DELICOURT Raphaël
DELESALLE Thibault
15/12/2021

# Rapport de TP3 – Représentation visuelle d'objets

## I. Introduction

Le but de ce TP va être de représenter des objets en 3D grâce aux modules Pygame et PyOpenGL de Python. Nous allons utiliser pour cela différentes fonctions qui utilisent des matrices 4x4.

## II. Préparation à faire avant le TP

### 1. Question 1a

Fichier Q1 Main.py : Le fichier main contient une fonction main qui exécute la classe Configuration.
Fichier Q1Configuration.py : Le fichier configuration contient une classe Configuration avec plusieurs méthodes.
La première méthode sert à initialiser la classe. Elle va fixer les couleurs des axes x, y et z respectivement sur du rouge, du vert et du bleu et aussi la position de l'écran sur l'axe z a -10.
La méthode initializePygame permet d'initialiser Pygame, de définir la taille de l'écran et de l'afficher. La méthode initializeOpenGL met l'écran en blanc. TransformationMatrix écrit la matrice de translation avec comme valeur a=0, b=0 et c égal à-10 c'est-à-dire la valeur de la position de l'écran.
La méthode generateCoordinates permet de créer un repère (O,x,y,z). La méthode add ajoute un objet à la liste objects. La méthode draw dessine les axes avec les bonnes couleurs et les objets de notre liste. Au début, de notre méthode on dit que l'on veut dessiner des lignes grâce à GL_LINES.
Les dernières méthodes vont servir à établir les différentes actions possibles avec le clavier concernant la manipulation des axes. En effet, la première méthode va permettre de faire pivoter notre objet dans le sens négatif en appuyant sur Z et dans le sens positif en appuyant sur z.
Cette méthode va aussi faire disparaitre nos axes en appuyant sur « a » (et les faire réapparaitre en appuyant à nouveau). Enfin, la méthode display va afficher tous les objets.

### 2. Question 1b

Le chainage est possible car la méthode setParameter() renvoie un objet (self). Comme Configuration est un objet, Configuration.setParameter() reverra un objet donc on pourra faire Configuration.setParameter().setParameter().
Le paramètre screenPosition étant modifiée, on refait alors l'initialisation de la matrice Transformation car elle dépend du paramètre screenPosition.

### 3. Question 1c

La méthode initializeTransformationMatrix sert à faire un changement d'axe pour passer de y à z. On utilise pour cela glRotatef(theta,a,b,c) pour effectuer une rotation de 90° autour de l'axe x de coordonnées a=1 ; b=0 ; c=0.

```
69      # Initializes the tranformation matrix
70      def initializeTransformationMatrix(self):
71          gl.glMatrixMode(gl.GL_PROJECTION)
72          gl.glLoadIdentity()
73          glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)
74
75          gl.glMatrixMode(gl.GL_MODELVIEW)
76          gl.glLoadIdentity()
77          gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
78          gl.glRotatef(-90,1,0,0)
```

Sur la fenêtre pygame, on voit que la modification a bien eu lieu : l'axe horizontal est toujours rouge mais l'axe vertical n'est plus vert mais bleu.

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

### 1. Question 1d

Tout d'abord, on réalise la rotation avec la touche z du clavier. On applique une rotation dans le sens négatif si la touche Z est pressée (qui vaut -2.5) et une rotation positive si la touche z est pressée (qui vaut 2.5). Ensuite, on ajoute la condition pour laquelle si a est pressé, les axes disparaissent (ou réapparaissent). Enfin, pour zoomer et dézoomer on va utiliser la fonction glScalef() qui permet de changer d'échelle. Si on appuie sur Page Up on zoome, si on appuie sur Page Down on dézoome.

```
139     def processKeyDownEvent(self):
140         # Rotates around the z-axis
141         if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z)
142             gl.glRotate(-2.5, 0, 0, 1)
143         elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
144             gl.glRotate(2.5, 0, 0, 1)
145 
146         # Draws or suppresses the reference frame
147         elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
148             self.parameters['axes'] = not self.parameters['axes']
149             pygame.time.wait(300)
150 
151         #Zoom
152         elif self.event.key == pygame.K_PAGEUP :
153             gl.glScale(1.1,1.1,1.1)
154 
155         elif self.event.key == pygame.K_PAGEDOWN :
156             gl.glScale(1/1.1,1/1.1,1/1.1)
```

