MOREL Rémi
DUMIRE Thibault
Date du TP (09/12/2021)

# Rapport de TP3 – Représentation visuelle d'objet 3D

### Introduction

Nous allons vous présenter notre compte rendu de notre TP qui a pour objectif la représentation visuelle d'objet 3D. Pour réaliser ceci, nous allons utiliser les différents fichiers mis à notre disposition. Qui à la fin, pourra nous permettre de voir une image 3D et de pouvoir réaliser des zooms, des translations et des rotations. Pour l'image 3D nous allons réaliser une maison qu'on pourra observer. Pour cela on va réaliser à l'aide d'objets simples que l'on va construire comme des murs, des portes, des fenêtres, …

## I. Préparation avant TP

### 1. Utilisation de Pygame

Nous avons copié le code, nous avons observé l'ouverture d'une fenêtre de dimension 200x300 et sa fermeture juste après car on lui demande de se fermer à l'aide de ligne 4.

Nous avons ensuite copié le second code qui nous permet de fermer la fenêtre en appuyant sur une touche du clavier. Ce code enregistre les différents évènements et vérifie s'il y a un évènement où on a appuyé sur une touche de notre clavier à l'aide de la commande « pygame.KEYDOM » une fois validé on sort de la boucle d'enregistrement des évènement et il ferme la fenêtre.

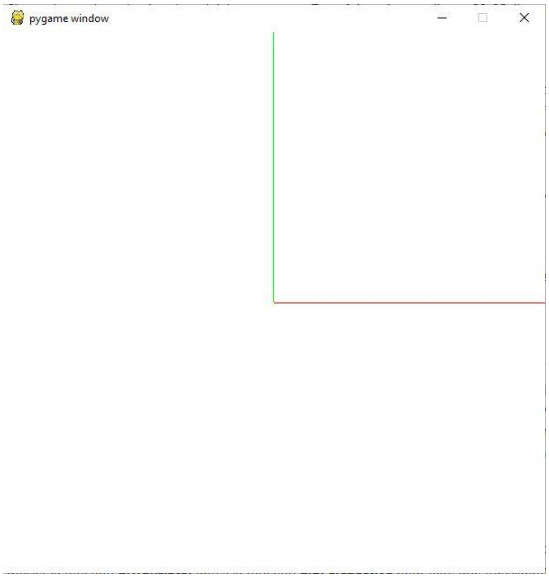
```
1   import pygame
2
3
4   pygame.init()
5   ecran = pygame.display.set_mode((300, 200))
6
7   continuer = True
8   while continuer:
9       for event in pygame.event.get():
10          if event.type == pygame.KEYDOWN:
11              continuer = False
12
13  pygame.quit()
```

## 2. Utilisation de Pyopengl

Pour cette partie nous avons réalisé les différentes commandes pour tracer les différents axes représentant l'axe x, y et z pour pouvoir visualiser les différents axes à l'aide du code suivant :

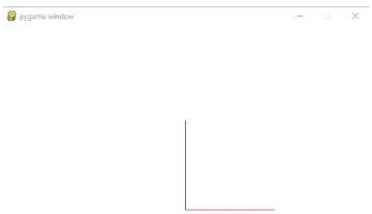
```
import pygame
import OpenGL.GL as gl
import OpenGL.GLU as glu

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(0,2,-5)
    gl.glRotatef(-90,1,0,0)

    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode li

    gl.glColor3fv([255, 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

    gl.glColor3fv([0, 255, 0])
    gl.glVertex3fv((0,0, -2))
    gl.glVertex3fv((0,1, -2))

    gl.glColor3fv([0, 0, 255])
    gl.glVertex3fv((0,0, -2))
    gl.glVertex3fv((0,0, -1))

    gl.glEnd() # Find du tracé
```

On obtient en premier cette photo où on peut observer l'axe x et y :




On a rajouté les commande gl.glTranslatef(0,2,-5) pour recentrer l'image et gl.glRotatef(-90,1,0,0) pour réaliser une rotation de -90° autour de l'axe x pour pouvoir observer l'axe z.

On obtient l'image suivante :



### 3. Découverte de l'environnement de travail du TP

- 1a) On a ajouté la fonction Q1a() au fichier main celle-ci lance la classe configuration du fichier configuration.py. A l'intérieur de cette classe il y a plusieurs méthodes permettant l'affichage d'une fenêtre graphique avec tracé les axes. Dans cette classe il y a une méthode permettant de réaliser des rotations selon l'axe z en appuyant soit sur « z » soit sur « Z » dans un sens puis dans l'autre. On a aussi une méthode pour la touche « a » qui permet de mettre en pose la fenêtre graphique pendant un temps défini en paramètre. (affiche une fenêtre graphique blanche)

- 1b)

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

En Initialisant les paramètre d'un objet de classe Configuration on obtient une fenêtre graphique avec deux axes qui sont plus proche que la configuration par défaut et elle change la couleur de l'axe x passant de la couleur rouge à la couleur jaune.



Le chainage entre setParameter() et display() est possible car la méthode setParameter() renvoie self un objet de la classe configuration.

Le traitement particulier dans le setter pour le paramètre screenPosition est due au fait que lorsque l'on change la position de l'écran il faut changer l'affichage dans la fenêtre graphique et garder le centrage.

- 1c) On a rajouté une instruction dans la méthode initializeTransformationMatrix(). L'instruction est la suivante : gl.glRotatef(-90,1,0,0) après gl.glTranslatef pour avoir une rotation de -90° selon l'axe x pour avoir l'axe des z verticale et que l'axe des y soit notre profondeur.

