

Rapport de TP3 – Représentation visuelle d'objets. 1/2

I. Introduction

L'objectif de ce TP est de représenter dans l'espace une maison et pouvoir naviguer à l'intérieur de cet espace. Pour cela nous utiliserons le module **pygame** permettant un affichage graphique en 3D. Nous commencerons par créer un espace à 3 dimensions et ses fonctions de navigation (zoom, translation, rotation), puis nous créerons une section qui représentera le patron de notre maison, avant de terminer par la création de notre maison avec ses murs, ses fenêtres, ses portes etc.

II. Préparation à faire avant le TP

1. Question (1).

```
import pygame
pygame.init()
ecran = pygame.display.set_mode((300, 200))
pygame.quit()
```

La première ligne sert à importer le module **pygame**, la deuxième, à initialiser tous les modules **pygame** importés. La troisième ligne sert quant à elle à afficher une fenêtre de longueur 300 et de largeur 200. La dernière ligne sert à désinitialiser tous les modules **pygame** qui ont été précédemment initialisés. Lorsqu'on lance le programme la fenêtre s'affiche et disparaît immédiatement car nous avons pas mis de conditions permettant son affichage de manière plus longue.

2. Question (2).

```
import pygame

pygame.init()
ecran = pygame.display.set_mode((600, 200))

continuer = True
while continuer:
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            continuer = False
pygame.quit()
```

Ce morceau de programme permet l'affichage d'une fenêtre de longueur 600 et de largeur 200 jusqu'à tant que nous, utilisateur, la fermons en appuyant sur n'importe quelle touche de notre clavier. Pour cela nous commençons par écrire le même programme qu'à la question précédente, ensuite on initialise le booléen **continuer** en **True** avant d'effectuer une boucle **while**. Cette boucle permet à ce que la fenêtre qui se coupait immédiatement à la question précédente reste affichée jusqu'à la sortie de la boucle. Au sein de celle-ci on utilise **event** c'est à dire un événement. Ici l'événement est **pygame.KEYDOWN** signifiant l'appui sur une touche du clavier. Ainsi la fenêtre « est en attente », c'est à dire affichée, et si l'utilisateur appuie sur une touche (notre événement **pygame.KEYDOWN**), la boucle **if** est vérifiée et donc notre booléen **continuer** devient **false** provoquant la sortie de la boucle **while** et la fermeture de la fenêtre.

3. Question (3).

Cette question nous demande de réaliser 3 droites qui seront les composantes (x,y,z) de notre espace à 3 dimensions. Pour cela on utilise OpenGL. On crée chaque droite avec une couleur différente et une direction différente. Ces droites sont orthogonales. Ainsi, lorsque l'on crée notre espace le troisième vecteur apparaît pas à l'écran car il vient vers nous, il est donc nécessaire d'effectuer une rotation pour pouvoir voir ce dernier.

Avant le rotate.

```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu


if __name__ == '__main__':
    pygame.init()
    display = (600, 600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

    # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)

    # Placer ici l'utilisation de gluPerspective
    glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50)
    #gl.glTranslatef(0, 0, 2, -5)
    #gl.glRotate(-90, 1, 0, 0)

    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
    gl.glColor3fv([125, 0, 0]) # Indique la couleur ici rouge
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([1, 1, -2]) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([0, 125, 0]) # Indique la couleur ici vert
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([-1, 1, -2]) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([0, 0, 125]) # Indique la couleur ici bleu
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([0, 0, 1]) # Deuxième vertice : fin de la ligne
    gl.glEnd() # Fin du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique

    CONTINUER=True
    while CONTINUER:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN or event.type == pygame.QUIT:
                pygame.quit()
                CONTINUER=False
```



Après le rotate.

```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

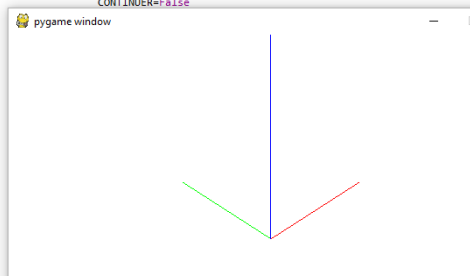
if __name__ == '__main__':
    pygame.init()
    display = (600, 600)
    pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)

    # Sets the screen color (white)
    gl.glClearColor(1, 1, 1, 1)
    # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
    gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
    gl.glEnable(gl.GL_DEPTH_TEST)

    # Placer ici l'utilisation de gluPerspective
    glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50)
    gl.glTranslatef(0, 0, 2, -5)
    gl.glRotate(-45, 1, 0, 0)

    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
    gl.glColor3fv([125, 0, 0]) # Indique la couleur ici rouge
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([1, 1, -2]) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([0, 125, 0]) # Indique la couleur ici vert
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([-1, 1, -2]) # Deuxième vertice : fin de la ligne
    gl.glColor3fv([0, 0, 125]) # Indique la couleur ici bleu
    gl.glVertex3fv([0, 0, -2]) # Premier vertice : départ de la ligne
    gl.glVertex3fv([0, 0, 1]) # Deuxième vertice : fin de la ligne
    gl.glEnd() # Fin du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique

    CONTINUER=True
    while CONTINUER:
        for event in pygame.event.get():
            if event.type == pygame.KEYDOWN or event.type == pygame.QUIT:
                pygame.quit()
                CONTINUER=False
```

