PONTAUD Thomas
MANOUKIAN Maxime
Date du TP : 07/12/2021

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Le but de ce TP est de générer différents affichages graphiques à l'aide d'outils déjà présents sur Python pour afficher et déplacer une maison représenté dans un repère en 3D.

II. Section 1 du TP

1. Utilisation de Pygame

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

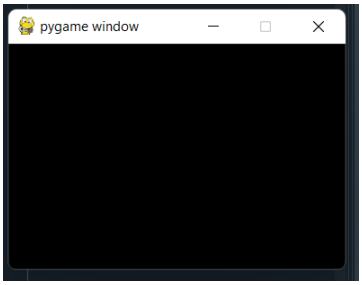
1) Ces lignes de commandes permettent d'initialiser une fenêtre pygame, ici de 300*200 pixels. Cette fenêtre se ferme quasiment instantanément.

```
In [8]: runfile('C:/Users/thoma/OneDrive/FI3 S5/INFO501/TP2/tp2-reconnaissance-chiffres-tp2_manoukian_pontaud-main 2/src/untitled0.py', wdir='C:/Users/thoma/OneDrive/FI3 S5/INFO501/TP2/tp2-reconnaissance-chiffres-tp2_manoukian_pontaud-main 2/src')
pygame 2.1.0 (SDL 2.0.16, Python 3.8.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

```
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()
```

2) Ces lignes de commande permettent d'ouvrir une fenêtre pygame et de la laisser ouverte tant que l'utilisateur n'a pas appuyé sur une touche du clavier. Ce ci est géré par la boucle while qui tourne tant que l'utilisateur n'a appuyé sur aucune touche en changeant la condition de sortie « continuer » dans ce cas, la fenêtre se ferme alors.



2. Utilisation de PyopenGL

```
8  import pygame
9  import OpenGL.GL as gl
10 import OpenGL.GLU as glu
11
12 if __name__ == '__main__':
13     pygame.init()
14     display=(600,600)
15     pygame.display.set_mode(display, pygame.DOUBLEBUF | pygame.OPENGL)
16
17     # Sets the screen color (white)
18     gl.glClearColor(1, 1, 1, 1)
19     # Clears the buffers and sets DEPTH_TEST to remove hidden surfaces
20     gl.glClear(gl.GL_COLOR_BUFFER_BIT | gl.GL_DEPTH_BUFFER_BIT)
21     gl.glEnable(gl.GL_DEPTH_TEST)
22
23     # Placer ici l'utilisation de gluPerspective.
24     glu.gluPerspective(45, 1, 0.1, 50)
25
26     while True:
27         for event in pygame.event.get():
28             if event.type == pygame.QUIT:
29                 pygame.quit()
30     exit()
31
```

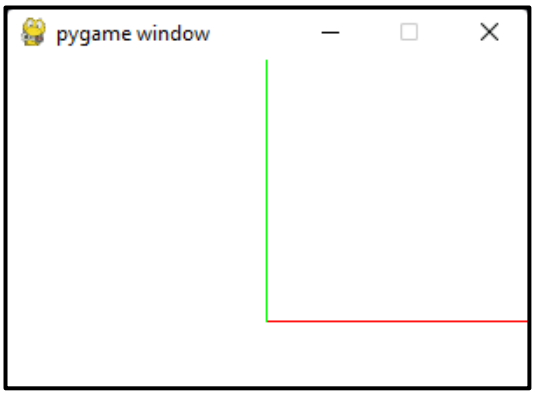
1) Ces lignes de code permettent d'ouvrir une fenêtre Pygame. On crée alors la matrice de perspective avec la fonction gluPerspective à la ligne 24.

Les lignes 26 à 30 permettent de quitter la fenêtre Pygame en cliquant sur la croix.

```
gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochain segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe x
gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de l'axe x
gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochain segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe y
gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de l'axe y
gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochain segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de l'axe z
gl.glVertex3fv((0, 0, -1)) # Deuxième vertice : fin de l'axe z
gl.glEnd() # Find du tracé
pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

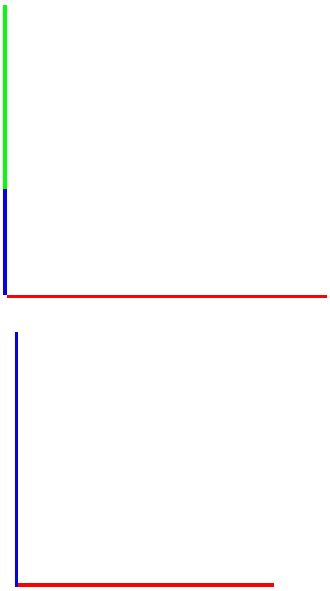
2) On ajoute alors les différents vertices correspondant aux différents axes, précédé de leur couleur au format RGB. Ce qui nous donne l'affichage suivant.

L'axe z normalement tracé en bleu ne se voit pas car il est dans l'axe normal à l'écran.



3) On ajoute la fonction glTranslatef(0,2,-5) pour faire une translation.
Voici le résultat obtenue :

```
# 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)
```



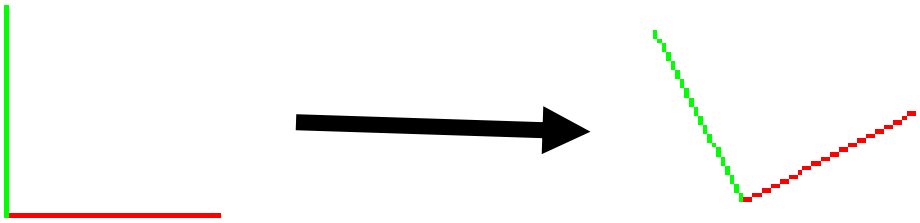
On peut également observer le résultat de la rotation de 90° autour de l'axe x.

3. Découverte de l'environnement de travail du TP

1.a) Lorsque l'on lance le fichier main, la fonction Q1a() lance la classe Configuration qui va elle créer une fenêtre de dimensions 800*600, initialiser les axes etc… Il nous est également possible avec la touche « z » de faire tourner la figure sur le graphique selon l'axe z de 2.5° à chaque appuie. Si on utilise un « