NICOLAS Thomas
GUERIOT Benjamin

Date du TP (24/11/2021)

Rapport de TP3

Représentation visuelle d'objets

NICOLAS Thomas

&

GUERIOT Benjamin

Sommaire

NICOLAS Thomas
GUERIOT Benjamin

Introduction

A travers ce TP, nous allons apprendre à manipuler des objets 3D en utilisant notamment la librairie **Pygame** ainsi que **OpenGL**. Ce TP est à réaliser en 2 séances.
Le but de ce TP est de représenter une maison à partir de différentes classes qui interagiront entre elles.

Le module Pygame permet, entre autres, de construire des jeux en *Python*. Pour traiter des jeux, le programme doit gérer des événements comme l'appui sur des boutons. Pygame sera utilisé afin de gérer l'affichage d'une fenêtre graphique et des interactions entre celle-ci et l'utilisateur.

Le module PyOpenGL permet d'accéder en *Python* aux très nombreuses fonctions de visualisation de la bibliothèque OpenGL dédiée à l'affichage de scènes tridimensionnelles à partir de simples primitives géométriques. Il sera par exemple utiliser pour construire et afficher les objets 3D sur la fenêtre graphique.

I/ Travail préparatoire

Cette partie est à réaliser à partir d'un fichier que l'on crée pour se familiariser avec les librairies citées précédemment.

Utilisation de Pygame

Abordons dans un premier temps la librairie Pygame.

1)

Nous devions d'abord utiliser une petite partie de code afin de comprendre comment l'affichage de Pygame fonctionne. La partie de code en question était celle-ci :

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

Premièrement, nous importons la librairie pygame. Puis on initialise tous les modules pygame importés. Ensuite, nous définissons une fenêtre pygame de taille 300 sur 200. (Ici nous l'appelons ecran). Enfin, nous désinitialisons tous les modules pygame.

Nous observons que la fenêtre ne s'affiche pas. Cependant nous pouvons apercevoir brièvement l'icône de la fenêtre pygame dans la barre de tâche.

2)

Dans un second temps, nous avons analysé un code plus complet :

```
import pygame


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

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

pygame.quit()
```

Cette fois ci, la fenêtre s'ouvre correctement puisqu'il y a une condition de fermeture. Le code utilise une boucle while à partir d'un boolean *continuer* que nous définissons comme True. Celui-ci sera modifié comme False si nous appuyons sur n'importe quelle touche. En effet, le code vérifie le type d'évènement et s'il est bien égal à une pression de touche.

Utilisation de Pyopengl pour représenter des objets 3D

Puis dans un second temps, nous allons travailler sur la librairie OpenGL.

1)

Aussi, nous utilisons une partie de code afin de comprendre cette fois-ci le fonctionnement de OpenGL. La partie de code en question était celle-ci :

```
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.

    while True:
        for event in pygame.event.get():
                if event.type == pygame.QUIT:
                        pygame.quit()
                exit()
```

NICOLAS Thomas
GUERIOT Benjamin

Nous avons ajouter la fonction **gluPerspective()** qui permet de projeter une matrice de perspective.