### 2. Question 1e.

Pour cette question, on réutilise glScalef() mais cette fois on écrit dans la méthode MouseButtonDownEvent car on va utiliser la mollette de la souris pour zoomer. L'action de la mollette est symbolisée par l'attribut button. Si button est égal 4 on va zoomer et s'il est égal à 5 on va dézoomer.

```
158     # Processes the MOUSEBUTTONDOWN event
159     def processMouseButtonDownEvent(self):
160         if self.event.button == 4:
161             gl.glScale(1.1,1.1,1.1)
162         elif self.event.button == 5:
163             gl.glScale(1/1.1,1/1.1,1/1.1)
```

### 3. Question 1f

Dans cette question, on veut utiliser la souris pour déplacer les axes. Si on se sert du bouton gauche, on fait une rotation et si on sert su bouton de droite, on fait une translation.

D'abord, on vérifie que le bouton est bien pressé. Si celui-ci est pressé, on va utiliser les fonctions glRotatef() et glTranslatef().

Pour la première rotation, on prend un angle de rotation égal au déplacement en x et qui va tourner par rapport à l'axe z (0,0,1). On fait de même avec l'axe y (0,1,0) pour la deuxième rotation.

Pour la première translation, on définit le vecteur translation avec le déplacement en x ; on fait de même pour le déplacement selon z (0, 0, déplacement en y).

```python
    # Processes the MOUSEMOTION event
    def processMouseMotionEvent(self):
        if pygame.mouse.get_pressed()[0]==1:
            gl.glRotatef(self.event.rel[0],0,0,1)
            gl.glRotatef(self.event.rel[1],0,1,0)

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

## IV. Création d'une section

### 1. Question 2a

Dans la partie « vertice », on rentre les coordonnées des 8 coins du parallélépipède et dans la partie « face », on rentre ses 6 faces en prenant les rangs de la liste des 4 coins de la face voulue, définis juste avant.

```python
    def generate(self):
        self.vertices = [
                [0, 0, 0 ],
                [0, 0, self.parameters['height']],
                [self.parameters['width'], 0, self.parameters['height']],
                [self.parameters['width'], 0, 0],
                [0,self.parameters['thickness'],0],
                [self.parameters['width'],self.parameters['thickness'],0],
                [0,self.parameters['thickness'],self.parameters['height']],
                [self.parameters['width'],self.parameters['thickness'],self.parameters['height']]
                ]
        self.faces = [
                [0,1,3,2],
                [2,7,5,3],
                [6,7,5,4],
                [1,6,4,0],
                [6,7,2,1],
                [4,5,3,0],
                ]
```

### 2. Question 2b

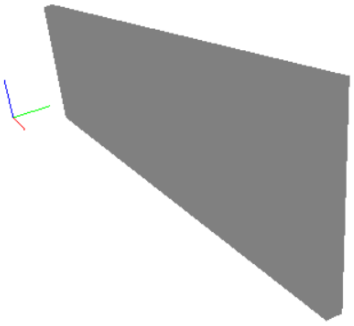
Dans la méthode draw de la classe Section, on crée une boucle for qui parcoure toutes les faces et qui, pour chaque face, sélectionne les 4 sommets pour ensuite remplir les espaces.

```python
    # Draws the faces
    def draw(self):
        for i in self.faces:
            gl.glPushMatrix()

            gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
            gl.glRotatef(self.parameters['orientation'],0,0,1)
            gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)


            gl.glBegin(gl.GL_QUADS)
            gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
            gl.glVertex3fv(self.vertices [i[0]])
            gl.glVertex3fv(self.vertices [i[1]])
            gl.glVertex3fv(self.vertices [i[2]])
            gl.glVertex3fv(self.vertices [i[3]])
            gl.glEnd()

            gl.glPopMatrix()
```



## 3. Question 2c

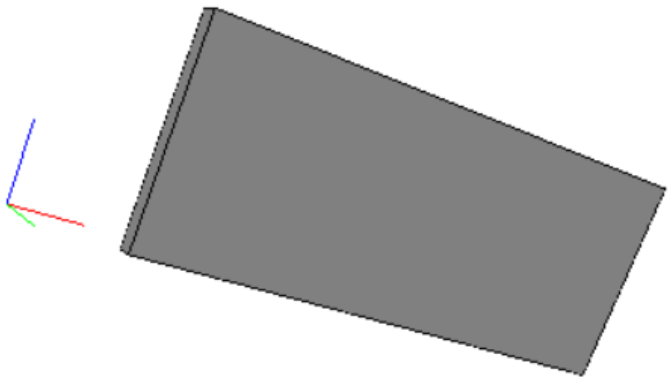
Pour drawEdges, on remplace simplement FILL par LINE et pour la couleur, on met la teinte la plus sombre pour que les arêtes ressortent, c'est-à-dire à 0.