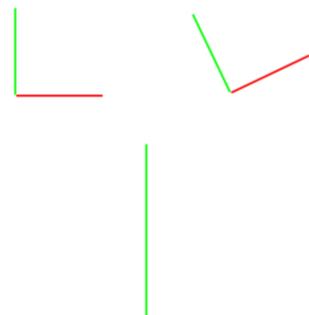


1. Découverte de l'environnement du travail du TP

(1. a) Cette question nous demande de tester deux fonctions différentes :

```
def Q1a():
    return Configuration()
```

```
def Q1a():
    return Configuration({'screenPosition': -5, 'axisColor': [1, 1, 0]}).display()
```



On observe que sur la deuxième exécution, la norme des vecteurs est plus grand, et que l'axe rouge est devenu jaune.

(1. b)



POLYTECH
ANNECY-CHAMBÉRY



UNIVERSITÉ
SAVOIE
MONT BLANC

Le chaînage de l'appel des méthodes **setParameter()** et **display()** est possible car nous travaillons uniquement avec des matrices de même dimension dont nous pouvons en faire des opérations.

Comme nous travaillons dans un espace à 3 dimensions **screenPosition** sert à déterminer la position (on initialise notre espace) de l'écran sous forme de matrice, d'où le fait d'utiliser la méthode **initializeTransformationMatrix()**.

(1. c) Pour cette question on souhaite avoir l'axe z représenté verticalement sur l'écran et que l'axe x soit représenté horizontalement.

```
def initializeTransformationMatrix(self):  
    gl.glMatrixMode(gl.GL_PROJECTION)  
    gl.glLoadIdentity()  
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)  
  
    gl.glMatrixMode(gl.GL_MODELVIEW)  
    gl.glLoadIdentity()  
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])  
    gl.glRotatef(-90,1,0,0)
```

On effectue donc une rotation autour de l'axe x de -90°.

III- Mise en place des interactions avec l'utilisateur avec Pygame

(1. e) Après avoir lu la documentation de Pygame on a trouvé que pour pouvoir utiliser l'appui de touche c'était la commande **pygame.K_(touche appuyée)**. Ici on veut que cela zoom de 1,1 dans toutes les directions vers l'avant, donc quand la touche **PAGEUP** est enfoncée, et dézoom de 1/1,1 dans toutes les directions quand on appui sur **PAGEDOWN**.

On effectue simplement une condition vérifiant cela.

```
def processMouseButtonDownEvent(self):  
    if self.event.key == pygame.K_PAGEUP:  
        gl.glScalef(1.1,1.1,1.1)  
    elif self.event.key == pygame.K_PAGEDOWN:  
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```

(1. f) L'issue de cette question est de nous permettre de faire une rotation dans notre espace et une translation pour naviguer dans l'espace une fois que nous aurons notre maison. Ainsi, on utilise l'appui du bouton gauche/droit de la souris (**pygame.mouse.get_pressed()[X]**). Si le bouton gauche est enfoncée on effectue deux rotations, selon x et selon z suivant les coordonnées du curseur de la souris sur l'écran (**self.event.rel[0]**, **self.event.rel[1]**). Si le bouton droit est enfoncé on effectue deux translations selon x et z suivant les coordonnées du curseur de la souris sur l'écran.

```
def processMouseMotionEvent(self):  
  
    if pygame.mouse.get_pressed()[0]==1:  
        gl.glRotate(self.event.rel[1],1,0,0)  
        gl.glRotate(self.event.rel[0],0,0,1)  
  
    elif pygame.mouse.get_pressed()[2]==1:  
        gl.glTranslatef(self.event.rel[0],0,0)  
        gl.glTranslatef(0,0,-self.event.rel[1])
```

Petit bilan de cette première partie de TP.

Cette première partie de TP nous a permis de poser les bases pour la suite. Nous avons ainsi notre espace à 3 dimensions et toutes ses fonctionnalités de navigation. Cette partie à été intéressante car elle nous a permis d'appréhender le module **pygame** et son affichage. La prochaine partie sera la création d'une section dans l'espace et la création de la maison.