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
```

Le 45 représente le champ de vision.
Le second paramètre représente le ratio d'aspect déterminant le champ dans la direction x.
Le paramètre suivant est le plan de clipping near
Enfin la valeur 50.0 représente le plan de clipping far.

2)

Pour pouvoir tracer une ligne à l'aide de OpenGL, il faut indiquer que nous débutons le tracer en mode ligne grâce à **glBegin()**.
Ensuite, **glColor3fv()** indique la couleur des vertices suivants.
Puis, nous indiquons 2 vertices symbolisant le point de départ et le point d'arrivé, ce qui créer le segment. (nous utilisons la fonction **glVertex3fv()**)
Enfin, il faut terminer le tracer avec **glEnd()**.

Voici le code d'exemple que nous avons utilisé :

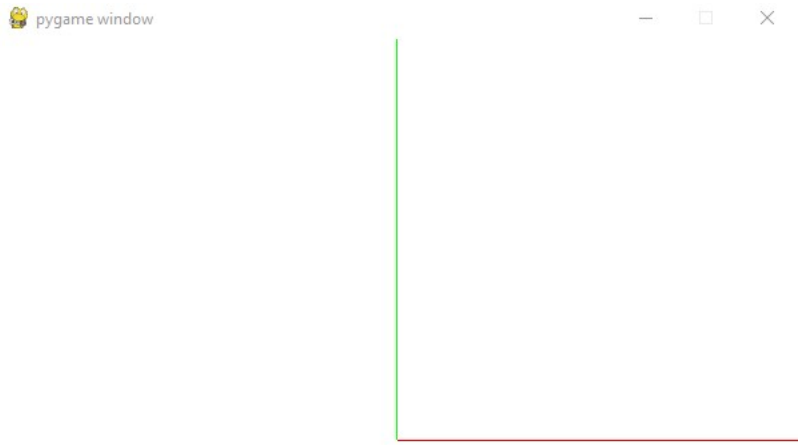
```
gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
gl.glColor3fv([0, 0, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 1, -2)) # Deuxième vertice : fin de la ligne
gl.glEnd()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

Il fallait utiliser ce code pour tracer les axes x, y et z avec des couleurs définies.

```
# Axe des X
gl.glColor3fv([1, 0, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne

# Axe des Y
gl.glColor3fv([0, 1, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne

# Axe des Z
gl.glColor3fv([0, 0, 1]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 0, -3)) # Deuxième vertice : fin de la ligne
```

NICOLAS Thomas
GUERIOT Benjamin

Voici l'affichage obtenu à l'aide de la fenêtre :



Nous observons l'axe des x en rouge ainsi que l'axe des y en vert.
Il faut noter que nous ne pouvons pas voir l'axe des z actuellement car il est en face du point du vu que nous avons.

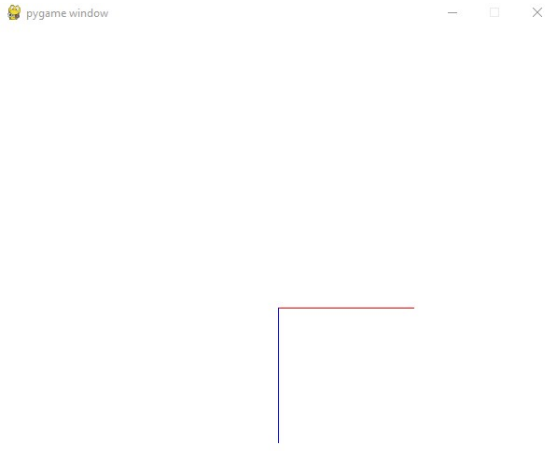
3)

Nous allons maintenant translater et effectuer un mouvement de rotation sur nos axes afin de pouvoir les observer sous différents angles. Pour cela, nous utilisons les fonctions **glTranslatef()** et **glRotatef()**.

Le code d'exemple mis à disposition était :

