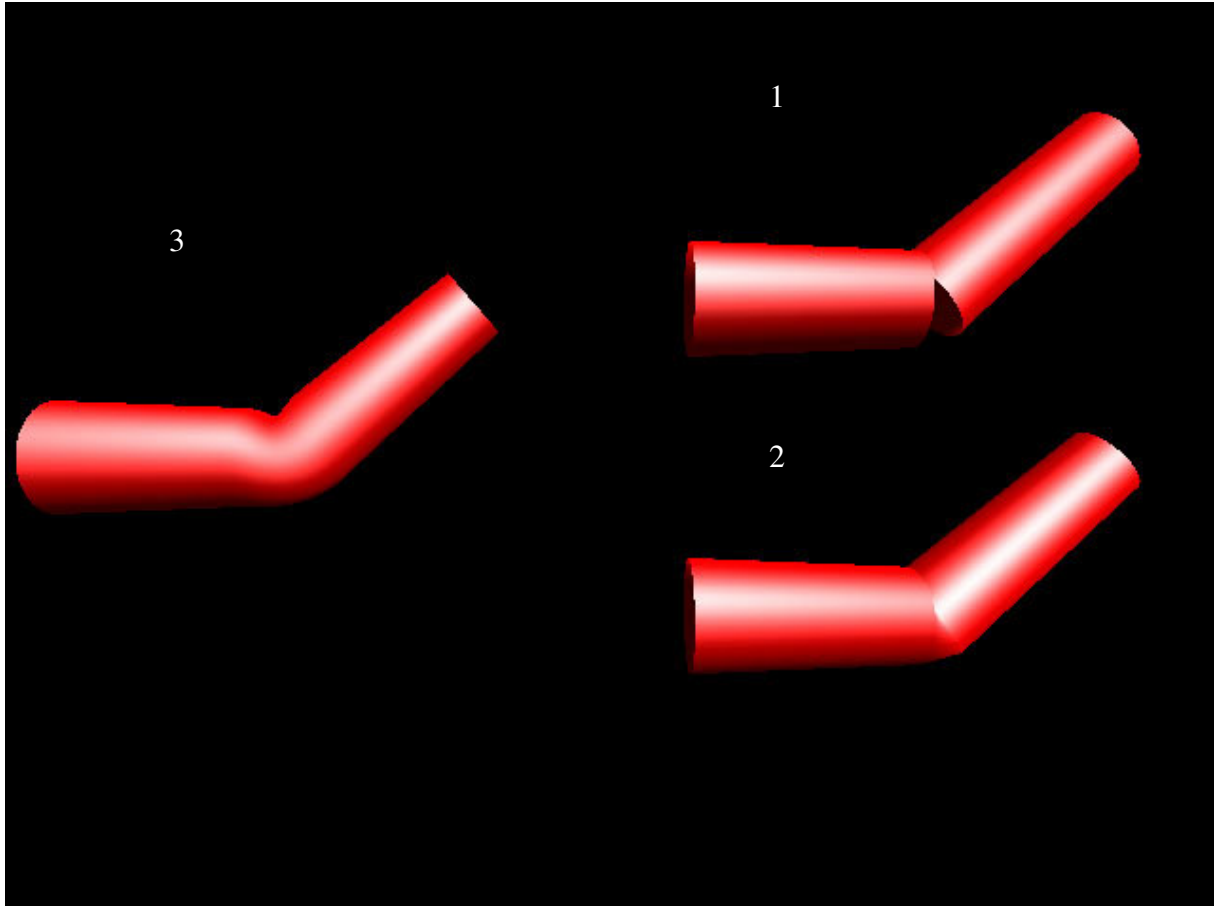
Fenêtre d'information

Fenêtre principale

¹ Valeur numérique identifiant un objet informatique (e. g. une fenêtre ou un fichier), et permettant donc sa « manipulation » (to handle en anglais).

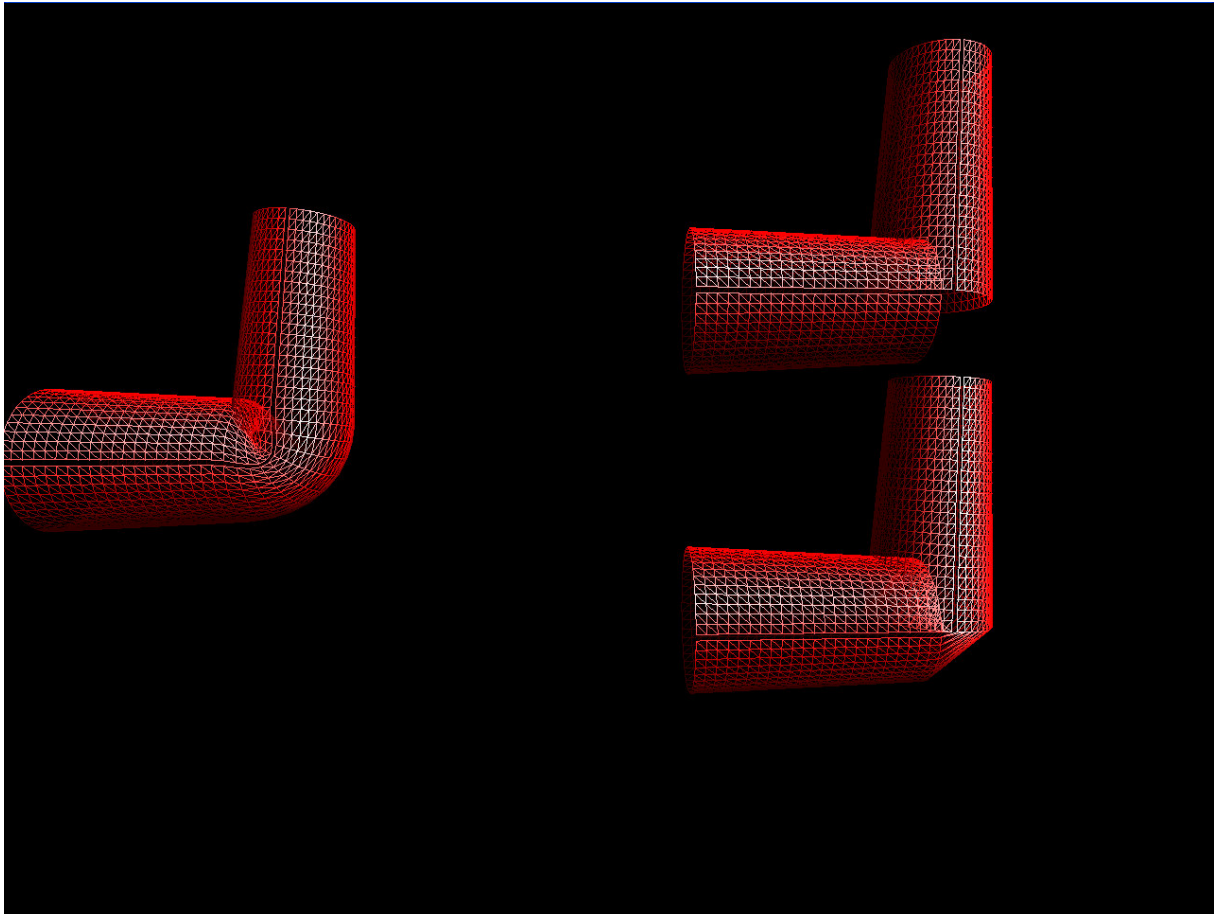
9. Rendu de la peau et des pliures

Nous n'avons pas eu le temps de finaliser le rendu de la main en lui appliquant une peau. Néanmoins, nous avons trouvé des pistes intéressantes comme par exemple l'utilisation de « vertex weight » pour simuler le rendu de pliures. La capture d'écran ci-dessous montre un exemple basique de modélisation :



Il s'agit en fait de lier deux objets (Cf. 1) par leur bord avec un ensemble de facettes suivant une droite dans un premier temps (Cf. 2) puis suivant une courbe (Cf. 3) en augmentant le nombre de facettes pour un rendu plus réaliste.

On comprend mieux le mécanisme lorsque l'on utilise le mode fil de fer sur la scène :



Lorsque les deux objets sont éloignés, les bords extérieurs sont étirés alors que les bords intérieurs sont rapprochés.

Le rendu est alors très proche des pliures de la peau entre deux phalanges.

Nous avons eu trois principaux problèmes pour mettre en œuvre cette solution :

- la liaison d'un objet de type vertex weight avec un objet complexe,
- le rendu de la paume semblait infaisable avec cette méthode à cause de la complexité des pliures,
- l'incompatibilité avec l'ensemble des cartes graphiques du marché car ces fonctionnalités font appels à des extensions d'OpenGL supportées par les cartes graphiques Nvidia (à partir de la gamme Geforce). Les cartes Matrox sont elles incompatibles et nous n'avons pas pu tester sur une carte graphique ATI de dernière génération.

Source : http://developer.nvidia.com/view.asp?IO=GL_EXT_vertex_demo

V. Partie Electronique

La partie électronique consiste à établir un lien fiable entre les capteurs utilisés et le programme de la main virtuelle. Nous l'avons séparé en deux parties : une carte mère qui ne devait pas changer comprenant un PIC porteur d'outils comme un convertisseur A/N, une liaison série,... et une deuxième, la carte de démultiplexage, assurant le lien entre les capteurs et le PIC.

1. la carte de traitement du signal

Nous devons réaliser un système permettant l'acquisition de 22 entrées analogiques sur un PC et notre bonne maîtrise du PIC16F877, nous a très rapidement incité à réaliser deux cartes. Une première portant ce microprocesseur de MICROCHIP™ ayant pour fonction d'assurer la transmission des données vers l'ordinateur et de réaliser directement le traitement du signal (les ports de communication sont trop lents pour que cette tâche soit réalisée sur le PC), il possédait également des outils nous permettant de faire face à un grand nombre d'imprévus : c'est un composant très polyvalent.

Ci-dessous la carte traitement du signal dont les caractéristique sont les suivantes :

Processeur 16F877 à 20 Mhz : puissance de Calcul 20 MIPS
E/S : 7 entrée analogique et 2 Sortie sur un connecteur 12 broches
Communication RS-232
Programmation ISP

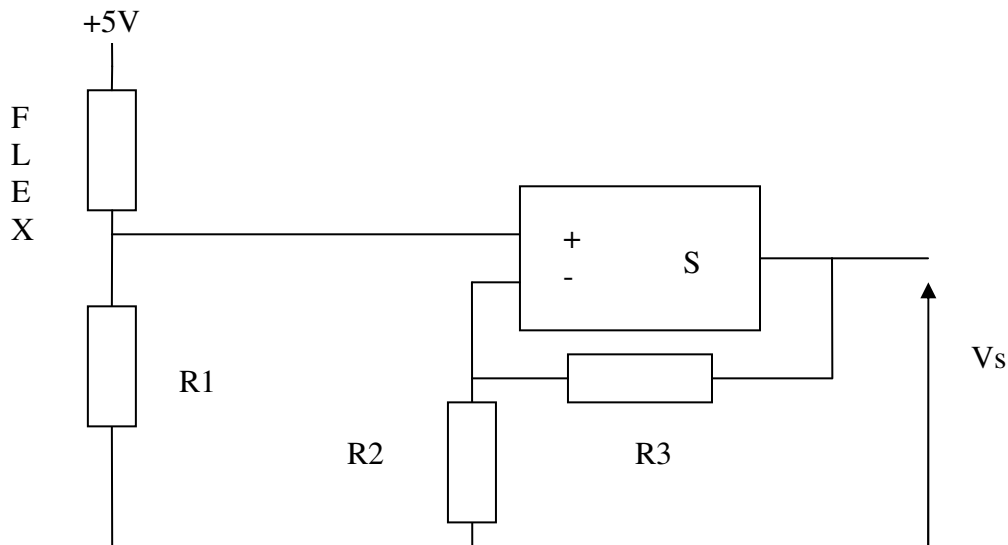
$$V_s = \frac{R_3 + R_2}{R_2} V_{pd}$$

Théoriquement, en prenant $R_3=10\text{ K}$ $R_2=1\text{ K}$, la tension varie entre $V_{50k}=9.1685\text{ V}$ et $V_{150k}=3.4375\text{ V}$

Expérimentalement, nous avons choisi d'autres résistances

Le bloc des deux premiers étages, représenté ci-dessous, donne donc comme sortie

$$V_s = \frac{R_3 + R_2}{R_2} * \frac{R_1}{R_1 + [50k - 150k]} V_{dd}$$



Nos contraintes étaient d'avoir une tension d'entrée comprise entre 1 et 5 V pour que l'amplificateur (TL074CN) fonctionne convenablement et renvoie des valeurs cohérentes, la tension de sortie idéale serait de 0 à 5V.

C'est en essayant plusieurs valeurs que nous avons trouvé une configuration satisfaisante : $R_1=47k$ $R_2=1k$ $R_3=1k$ et comme sorties :

$$V_{pt50k} = 2.42\text{ V} \quad \text{soit une dynamique de } 1.22\text{V}$$

$$V_{pt150k} = 1.19\text{ V}$$

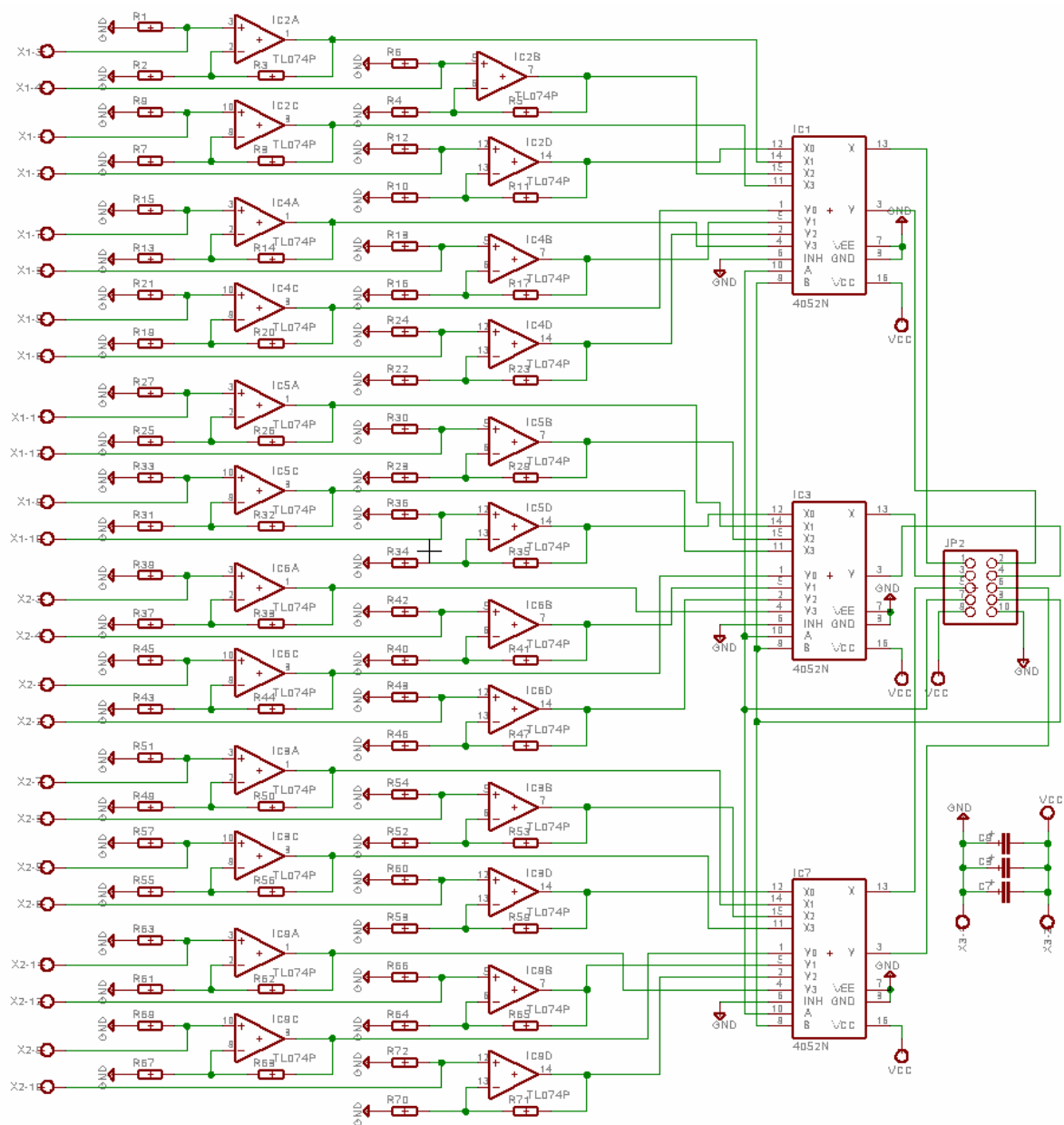
$$V_{s50k} = 4.84\text{ V} \quad \text{soit une dynamique de } 2.4\text{V}$$

$$V_{s150k} = 2.38\text{ V}$$

Notre gain en dynamique étant de 2 (2.4V), nous obtenons une résolution de 500 points pour un Flex soit pour un angle de 90° une précision de 0.18° . Ce résultat est suffisant, nous aurions pu être plus précis en élargissant la partie électronique notamment par un circuit offset pour décaler la sortie vers le 0V, ce qui, avec un autre amplificateur de coefficient 2 nous aurait donné une dynamique de 5V environ (0V 4,92V).

3/ Le multiplexeur permet de redistribuer les données au microcontrôleur, on sélectionne l'entrée analogique souhaitée par A0 ou A1.

Ci-dessous, le schéma global de la carte démultiplexage :



VI. Les DLL de communication

1. La connexion au PC

La carte de traitement du signal envoie les informations des différents canaux analogiques de la façon suivante, il faut savoir que chaque canal code la valeur analogique de son entrée sur 10 bits :

Le paquet fait 6*8bits de long soit 48bits

N°canal	0x2C	Valeur haute	Valeur basse	CRC	0x0
---------	------	--------------	--------------	-----	-----

Le débit de connexion est de 115200 bit/s

Le port série est configuré sans parité et avec un bit de stop.

Avec cette vitesse de communication on peut lire 2400 valeurs par seconde au maximum, si nous avons 24 capteurs cela nous fait 100 mesures par secondes. C'est suffisant pour un rafraîchissement correct de l'affichage.

La transmission par port série à une vitesse de 115200 bauds d'après la documentation provoque des erreurs de l'ordre de 3%. Ces erreurs se traduisent par l'affichage de valeurs aberrantes et par des mouvements erratiques de la main.

Il faut donc pouvoir détecter quand des erreurs ont eu lieu pour ne pas les prendre en compte. On a vu plus haut qu'on avait un échantillonnage de 100 valeurs environs, si on a 3% d'erreurs, nous avons en moyenne 97 valeurs correctes. Ceci pour dire que nous sommes pas obligés de corriger les erreurs car il nous reste suffisamment de valeurs pour avoir une bonne fluidité de mouvement.

Le CRC est assez simple mais efficace. Nous n'avons pas fait la théorie sur ce codage pour voir combien d'erreurs échappent à la détection, mais en pratique cela donne un bon rendement, il n'y a plus de mouvement erratique de la main.

$$(N^{\circ}\text{canal} + \text{Valeur haute} + \text{Valeur Basse}) \% 255 = \text{CRC}$$

Voici les différentes versions que nous avons réalisé et leurs caractéristiques :

Version 1

14815 bds 9 Hz 32 voies taille 1397 octets

38452 bds 24 Hz 32 voies taille 1397 octets

Version 1.1

44400 bds 27 Hz 32 voies taille 580 octets

46000 bds 35 Hz 24 voies taille 581 octets

Version 1.2

74000 bds 64 Hz 24 voies taille 356 octets

2. DLL de communication

Pour lire correctement les informations provenant du gant, il faut d'abord ouvrir un port de communication (ex : COM1), puis lire à une certaine fréquence les données provenant du port série.

Un canal est signalé en erreur quand sa valeur est nulle.

La DLL contient une variable qui est un tableau d'entiers positifs contenant toutes les valeurs des entrées analogiques de la carte, une fonction d'ouverture de port série et une fonction de lecture.

Cette fonction met dans le tableau AN toutes les valeurs des entrées analogiques :

.OpenDGComm(Nom du port)

Ex :

```
// Ouvre le port série
HANDLE cf
cf = OpenDGComm("COM1");

if (cf)
{ // Démarre le temporisateur d'horloge actualisation 50 fois par seconde
  SetTimer(ID_CLOCK_TIMER, 20, NULL);
}
```

.ReadDGComm(N° Handle)

Ex:

```
// Utilisation du timer
void CCommDlg::OnTimer(UINT nIDEvent)
{
  // Lit le port
  ReadDGComm(cf);
}
```

On utilise par exemple la commande AN_ext[10] pour récupérer la valeur du canal 10.

Version 1.0

Première version permet de communiquer avec la carte

Version 1.1

Ajout d'un CRC pour détecter les valeurs erronées

Version 1.2

Augmentation de la vitesse de transmission en utilisant un double buffer

3. DLL de rendu graphique

Cette DLL permet la commande du squelette par le clavier ou par le gant

La fonction **SetAxe()** permet de fixer les articulations

Déclaration : Void SetAxe(float* AN)

L'affectation des cases du tableau en fonction des axes est le suivant

Doigt	N°
Auriculaire	0,1,2
Annulaire	3,4,5
Majeur	6,7,8
Index	9,10,11
Pouce	12,13

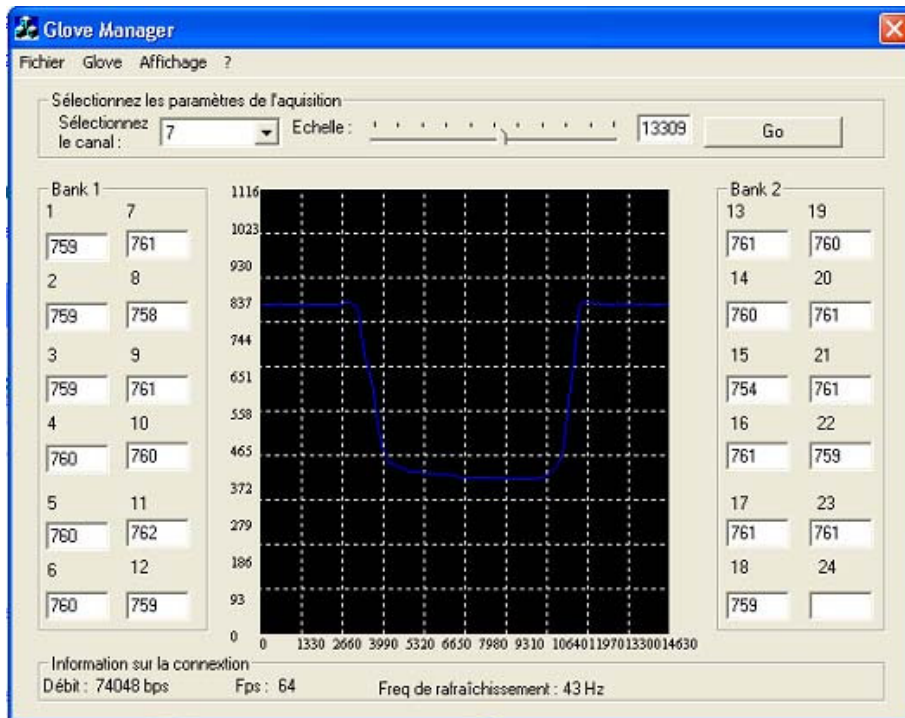
Pivot Auriculaire-Annulaire 14

La fonction **Hand()** permet d'afficher la main

Déclaration : Void Hand()

VII. Etude du FlexSensor

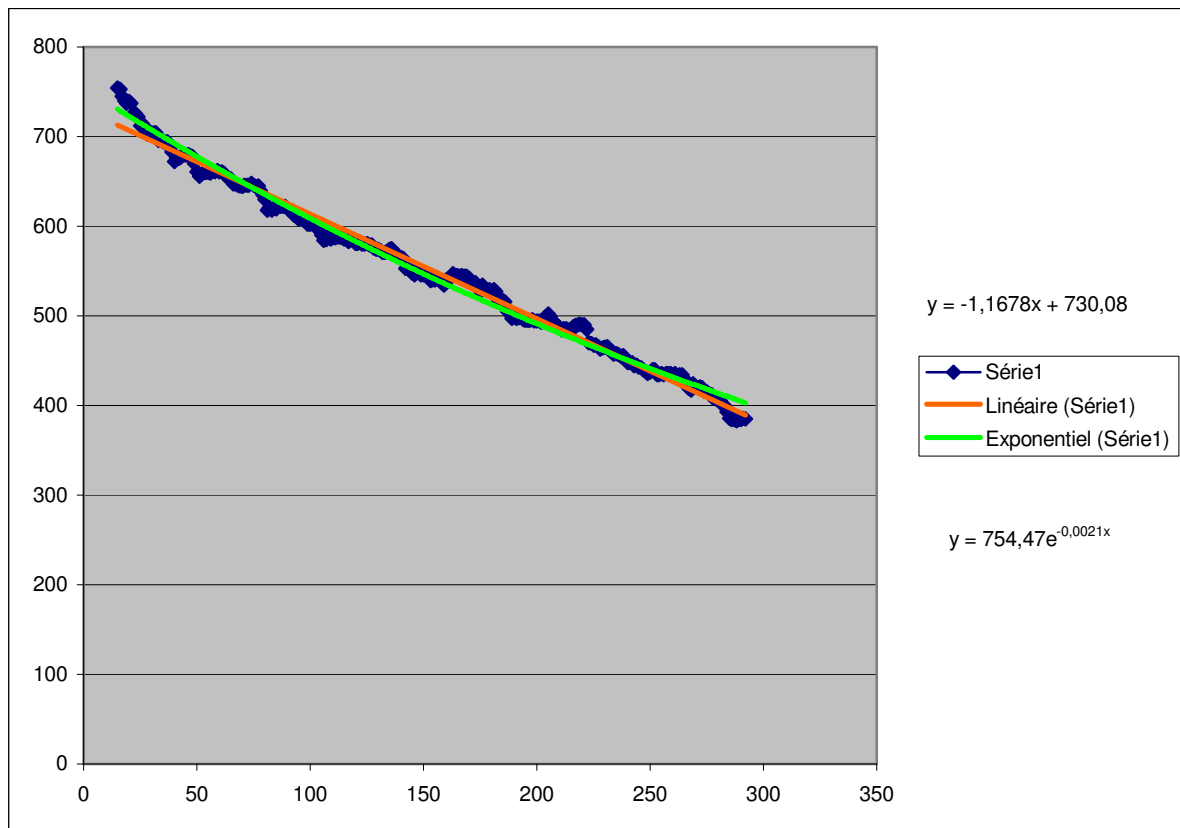
Pour l'étude des FlexSensors et du calibrage a été conçu un logiciel spécialisé dont voici l'interface graphique :



Ce logiciel permet l'exportation vers Excel d'échantillons capturés. Il permettra également d'effectuer le calibrage du gant en différentes étapes, il permettra également un affichage cohérent du squelette.

Ce que vous voyez en bas sont les statistiques de la ligne de réception et également du nombre d'échantillons qu'a pu traiter le programme par seconde. On a un débit de 74 kbds, le programme effectue la routine de calcul 64 fois par seconde et on a un taux de rafraîchissement des valeurs analogiques de 43.

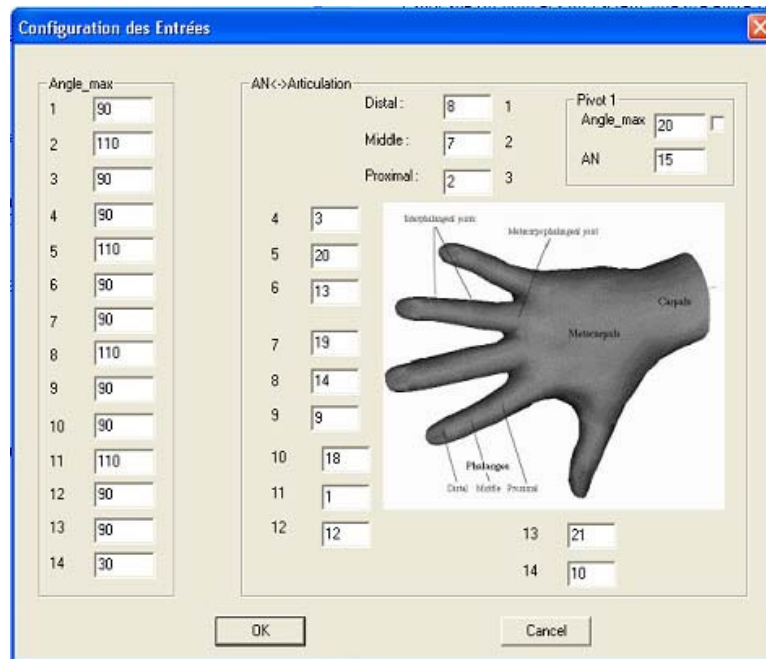
Pour pouvoir caractériser le Flex, nous avons utilisé Excel pour comparer son comportement avec des équations simples. Voici ci-dessous un graphique obtenu sous Excel à partir des échantillons recueillis par Glove Manager.



Nous avons comparé une régression linéaire avec une régression exponentielle et avons remarqué que la première permet une meilleure approximation de la courbe de réponse du Flex. Pour notre calibrage, nous utiliserons un facteur linéaire entre la valeur lue et l'angle réel.

VIII. Le programme Glove Manager

Voici le programme permettant l'utilisation du gant, son étalonnage,...
Après avoir choisi le port de communication auquel est reliée la carte, nous obtenons cette interface :



Vous pouvez personnaliser dans un premier temps les angles maximums de vos articulations. Pour calibrer le gant, il suffit de bouger les articulations initialement tendue de la main jusqu'à leur maximum.

Il est possible de régler l'échelle des temps et d'amplitude lors de l'acquisition des canaux mais également d'enregistrer et de charger les configurations personnelles des différents utilisateurs (enregistrement au format CSV possible).

Annexe : Programme du PIC

```
// Programme permettant la conversion des entrées analogique
// du 16F877 et de transmettre les valeurs par le port série
//-----
```

```
// Version 1.2
```

```
#include <pic.h>
```

```
// Declaration des Variables
```

```
//-----
```

```
char buf[8];
```

```
unsigned char
```

```
    udata, // Caractère reçu de l'USART
```

```
    mux    // bank A/D utilisé
```

```
;
```

```
unsigned int AD;
```

```
unsigned long ADTemp; //Stoche la valeur d'une voie
```

```
// Prototypes
```

```
//-----
```

```
void putsUSART(const char *data);
```

```
void Setup(void);
```

```
void temp(unsigned int delay);
```

```
void adc_read(unsigned char channel);
```

```
// Fonctions
```

```
//-----
```

```
//-----
```

```
// Setup() initialise les variables du programme
```

```
//-----
```

```
void Setup(void)
```

```
{
```

```
    udata = RCREG;
```

```
    udata = RCREG;
```

```
    RCREG = 0;
```

```
    udata = 0;
```

```
    SPBRG = 10; // 115200 baud @ 20MHz
```

```
    TXSTA = 0x24; // setup USART transmit
```

```
    RCSTA = 0x90; // setup USART receive
```

```
    TRISA = 0xFF; // PORT A en entrée
```

```

    TRISE = 0x07; // PORT E en entrée
    TRISB = 0x00; // PORT B en sortie;
    ADCON1 = 0x80; // Setup A/D converter
    ADCON0 = 0x81; // Configure la fréquence à Fosc/32 et active les entrées
    analogiques
  }

//-----
// putsUSART()
//
// Envoie une chaîne de caractère de la mémoire jusqu'à l'interface série
//-----
void putsUSART(const char *data)
{
    do
    {
        while(!(TXSTA & 0x02));
        TXREG = *data;
    } while( *data++ );
}

//-----
// Main()
//-----
main() {
    Setup();

    while(1) {
        char channel; // Channel numérisé
        int CRC; // Valeur du CRC

        CLRWDT();

        // Multiplexage des entrées analogiques
        for(mux=0;mux<4;mux++) {
            PORTB=mux;
            for(channel=2;channel<8;channel++) {

                adc_read(channel); // Lit un canal
                AD= (ADRESH*256 + ADRESL); // Met dans ADTemp les 10
                premiers bit résultant de la conversion

                temp(10); // Tempo de stabilisation du port série

                // Calcul du CRC
                CRC=(((mux*8)+channel+1)+AD)%255;

                // envoie sur le port série
                buf[0]=(mux*8)+channel+1;
                buf[1]=0x2C;
            }
        }
    }
}

```

```
        buf[2]=AD/256;
        buf[3]=AD;
        buf[4]=CRC;
        buf[5]=0;
        putsUSART(buf); // Envoie la chaine sur USART
    }
} //end while(1)
}

//-----
// Temp(int delay);
// Temporisation
//-----

void temp(unsigned int delay) {
    unsigned int j;
    for(j=0;j<delay;j++);
}

//-----
// adc_read(unsigned char channel)
// Fonction qui permet la lecture des canaux analogiques
//-----
void adc_read(unsigned char channel)
{
    ADCON0 = (channel << 3) + 0x81;    // enable ADC, Fosc/32.
    temp(15); // temps d'attente nécessaire pour l'initialisation de la conversion
    ADGO = 1;
    while(ADGO)
        continue;    // Attend qu'une conversion soit terminé
}
}
```