## II. Mise en place des interactions avec l'utilisateur avec Pygame

- 1d) Nous avons ajouté des instructions dans la méthode processKeyDownEvent pour pouvoir assurer la gestion des touches « Page Up » et « Page Down » qui nous permettra par la suite un zoom ou un dézoom.

```
elif self.event.key == pygame.K_UP:
    gl.glTranslatef(0,-0.1,0)

elif self.event.key == pygame.K_DOWN:
    gl.glTranslatef(0,0.1,0)
```

- 1e) Nous modifions maintenant la méthode processMouseButtonDownEvent pour pouvoir zoomer ou dézoomer à l'aide de la souris. Pour cela nous avons mis la méthode suivante :

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.type == pygame.MOUSEBUTTONDOWN:
        if (self.event.button == 4):
            gl.glTranslatef(0,-0.1,0)
        elif (self.event.button == 5):
            gl.glTranslatef(0,0.1,0)
```

Cette méthode vérifie que l'évènement de la molette est vérifié et vérifie le sens de rotation de la molette en vérifiant si le Button4 ou le Button5 est activer. On rajoute ainsi la commande demandée pour chaque condition.

- 1f) Pour cette question, nous allons modifier la méthode processMouseMotionEvent pour obtenir une rotation et une translation selon les touches de la souris et du déplacement de la souris.

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if self.event.type == pygame.MOUSEMOTION:
        if (pygame.mouse.get_pressed()[0] == 1):
            gl.glRotatef(self.event.rel[0],0,0,1)
            gl.glRotatef(self.event.rel[1],1,0,0)
        if (pygame.mouse.get_pressed()[2] == 1):
            gl.glTranslate(self.event.rel[0]/50,0,0)
            gl.glTranslate(0,0,-self.event.rel[1]/50)
```

On a choisi dans les commandes de translation de diviser par 50 le déplacement en x et en y pour avoir une sensibilité plus faible et ainsi ne pas créer une translation trop importante.

## III. Création d'une section

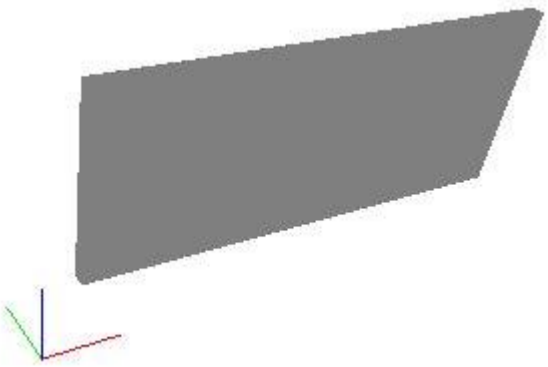
Maintenant que nous avons fini la préparation, nous allons passer à la création des sections qui plus tard nous permettront de faire les murs. Dans un premier temps, nous devions initialiser les différents sommets de nos faces à l'aide des différents paramètres de la section, la hauteur, la largeur et la longueur en fonction de la position du premier point qui est notre origine. Pour cela on a complété la méthode generate(self) dans la classe section :

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [
            self.parameters['position'],
            [self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]],
            [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
            [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]]
            ]

    self.faces = [
            [0,1,2,3],
            [2,3,4,5],
            [4,5,6,7],
            [0,1,6,7],
            [1,2,5,6],
            [0,3,4,7]
            ]
```

Une fois que cette méthode a été réalisé nous devions pourvoir visualiser la section. Dans un premier temps il faut ajouter l'objet section à la classe configuration pour pouvoir l'afficher. C'est la commande Configuration().add(section).display() dans le main qui réalise ceci. Il faut ensuite pourvoir définir les faces qu'il va dessiner, on complète donc la méthode draw(self) dans la classe section pour pouvoir observer et remplir d'une couleur les faces :

```
# Draws the faces
def draw(self):

    if (self.parameters['edges']==True):
        self.drawEdges()

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL

    gl.glPushMatrix()

    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)

    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv(self.parameters['color'])
    for i in self.faces:
        for j in i:
            gl.glVertex3fv(self.vertices[j])
    gl.glEnd()

    gl.glPopMatrix()
```

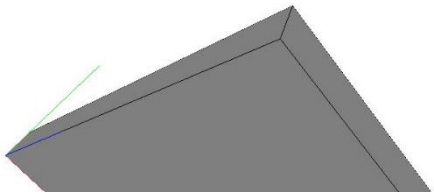
Voici notre section :



On réalise ensuite la même méthode drawEdges() pour pouvoir observer les arêtes :

```
# Draws the edges
def drawEdges(self):
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)

    gl.glPushMatrix()

    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0,0,0])
    for i in self.faces:
        for j in i:
            gl.glVertex3fv(self.vertices[j])
    gl.glEnd()

    gl.glPopMatrix()
```

Image :



## IV. Création des murs

Maintenant que nous avons réalisé la classe Section on peut utiliser ses objets pour pouvoir réaliser des murs objets de la classe Wall.

La classe Wall possède 6 paramètres qui sont les mêmes paramètres que la section. Pour l'instant les murs ne possèdent qu'un type d'objet : une section.

## V. Création d'une maison

## VI. Création d'ouverture

## VII. Conclusion

Grâce à ce TP on a pu apprendre comment réaliser le contrôle d'une fenêtre graphique par exemple le zoom, le déplacement, la rotation, etc... Nous n'avons pas réussi à terminer ce TP nous avons rencontré plusieurs problèmes notamment au niveau de la classe Wall où nous n'avons pas réussi à placer les murs correctement une fois dans la classe House du fait qu'il manque la commande de translation dans la classe Section.