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
gl.glTranslatef(0.0, 2, -5)
gl.glRotatef(-90, 1, 0, 0)
```

Nous avons copié cette partie et l'avons collé dans notre code. Nous avons alors observé l'affichage graphique obtenu :



Nous remarquons que grâce à la rotation de 90°, nous pouvons désormais observer l'axe z (en bleu).

NICOLAS Thomas
GUERIOT Benjamin

Découverte de l'environnement du travail du TP

Nous utilisons maintenant les fichiers fournis pour réaliser le TP. Ceux-ci contiennent les différentes classes que nous allons utiliser dans la suite de ce TP.

(1). a)

Nous devions ici, d'ajouter la commande return Configuration() dans une fonction du fichier main. Lorsque nous exécutons notre programme, la fenêtre de Pygame nous affiche les 3 axes x, y et z. Nous obtenons le même affichage que la question 2) de la partie précédente. Cependant les touches « a », « z » et « Z » nous permettent respectivement d'afficher ou non les axes, de tourner les axes dans un sens et enfin de tourner les axes dans l'autre sens.

Le fichier main contient une fonction **main()** qui instancie un objet à partir de différentes fonctions (ici comme nous somme à la première partie, c'est Q1a()). Ces fonctions retournent des objets de types de Configuration. Puis nous utilisons la fonction **display()** de l'objet instancié.

A noter que nous importons toutes futures classes qui servirons à la réalisation de ce TP.

Passons désormais au fichier Configuration. Celui-ci contient la classe Configuration qui possède de nombreuses fonctions que nous traiterons par la suite. Il faut noter cependant, que pour initialiser un objet Configuration, il faut ajouter en paramètre un dictionnaire comprenant des informations comme les axes, la couleurs de ceux-ci ainsi que la position de l'écran. En outre, toujours dans l'initialisation, on initialise Pygame, OpenGL, TransformationMatrix et une liste d'objets comme vide. Enfin, on génère la liste des coordonnées des axes.

```
# Initializes PyGame
self.initializePyGame()

# Initializes OpenGl
self.initializeOpenGL()

# Initializes the tranformation matrix
self.initializeTransformationMatrix()

# Initializes the object list
self.objects = []

# Generates coordinates
self.generateCoordinates()
```

Aussi, la fonction **draw()** permet de tracer les axes x, y et z suivants les paramètres renseignés lors de l'instanciation.

De plus, l'une des fonctions de cette classe est display. D'abord celle-ci appelle la fonction draw() pour tracer les axes. Ensuite, elle affiche dans une fenêtre pygame ceux-ci. Une boucle infinie lui permet de vérifier les différents types d'évènements qui peuvent survenir.

(1). b)

Nous avons réalisé une modification dans notre main afin de remarquer des changements possibles.

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

Ici, lors de l'instanciation, nous définissions les paramètres screenPosition et xAxisColor ce que changement la position de l'écran ainsi que la couleur de l'axe x.

Puis nous avons utilisé les setters **setParameter()** :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisCo lor', [0,1,1]). \
    display()
```

Un setter permet de modifier les valeurs d'un paramètre sans pour autant l'avoir définit lors de l'instanciation.

Dans ce cas-ci, le premier setParameter() était inutile puisque nous avions déjà spécifié la couleur de l'axe x dans les paramètres d'instanciation.

Aussi, le chaînage des méthodes ***setParameter()*** et ***display()*** est possible car ces fonctions se situent dans la classe Configuration. Donc ceci revient à instancier un objet et utiliser les fonctions mis à disposition.

Cela revient par exemple à faire :

```
c = Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]})
c.setParameter('xAxisColor', [1, 1, 0])
c.setParameter('yAxisColor', [0,1,1])
return c.display()
```

Un traitement particulier est effectué dans le « setter » pour le paramètre ***screenPosition***. En effet, si nous changeons la distance de l'écran, il faut recalculer la nouvelle perspective. C'est pourquoi, cette fonction appelle *initializeTransformationMatrix*.

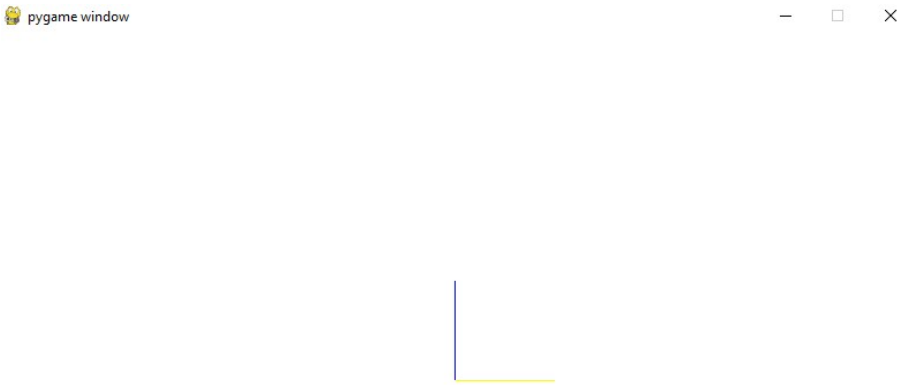
(1). c)

