IV Construction d'une scène 3D

Il s'agit de définir ici les différents objets qui composent une scène, et de les positionner dans l'espace à l'aide de transformations géométriques 3D.

Sous GLUT, la scène est construite dans la fonction déclarée par `glutDisplayFunc` avant de lancer la boucle d'événement (`glutMainLoop`). On rappelle que cette fonction d'affichage est déclenchée par le système :
- soit parce que la fenêtre a été modifiée,
- soit parce que le programmeur l'a demandé par l'intermédiaire de la fonction `glutPostRedisplay()`

OpenGL appliquera automatiquement la matrice de projection aux objets que l'on a construit pour obtenir une image 2D qui sera affichée dans la fenêtre.

1. Transformations géométriques de bases

La construction d'un objet et son positionnement dans la scène vont se faire à partir de trois transformations géométriques de bases :
- la translation,
- la rotation (en degré) autour d'un axe porté par un vecteur,
- l'homothétie suivant les trois axes X, Y et Z.

On rappel que ces transformations sont représentées par des matrices de dimension 4 dans l'espace projectif. Appliquer une de ces transformations consiste à opérer sa matrice sur les coordonnées des différents sommets de l'objet considéré.

2. Enchaînement des transformations

OpenGL dispose d'une matrice de transformation courante pour la modélisation, matrice qui est rendue active par la fonction :

```
glMatrixMode(GL_MODELVIEW)
```

Cette **matrice de modélisation** est appliquée automatiquement à tous les objets qui vont être tracés.

Le positionnement d'un objet dans une scène est décomposé comme une succession de transformations de base qu'OpenGL traduit par un produit matriciel (cf les *espaces projectifs*). La matrice de transformation courante est construite :
- à partir de la matrice identité,
- et par produits successifs avec des matrices d'opérations de base.

La matrice de modélisation est initialisée avec l'identité en appelant la fonction :

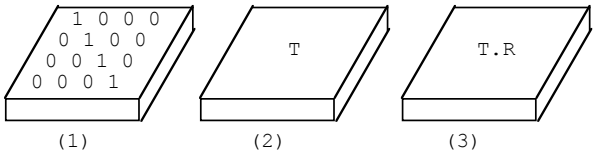
```
glLoadIdentity()
```

Il est possible de gérer soit même les opérations de base mais OpenGL propose des fonctions simples où le produit matriciel avec la matrice de transformation courante est implicite :

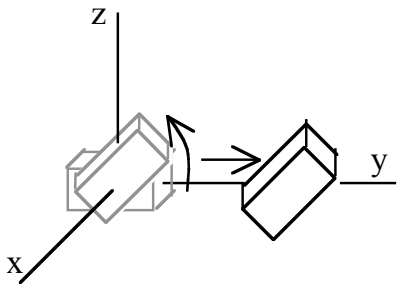
```
glTranslatef(GLfloat x, GLfloat y, GLfloat z) ;
glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z) ;
glScalef(GLfloat x, GLfloat y, GLfloat z) ;
```

Les appels successifs de ces fonctions composent donc une seule transformation.
Remarque : il faut lire les opérations effectuées sur l'objet dans l'ordre inverse de leur apparition dans le code.

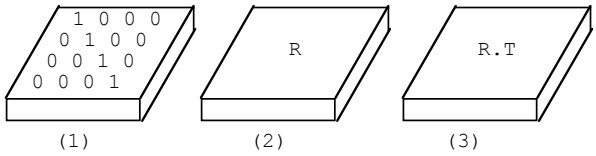
```
/* matrice de transformation A */
glLoadIdentity() ;      /* 1 */
glTranslatef(0, 5, 0);  /* 2 */
glRotatef(45, 1, 0, 0); /* 3 */
/* objet qui subira la transformation*/
dessineBoite();
```



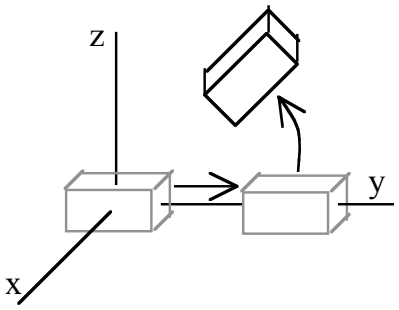
Evolution de la matrice de modélisation (A)




```
/* matrice de transformation B */
glLoadIdentity() ;      /* 1 */
glRotatef(45, 1, 0, 0); /* 2 */
glTranslatef(0, 5, 0);  /* 3 */
/* objet qui subira la transformation*/
dessineBoite();
```



Evolution de la matrice de modélisation (B)



3. Gestion des transformations

OpenGL dispose en fait d'une **pile de matrices de modélisation** qui va faciliter la description hiérarchique d'un objet complexe. La matrice courante (la seule active) est celle se trouvant au sommet de pile, mais les jeux d'empilage et dépilage permettront d'appliquer la même transformation à plusieurs assemblages ayant eux mêmes nécessité des transformations spécifiques pour leur construction.