```
86      # Draws the edges
87      def drawEdges(self):
88          for i in self.faces:
89              gl.glPushMatrix()
90
91              gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
92              gl.glRotatef(self.parameters['orientation'],0,0,1)
93              gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)
94
95              gl.glBegin(gl.GL_QUADS)
96              gl.glColor3fv([0, 0, 0]) # Couleur plus sombre
97              gl.glVertex3fv(self.vertices [i[0]])
98              gl.glVertex3fv(self.vertices [i[1]])
99              gl.glVertex3fv(self.vertices [i[2]])
100             gl.glVertex3fv(self.vertices [i[3]])
101             gl.glEnd()
102
103             gl.glPopMatrix()
```

Pour que la méthode drawEdges soit exécutée en premier, on ajoute une condition if dans la méthode draw.

```
105     # Draws the faces
106     def draw(self):
107         if self.parameters['edges']:
108             self.drawEdges()
109
```

Voici la figure affichée, avec les arêtes en noir.



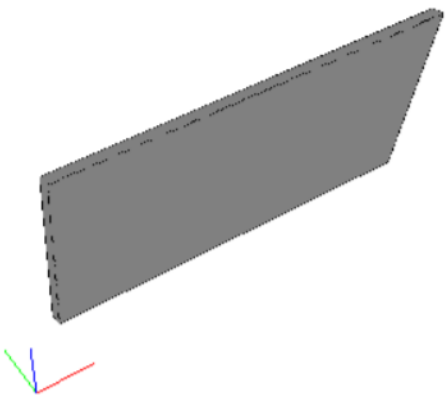
# V. Création des murs

## 1. Question 3a

Dans la classe Wall, on reprend la trame de la méthode draw de Section et on stocke la matrice pour ensuite dessiner ce qui se trouve dans la liste objets.

```
70      # Draws the faces
71      def draw(self):
72          for j in self.objects:
73
74              gl.glPushMatrix()
75
76              gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
77              gl.glRotatef(self.parameters['orientation'],0,0,1)
78
79
80              j.draw()
81              j.drawEdges()
82
83              gl.glPopMatrix()
```

Voici la figure obtenue.



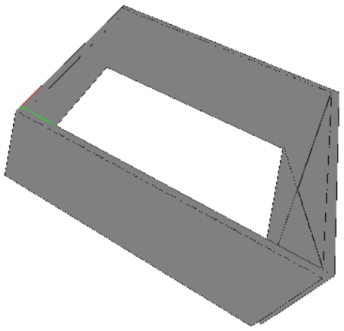
# VI. Création d'une maison

## 1. Question 4a

Dans le main, on crée nos 4 murs, tous de mêmes largeurs et hauteurs (ici 7 et 2.6). Le premier mur part de l'origine, le deuxième aussi mais avec une orientation de 90°. Le troisième est parallèle au premier mais part du point (0,7,0) c'est-à-dire à la même largeur que l'autre mur. Le quatrième mur est parallèle au deuxième (avec la même orientation de 90°) et au point (0,7,0).

```
38  def Q4a():
39      wall1 = Wall({'position':[0,0,0], 'width' : 7, 'height':2.6, 'edges':True})
40      wall2 = Wall({'position':[0,0,0], 'width' : 7, 'height':2.6, 'edges': True, 'orientation':90})
41      wall3 = Wall({'position':[0,7,0], 'width' : 7, 'height':2.6, 'edges': True})
42      wall4 = Wall({'position':[0,-7,0], 'width' : 7, 'height':2.6, 'edges': True,'orientation':90})
43      house = House()
44      house.add(wall1).add(wall3).add(wall4).add(wall2)
45      return Configuration().add(house)
```

La figure obtenue est la suivante :



## VII. Conclusion

Pour conclure, ce TP nous a permis d'apprendre des notions de base dans la création d'objets 3D en Python. Nous n'avons pas pu traiter la partie sur les ouvertures par manque de temps. Nous avons pu néanmoins, réaliser une maison de 4 murs, de façon guidée et intuitive.