Afin de pouvoir représenter l'axe z verticalement et l'axe x horizontalement, nous devons effectuer une rotation autour de l'axe x. Par conséquent, nous ajoutons la ligne de code suivante dans la méthode ***initializeTransformationMatrix()***.

```
gl.glRotatef(-90, 1, 0, 0)
```

NICOLAS Thomas
GUERIOT Benjamin

Voici le résultat obtenu lors de l'affichage graphique :



Nous observons bien x (en jaune) horizontalement et z (en bleu) verticalement.

II/ Mise en place des interactions avec l'utilisateur avec Pygame

Dans cette partie, nous allons gérer les interactions avec l'utilisateur.

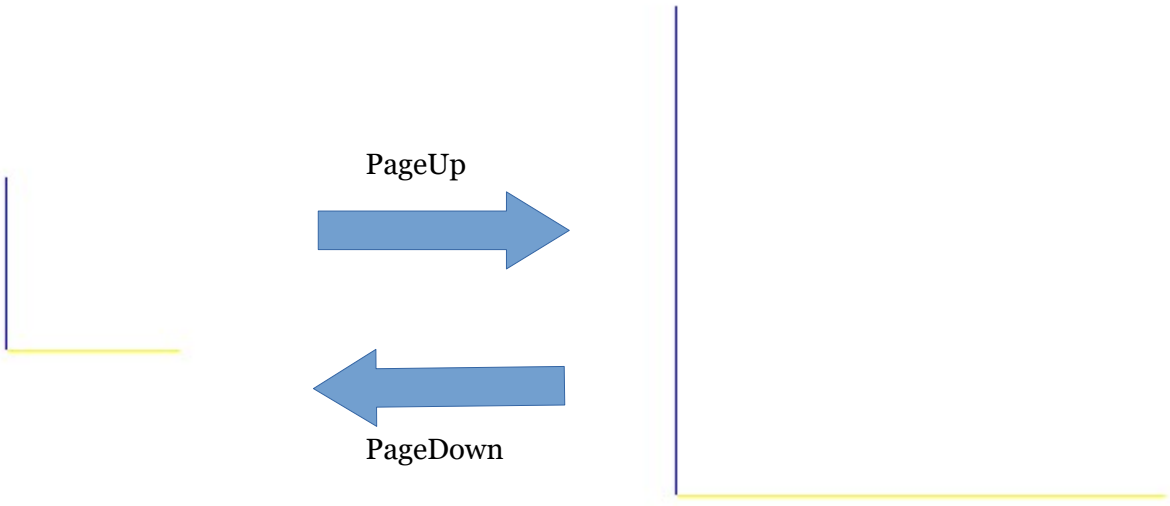
(1). d)

Nous avons ajouté les gestions des touches « Page Up », « Page Down » pour, respectivement, grossir ou réduire l'affichage. Lorsque la pression sur une touche est détectée, la méthode **processKeyDownEvent()** est appelée. Nous vérifions ainsi si la touche appuyée est « Page Up » ou « Page Down ». Si celle-ci est bel est bien pressée, nous changeons l'échelle.

La fonction devient ainsi :

```
def processKeyDownEvent(self):

    # Fait une rotation autour de z
    # Vérification des touches
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key ==
        gl.glRotate(-2.5, 0, 0, 1)   # Rotation
    # Vérification des touches
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)    # Rotation

    # Trace ou supprime les axes de référence
    # Vérification des touches
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        # Vérification que les axes ne sont pas nuls
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)

    # Vérification de la touche PageUp
    elif (self.event.key == pygame.K_PAGEUP) :
        # Mise à l'échelle de facteur 1.1
        gl.glScalef(1.1, 1.1, 1.1)

    # Vérification de la touche PageDown
    elif (self.event.key == pygame.K_PAGEDOWN) :
        # Mise à l'échelle de facteur 1/1.1
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

Et voici un exemple du changement observé avec l'interface graphique :



(1). e)

Dans cette partie, nous nous intéressons à l'effet de zoom. Cette fois-ci, il fallait détecter lorsque l'utilisateur utilise la mollette de la souris. Pour cela, on utilise l'attribut button qui prend la valeur 4 lorsque le mouvement de la mollette est vers le haut et 5 lorsque celui-ci est vers le bas.

Voici la fonction que nous avons réalisé :

```
def processMouseButtonDownEvent(self):

    # Verifie si le mouvement de la mollette est vers le haut
    if self.event.button == 4 :
        # Mise à l'échelle de facteur 1.1
        gl.glScalef(1.1, 1.1, 1.1)

    # Verifie si le mouvement de la mollette est vers le bas
    elif self.event.button == 5 :
        # Mise à l'échelle de facteur 1/1.1
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

(1). f)

Nous souhaitons maintenant gérer le déplacement des objets. Le bouton gauche de la souris nous permettra de réaliser une rotation alors que le droit réalisera une translation.

Voici le code de notre fonction :

```
def processMouseMotionEvent(self):

    # On verifie si le pouton pressé est le gauche
    if pygame.mouse.get_pressed()[0] == 1 :

        # On récupère le déplacement en x et y
        x = self.event.rel[0]
        y = self.event.rel[1]
        # On effectue une rotation
        gl.glRotate(1, x, 1, y)

    # On verifie si le pouton pressé est le droit
    elif pygame.mouse.get_pressed()[2] == 1 :
        # On récupère le déplacement en x et y puis on divise
        # par 100 pour que le changement ne soit pas trop brutal
        x = self.event.rel[0]/100
        y = self.event.rel[1]/100
        # On effectue une translation en x et y
        gl.glTranslatef(x, 0, y)
```

NICOLAS Thomas
GUERIOT Benjamin

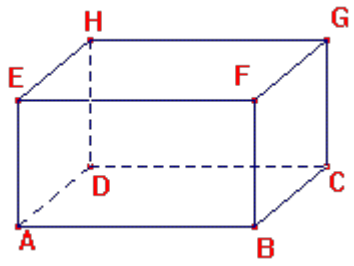
IV - Création d'une section

Nous nous intéressons désormais à la création d'une section. Pour ce faire, nous allons utiliser le fichier section qui contient la classe Section.

(2) .a)

Nous avons écrit la méthode generate(self) qui crée les sommets et les faces d'une section orientée selon l'axe x.

Pour représenter la section que nous allons réaliser, voici un schéma sur lequel notre méthode se base.



Voici désormais le code correspondant :

```
def generate(self):
    # Définie tous les points à partir des paramètres
    self.vertices = [
            # Point A
            self.parameters['position'],
            # Point E
            [self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2] + self.parameters['height']],
            # Point F
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2] + self.parameters['height']],
            # Point B
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]],
            # Point D
            [self.parameters['position'][0], self.parameters['position'][1] + self.parameters['thickness'], self.parameters['position'][2]],
            # Point H
            [self.parameters['position'][0], self.parameters['position'][1] + self.parameters['thickness'], self.parameters['position'][2] + self.parameters['height']],
            # Point G
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2] + sel
            # Point C
            [self.parameters['position'][0] + self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'],self.parameters['position'][2] ]
            ]

    # Définie toutes les faces
    self.faces = [
            [0,3,2,1], # Face AEFB
            [0,4,5,1], # Face ADHE
            [4,7,6,5], # Face DCGH
            [3,7,6,2], # Face BCGF
            [0,3,7,4], # Face ABCD
            [1,2,6,5]  # Face EFGH
            ]
```

NICOLAS Thomas
GUERIOT Benjamin

FIN DE SEANCE 1