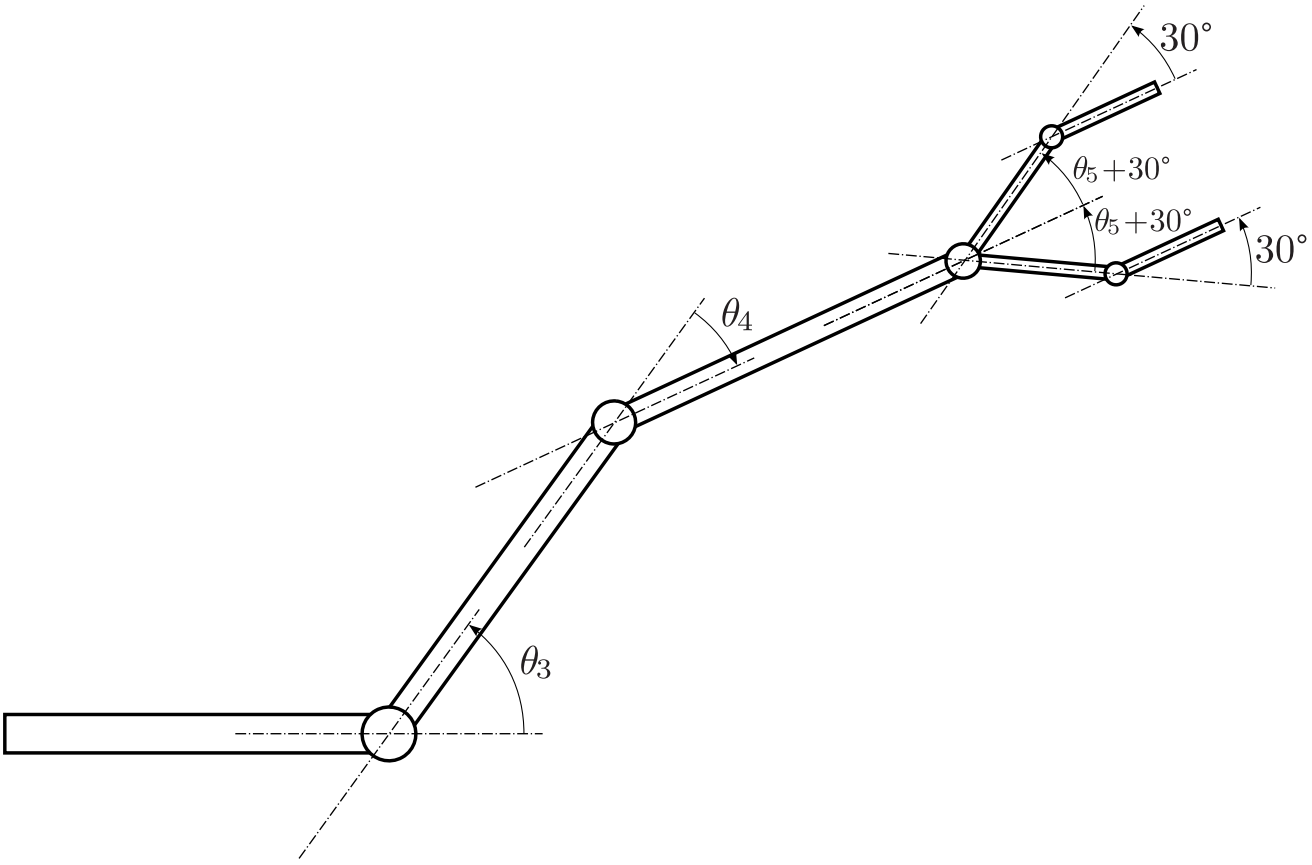
# [Visu-M2IM] TP 5 : Matrice de modélisation en OpenGL

Vincent Feuvrier (vincent.feuvrier@normalesup.org)

On se propose d'implémenter le dessin d'un bras articulé en 3D. Pour cela on va utiliser la matrice de modélisation d'OpenGL ainsi que certaines primitives graphiques fournies par GLUT (cube et sphère).




*Le bras articulé, composé de 7 pavés et de 5 sphères, vu perpendiculairement à l'axe $(Oz)$. Le premier tronçon est parallèle à l'axe $(Ox)$ et le centre de la première sphère est situé au point de coordonnées $(-3, 0, 0)$.*

Les rectangles du dessin sont des pavés en 3D de dimensions (de la gauche vers la droite) : $4\times0.6\times0.6$, $4\times0.5\times0.5$, $4\times0.4\times0.4$, $1.5\times0.2\times0.2$ et $1\times0.2\times0.2$.

