Géométrie pour la 3D

Pipeline OpenGL et Glut

On cherche à créer et animer un modèle humanoïde du type de la figure ci-dessous.

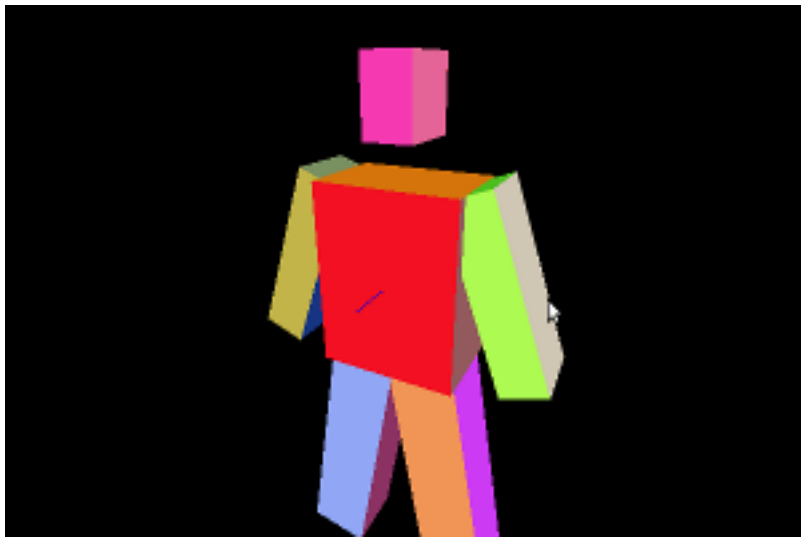

Figure 1. Modèle humanoïde à construire et à animer.

Pour ce faire, nous allons utiliser la librairie *OpenGL* pour la modélisation et la librairie *glut* pour la gestion des événements. Pour cela vous pourrez utiliser l'archive TP2.tgz dont vous trouverez le lien sur la page Moodle de ce cours. Il s'agit d'un squelette de programme que vous pourrez compléter pour parvenir au résultat recherché. Tous ces programmes pourront être compilés par la commande make.

1. Événements graphiques

Pour commencer nous allons nous familiariser avec la gestion des événements et la librairie *glut*. Pour cela on pourra partir du fichier Question1.c. Ce programme ne dessine rien. Ne soyez pas surpris par l'écran noir qu'il produit. Par contre, il est capable de réagir à des événements graphiques : clic de souris, frappe de touche au clavier etc.

- A présent, ce programme s'achève lorsque vous frappez la touche Escape. Modifiez-le pour qu'il s'achève lorsqu'on frappe la touche 'q'.
- Modifiez-le de nouveau pour qu'à chaque clic de souris, un message « bonjour » s'affiche sur le terminal.

2. Dessin élémentaire 3D

Téléchargez le fichier Question2.c. Celui-ci est presque identique à Question1.c à la différence près qu'il contient de nombreuses fonctions de dessin parmi lesquelles :

```
void chooseColor(double r, double g, double b) ;
void drawLine(Vector p1, Vector p2)
```

La première fonction change la couleur courante et la deuxième fonction trace un segment de droite de la couleur courante entre les points p1 et p2. Grâce à ces deux fonctions, que vous appellerez dans la fonction display(), dessiner un repère au centre de l'espace. Plus précisément, vous dessinerez :

- un segment de droite rouge reliant le point (0,0,0) et le point (1,0,0) ;
- un segment de droite vert reliant le point (0,0,0) et le point (0,1,0) ;
- un segment de droite bleu reliant le point (0,0,0) et le point (0,0,1).

3. Dessiner un cube

Le programme Question2.c contient aussi les fonctions

```
void drawQuad(Vector p1, Vector p2, Vector p3, Vector p4);
void drawCube();
```

La première dessine un quadrilatère de la couleur courante (représenté par ses arêtes ou par ses faces opaques) reliant les points p1, p2, p3 et p4. La seconde fonction est supposée dessiner un cube opaque de côtés de longueur 1. Mais c'est à vous d'écrire cette fonction grâce à la première fonction. Visualiser votre cube en appelant la fonction `drawCube()` dans la fonction `display()`.

4. Les matrices en OpenGL

A chaque instant pendant l'exécution de votre programme, le pipeline OpenGL peut être caractérisé par :

- une matrice de modélisation et de point de vue (GL_MODELVIEW) ;
- une matrice de projection (GL_PROJECTION) ;
- une matrice de couleurs (GL_COLOR) ;
- une matrice de textures (GL_TEXTURE).

Ici nous nous intéresserons à la matrice GL_MODELVIEW. Cette matrice peut :

- n'avoir aucun effet (lorsqu'il s'agit de la matrice identité)
- appliquer aux objets dessinés une translation, une rotation ou une homothétie selon la matrice ;
- appliquer aux objets une séquence de transformations (lorsque la matrice est le produit des matrices de ces transformations).