`glLoadIdentity()` : mettre l'identité au sommet de pile
`glPushMatrix()` : empiler
`glPopMatrix()` : dépiler

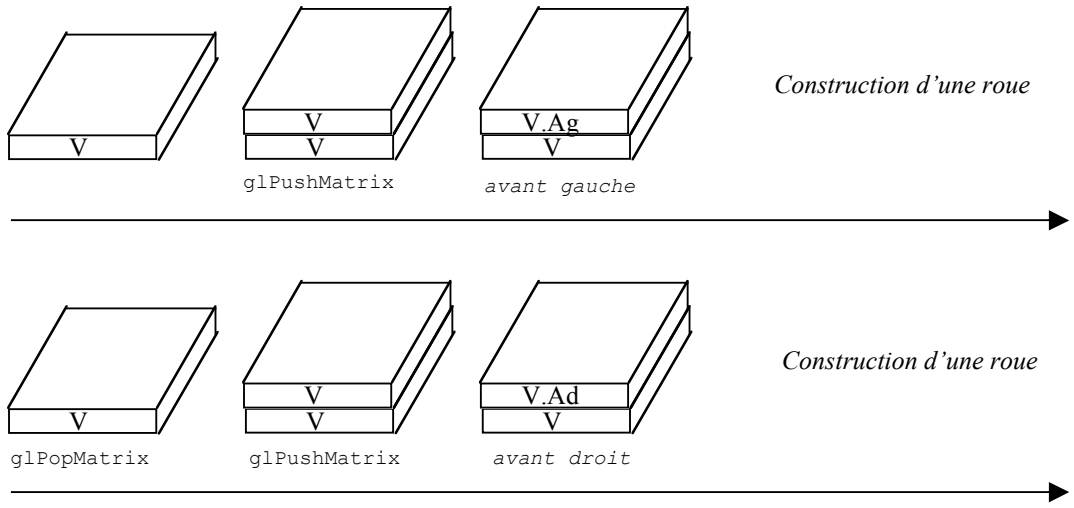
Si l'opération de dépilage ne présente pas de difficulté particulière (on retrouve la transformation précédente), l'opération d'empilage réclame quelques précisions : il s'agit en fait d'une duplication de la matrice se trouvant au sommet de pile. Toutes les opérations qui seront effectuées par la suite seront combinées à la transformation initiale (dupliquée dans le sommet de pile) de sorte que le sous-objet que l'on est en train de construire « hérite » de la transformation appliquée globalement à l'objet.
Bien sûr, il est toujours possible de faire suivre un `glPushMatrix()` par `glLoadIdentity()` pour oublier temporairement une matrice de modélisation.

Par exemple, pour dessiner une voiture, on définira ce qu'est la construction d'une roue dans le repère absolu, et on se positionnera successivement aux quatre coins de la voiture (repère voiture) avant d'appeler cette procédure. Si la voiture a été positionnée à un endroit spécifique de la scène, ses roues subiront aussi la transformation.

```
void dessineRoueEtBoulons()
{ int i ;
  dessineRoue() ;
  for (i=0; i<3; i++)
  { glPushMatrix() ;
      glRotatef(120*i, 0, 0, 1) ;
      glTranslatef(2, 0, 0) ;
      dessineBoulon() ;
    glPopMatrix() ;
  }
}

void dessineVoiture()
{ int i, posx[4]={20, 20, -20, -20}, posz[4]={8, -8, 8, -8};
  dessineCarrosserie() ;
  for (i=0; i<4; i++)
  { glPushMatrix() ;
      glTranslatef(posx[i], 5, posz[i]) ;
      dessineRoueEtBoulons() ;
    glPopMatrix() ;
  }
}
```

V : matrice positionnant la voiture dans la scène (repère « scène »)
Ag : matrice définissant l'avant gauche de la voiture dans le repère « voiture »
Ad : matrice définissant l'avant droit de la voiture dans le repère « voiture »



Evolution de la pile de modélisation dans la construction hiérarchique d'une voiture

Remarques :

- La pile est initialisée par OpenGL avec la matrice identité.
- Il est intéressant de conserver la matrice identité en bas de pile pour éviter de rappeler systématiquement `glLoadIdentity()` (=> une construction débute toujours par un *push*) .

4. Listes d'affichage

Il est possible de stocker une suite de routines OpenGL (à l'exception de quelques fonctions...) dans une liste qui pourra être réutilisée plusieurs fois. Il y a alors une précompilation des instructions GL et cette opération sera particulièrement rentable lorsqu'un objet « définitif » est dessiné plusieurs fois, soit parce qu'il constitue une primitive employée à plusieurs reprises (roue d'une voiture), soit parce qu'il se déplace dans la scène (animation).

- une liste est identifiée par un numéro (`GLint`) strictement positif,
- la création et la suppression des listes est gérée par OpenGL : **`glGenLists(`*`nbIndex`*`)`** attribue *`nbIndex`* numéros de listes consécutifs, le premier numéro est retourné par cette fonction (0 si échec d'allocation),
- **`glDeleteLists(`*`numListe, nbre`*`)`** restitue au système le *nbre* de listes indiqué à partir de *`numListe`*.
- **`glNewList(`*`numListe, mode`*`)`** et **`glEndList()`** permettent de créer une liste,
- **`glCallList(`*`numListe`*`)`** exécute une liste d'affichage,

Par exemple, pour dessiner un tricycle, on pourra stocker la construction d'une roue dans une liste et appeler 3 fois cette liste après avoir définie les spécificités de chaque roue (position, taille).

```
int listeRoue ;
...
listeRoue  = glGenLists(1);
...
glNewList(listeRoue, GL_COMPILE) ; // ou encore GL_COMPILE_AND_EXECUTE
  suite d'instructions pour dessiner une roue de tricycle
glEndList() ;


void dessinerTricycle()
{ ...
  transformations pour positionner la roue arrière gauche
  glCallList(listeRoue);
  transformations pour positionner la roue arrière droite
  glCallList(listeRoue);
  transformations pour positionner la roue avant
  glScalef(1.2, 1.2, 1.) ; // roue avant 20% plus grande mais même largeur
  glCallList(listeRoue);
  ...
}
```

Remarque : Une liste peut-être construite avant de déclencher la boucle d'événements dans le `main()` (`glutMainLoop()`). Il faut toutefois que le processus graphique soit déjà initialisé (i.e. après `glutInit()` et `glutInitDisplayMode()`) et que l'on choisisse l'option `GL_COMPILE` .