Synthèse d'Image - TD03

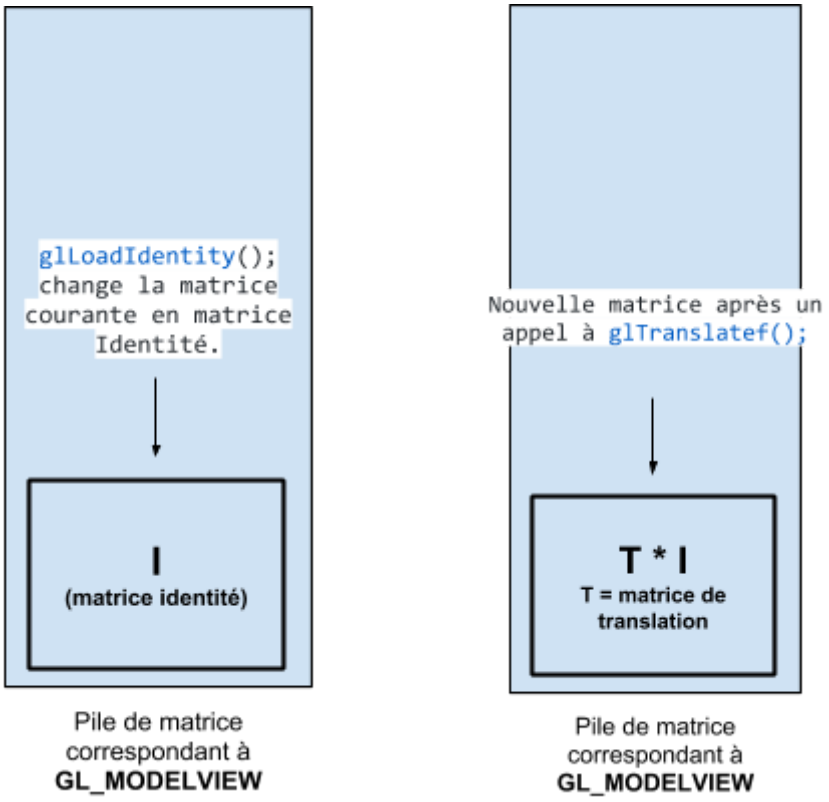
Piles de matrices et listes d'affichage

Lors de cette séance, nous aborderons les piles de matrices et les listes d'affichage OpenGL

glPushMatrix() et glPopMatrix()

Au TP précédent, vous avez appliqué des transformations sur vos objets via les fonctions `glTranslatef`, `glRotatef` et `glScalef`. Lorsque vous faites appel à l'une de ces fonctions, OpenGL va en réalité **modifier la matrice courante en la multipliant par une autre matrice qui représente la transformation voulue** (une matrice de translation, une matrice de rotation, une matrice de scale, etc.).

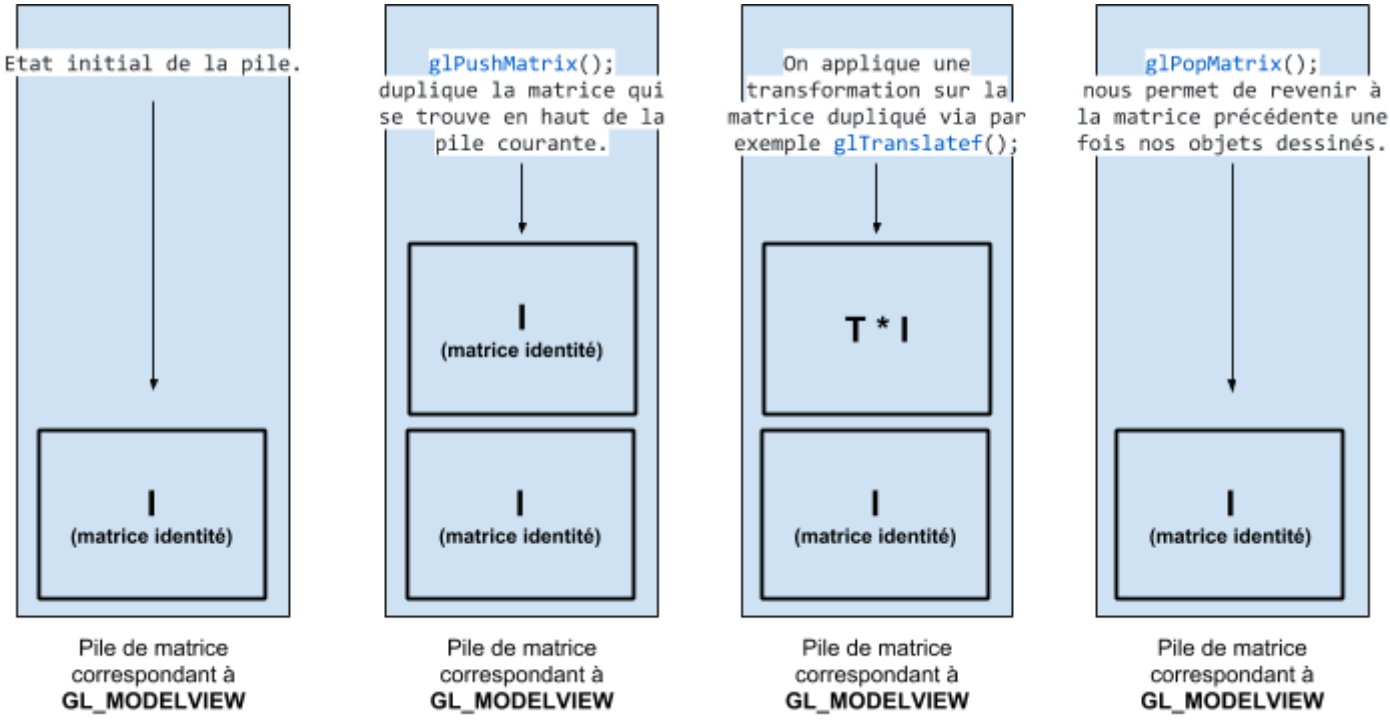
En vérité, OpenGL ne possède pas qu'une seule matrice, mais une **pile de matrice**. Et chaque appel à une fonction de transformation modifie la matrice qui se situe en haut de la pile courante :