Les cercles sont des sphères de rayons (de la gauche vers la droite) : 0.5, 0.4, 0.3 et 0.2.

Les angles de la figure sont indiqués sur le dessin et dépendent de la variable globale `Data`.

On définira une variable globale `Data` en début de programme qui contiendra 5 angles de rotation, initialement définis à zéro :

```
struct {
  int Theta1, Theta2, Theta3, Theta4, Theta5;
} Data={
  0,0,0,0,0
};
```

Ces 5 angles seront incrémentés/décrémentés respectivement par les 5 couples de touches du clavier `'0'/'1'`, `'2'/'3'`, `'4'/'5'`, `'6'/'7'` et `'8'/'9'`. La scène sera représentée par une caméra perspective d'angle vertical 70°, d'angle horizontal proportionnel au rapport des dimensions de la fenêtre, placée en $(0, 0, 10)$ et dirigée vers l'origine avec une pyramide de vision tronquée entre $z = 1$ et $z = 20$.

Pour cela on définira trois callbacks GLUT :
– pour le redimensionnement de la fenêtre

```
void reshapeFunc(int width,int height){
  glViewport(0,0,width,height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(70,(float) width/height,1,20);
}
```

– pour l'affichage :

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0,0,10,0,0,0,0,1,0);

  /* ... */

  glFlush();
  glutSwapBuffers();
```

La fonction `displayFunc` devra en outre appliquer à toute la scène une rotation d'angle $\theta_2$ perpendiculairement à l'axe $(Ox)$ suivie d'une rotation d'angle $\theta_1$ perpendiculairement à l'axe $(Oy)$ (attention, cela signifie qu'il faut faire les appels à `glrotatef` dans l'ordre inverse : d'abord avec $\theta_1$ et ensuite avec $\theta_2$).
– pour l'appui sur une touche du clavier :

```
void keyboardFunc(unsigned char key,int x,int y){
  switch (key){
    case '0':
      Data.Theta1++;
      break;
    case '1':
      Data.Theta1--;
      break;

    /* ... */

    case 'Q':
    case 'q':
      exit(0);
  }
  glutPostRedisplay();
}
```

Pour tracer les objets de la figure on utilisera :
– pour les pavés la fonction

```
void glutSolidCube(GLdouble size);
```

qui affiche un cube centré à l'origine dont les arêtes mesurent `size` ;
– pour les sphères la fonction

```
void glutSolidSphere(GLdouble radius,GLint slices,GLint stacks);
```

qui affiche une sphère centrée à l'origine de rayon `radius`. On prendra par exemple `slices` = `stacks` = 10 ;
– pour les translations la fonction

```
void glTranslatef(GLfloat x,GLfloat y,GLfloat z);
```

qui multiplie la matrice courante (à droite) par celle de la translation de vecteur (`x`, `y`, `z`) :

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– pour les dilatations la fonction

```
void glScalef(GLfloat x,GLfloat y,GLfloat z);
```

qui multiplie la matrice courante (à droite) par celle de la dilatation de facteurs `x`, `y` et `z` selon les trois axes $(Ox)$, $(Oy)$ et $(Oz)$ :

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– pour les rotations la fonction

```
void glRotatef(GLfloat angle,GLfloat x,GLfloat y,GLfloat z);
```

qui multiplie la matrice courante (à droite) par celle de la rotation d'angle `angle` perpendiculairement au vecteur (`x`, `y`, `z`) ;

$$\begin{bmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ zx(1-c)-ys & zy(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

où $c = \cos(angle)$, $s = \sin(angle)$ et $x, y, z$ sont les coordonnées du vecteur fourni après normalisation.

– pour sauvegarder la matrice courante dans la pile de matrices :

```
void glPushMatrix(void);
```

– pour revenir à la matrice courante précédemment sauvegardée dans la pile :

```
void glPopMatrix(void);
```

On pourra définir le programme principal ainsi, pour créer une fenêtre et gérer facilement l'éclairement de la scène avec une lampe :

```
int main(int argc,char *argv[]){
  int a=800,b=600,x,y;
  glutInit(&argc,argv);
  x=glutGet(GLUT_SCREEN_WIDTH);
  y=glutGet(GLUT_SCREEN_HEIGHT);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowPosition((x-a)/2,(y-b)/2);
  glutInitWindowSize(a,b);
  glutCreateWindow("Robot arm");
  glutReshapeFunc(reshapeFunc);
  glutDisplayFunc(displayFunc);
  glutKeyboardFunc(keyboardFunc);
  glEnable(GL_LIGHTING);
  glEnable(GL_COLOR_MATERIAL);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
}
```