Par exemple voici la suite d'instructions suivante :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1,0,0);
glScalef(3,1,1);
drawCube();
```

- La première instruction spécifie que toutes les opérations matricielles qui suivent opéreront sur la matrice GL_MODELVIEW et non sur les autres matrices d'OpenGL.
- La seconde instruction initialise cette matrice et en fait la matrice identité. Si on dessine le cube tout de suite après cette instruction, aucune transformation ne lui serait appliquée.
- La troisième instruction glTranslatef(1,0,0) multiplie la matrice GL_MODELVIEW par une matrice de translation de vecteur (1,0,0) ;
- La quatrième instruction glScalef(3,1,1); multiplie la matrice GL_MODELVIEW obtenue précédemment par une matrice d'homothétie. En l'occurrence il s'agit d'une homothétie de rapport 3 suivant l'axe X.

Incluez ces lignes dans votre programme et observez-en les effets.

5. L'ordre d'application des matrices

- Remplacer glScalef(3,1,1) par glScalef(coef,1,1) où coef sera une variable globale. Modifiez votre programme pour qu'en frappant la touche 'a', la variable coef soit augmentée de 1 et qu'en frappant la touche 'A', elle soit diminuée de 1.
- En utilisant les touches 'a' et 'A' observez les transformations et déterminer le centre de l'homothétie. Même chose en changeant l'ordre des instructions glTranslatef et glScalef : où est le centre de l'homothétie ?
- Remplacer glScalef(coef,1,1,1) par glRotate(theta,0,1,0) où theta sera une variable globale. Modifiez votre programme pour qu'en frappant les touches 'z' et 'Z' on puisse augmenter et diminuer theta de 10 degrés.

- En utilisant les touches 'z' et 'Z' déterminer le centre de la rotation. Que se passe-t-il lorsqu'on change l'ordre d'application des deux instructions (glTranslatef et glRotatef) ? Comment expliquez-vous ces différences de comportement ?

Appliquer une homothétie de rapport 5 en x à un cube de façon à en faire une poutre. Comment peut-on faire tourner cette poutre autour de son extrémité gauche ?

6. Empilement de matrices et parenté

Soit la suite d'instructions suivante :

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(theta1,0,1,0);
drawCube();
glRotatef(theta2,1,0,0);
glTranslatef(0,2,0);
drawCube();
glRotatef(theta3,0,0,1);
glTranslatef(0,0,2);
glScalef(2,1,1);
drawCube();
```

Faire de façon à pouvoir contrôler la valeur de theta1, theta2 et de theta3 par des touches du clavier. Expliquer pourquoi lorsqu'on modifie theta1, tous les objets bougent alors que lorsqu'on modifie theta3 seul le dernier cube bouge.

L'application successive de matrice produit un système de *parenté*. Les objets dessinés en premiers sont parents des objets suivants car toute transformation appliquée aux parents s'appliquent aussi à tous les descendants. Cette parenté est intéressante dans certains cas mais pas dans tous. En particulier dans le cas du modèle humanoïde, si on dessine d'abord le torse, puis le bras gauche, toute transformation appliquée au torse se répercute au bras gauche, ce qui ne manque pas de sens. Par contre si on continue par le bras droit, alors toute transformation appliquée au bras gauche s'applique au bras droit ce qui, pour le coup, est absolument absurde.

Pour pouvoir manipuler de façon indépendante le bras droit et le bras gauche, l'idéal serait qu'après avoir appliqué la transformation au bras gauche, on puisse défaire cette transformation avant de passer à la suite. Cela reviendrait à inverser une matrice, ce qui n'est pas une opération très rapide. Une alternative consiste à mémoriser l'état de la matrice avant la transformation appliquée au bras gauche. Autrement dit, on peut faire des piles de matrices. Cela revient à faire les opérations suivantes :

- Appliquer transformations torse
- Dessiner le torse
- Empiler Matrice (Mémoriser la matrice juste après le dessin du torse)
- Appliquer transformations bras gauche
- Dessiner le bras gauche
- Dépiler Matrice (Retrouver la matrice juste après le dessin du torse)
- Appliquer transformations bras droit
- Dessiner le bras droit
- Dépiler Matrice
- etc.

L'empilement et le dépilement de matrices s'effectue grâce aux fonctions

- `glPushMatrix() ;`
- `glPopMatrix();`

7. Character setup

Grâce aux fonctions ci-dessus, construire le modèle humanoïde représenté sur la première figure de ce document. Dessiner, de plus, un grand carré représentant le sol.

Définir deux variables theta et phi qui permettront à l'utilisateur de tourner autour du modèle ou, ce qui revient au même, de tourner le modèle et le sol. Contrôler ces deux variables par des touches du clavier.

Le torse devra pouvoir monter et descendre (translation le long de l'axe y). Les membres (bras et jambes) devront pouvoir tourner autour de leur extrémité suivant deux axes. Contrôler ces rotations par des variables qui seront, elles-mêmes contrôlées par les touches du clavier.

8. Animation

Lorsqu'il n'y a plus aucun événement graphique à traiter, glut permet de lancer une fonction. C'est la fonction donnée en paramètre à la fonction glutIdleFunc() ; En l'occurrence, il s'agit de la fonction idle(). Faire évoluer les variables définies ci-dessus dans la fonction idle de façon à animer le modèle humanoïde.