Lorsque vous modifiez la matrice en haut de la pile de matrices courante, l'**ancienne matrice est perdue** (on pourrait la retrouver en appliquant la matrice de transformation inverse, mais cela serait trop contraignant). Ca devient problématique

dans le cas où vous voudriez appliquer une première transformation à plusieurs objets, puis une deuxième transformation différente pour chacun de ces objets (vous seriez obligés d'appliquer l'ensemble de la transformation à tous les objets).

Pour remédier à ça, OpenGL dispose de la fonction `glPushMatrix`, qui permet de **dupliquer la matrice se situant en haut de la pile courante**. Une fois cette matrice copiée, vous pouvez lui appliquer n'importe quelle transformation sans avoir peur de perdre la matrice originale. Une fois vos transformations effectuées (et vos objets dessinés), vous pouvez "dépiler" la pile via la fonction `glPopMatrix`.



Bien évidemment, vous pouvez appeler plusieurs fois glPushMatrix pour appliquer des transformations successives, par exemple :

```
glPushMatrix();
	glTranslatef(0.8, 0.0, 0.0);
	glPushMatrix();
		glRotatef(45, 0, 0, 1);
		drawSquare();
	glPopMatrix();
	glPushMatrix();
		glRotatef(-45, 0, 0, 1);
		drawSquare();
	glPopMatrix();
glPopMatrix();
```

NB : Vous pouvez clarifier votre code en mettant des indentations pour représenter l'empilement des matrices.

Prérequis

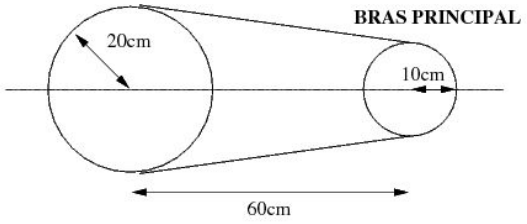
Pour ce TP, vous aurez besoin de réutiliser les fonctions canoniques de dessins du TP2 : `drawSquare()` et `drawCircle()`.

Dans ce TP, nous allons contruire un bras mécanisé de plusieurs pièces reliées entre elles.

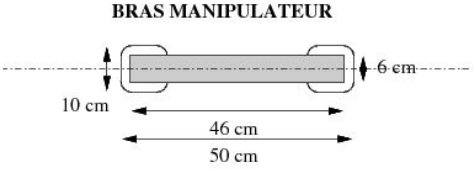
EXERCICE 01 : Construction des morceaux

Notre bras mécanique sera constitué de 3 parties :

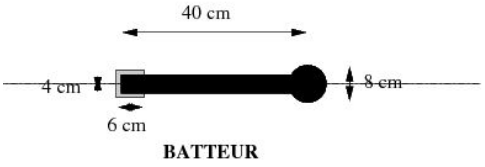
1. le bras principal, composé de 2 cercles et d'un parallélépipède reliant ces deux cercles.



2. le bras manipulateur, composé de 2 carrés aux bords arrondis et d'un rectangle reliant ces deux carrés.



3. le batteur, composé d'un carré aux bords arrondis, d'un rectangle et d'un cercle.



Les unités sont données à titre indicatif. Vous pouvez utiliser l'échelle 1cm = 1 unité OpenGL.

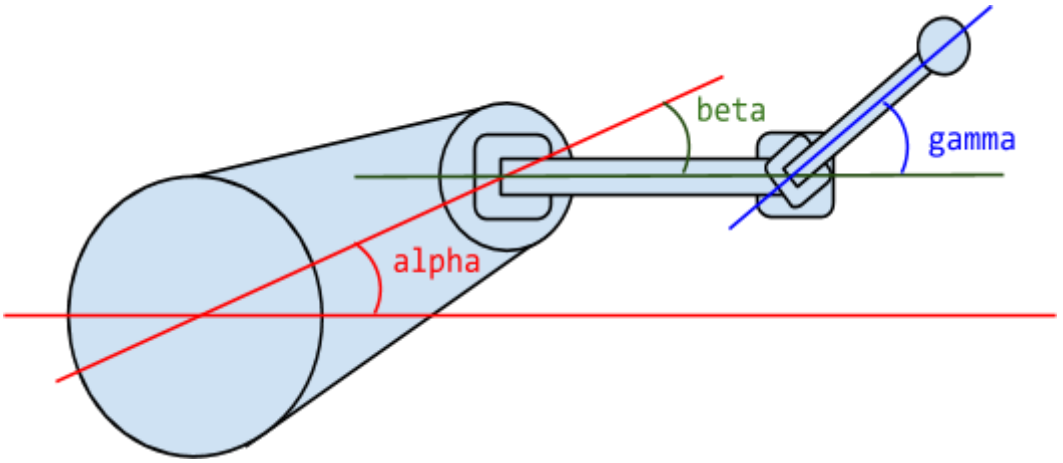
1. **Ecrire les fonctions suivantes** en utilisant uniquement les fonctions canoniques `drawSquare` et `drawCircle`, les fonctions `glPushMatrix` et `glPopMatrix`, ainsi que les fonctions de transformation `glTranslatef` et `glScalef`.

    a. **`drawRoundedSquare()` :** fonction qui dessine un carré à bords arrondis canonique (1 unité de côté).
    b. **`drawFirstArm()`** : fonction qui dessine le bras principal. Pour le dessin du carré central, vous pouvez exceptionnellement utiliser des instructions glBegin/glEnd.
    c. **`drawSecondArm()` :** fonction qui dessine le bras manipulateur (réutilisez `drawRoundedSquare` ici).
    d. **`drawThirdArm()` :** fonction qui dessine le batteur.

    Affichez un à un chacun des éléments pour vérifier votre travail.

EXERCICE 02 : Assemblage des morceaux

Nous allons maintenant assembler les morceaux entre eux selon le schéma suivant :



Dans votre boucle de rendu :

1. **Dessinez le bras mécanique complet** en utilisant vos fonctions de dessins de chacun des éléments, les fonction `glPushMatrix` et `glPopMatrix`, et les fonctions de transformation `glTranslatef` et `glRotatef`.
   Vous pouvez utiliser les valeurs d'angle suivantes :
    - **alpha** = 45°
    - **beta** = -10°
    - **gamma** = 35°

2. **Faites varier vos angles alpha, beta et gamma au cours du temps.**
3. **Dessinez plusieurs batteurs au bout du bras manipulateur, avec des valeurs d'angle différentes.**

EXERCICE 03 : Création de listes d'affichage

Nous allons maintenant optimiser notre rendu en créant des listes d'affichage OpenGL.

Lorsque vous utilisez des commandes comme `glVertex2f`, OpenGL envoie des informations de dessin à la carte graphique (comme les coordonées du point, sa couleur, etc.). Effectuer cet envoie à chaque tour de boucle de rendu n'est pas très optimal. Nous allons donc utiliser des listes d'affichage OpenGL.

Une liste d'affichage OpenGL (*display list*) est **un ensemble d'instructions de dessins** qu'OpenGL va stocker sur la carte graphique, permettant d'éviter de refaire appel aux mêmes fonctions de dessin à chaque tour de boucle. Pour créer une liste, il faut appeler les fonctions suivantes :

```
GLuint id = glGenLists(1); // 1 pour dire qu'on veut générer 1 liste

// GL_COMPILE permet d'envoyer la liste de commandes à la carte graphique sans
l'executer (il existe aussi GL_COMPILE_AND_EXECUTE permettant de compiler puis d'appeler
la liste).
glNewList(id, GL_COMPILE);

/* Code de dessin … */

glEndList();
```

Puis, pour dessiner la liste :

```
glCallList(id);
```

Entre les instructions `glNewList` et `glEndList`, vous pouvez appeler la plupart des fonctions openGL de dessin (tel que `glTranslatef`, `glRotatef`, …) que vous ne pouviez pas utiliser dans entre glBegin et glEnd. glCallList appellera votre code de dessin qui sera affecté par les transformations matricielles que vous aurez appliqué avant.

1. **Modifiez vos fonctions `drawFirstArm()`, `drawSecondArm()` et `drawThirdArm()` pour qu'elles renvoient un identifiant de liste au lieu de dessiner.** Vous pouvez les renommer respectivement `createFirstArmIDList()`, `createSecondArmIDList()` et `createThirdArmIDList()`.
2. **Dans votre boucle de rendu, appelez le dessin de ces listes en remplacement des appels aux fonctions de dessin.**
3. **Est-il intéressant de créer une liste d'affichage pour le bras mécanique en entier ? Pourquoi ?**
4. **Lister les avantages et les inconvénients d'utiliser les listes d'affichage.**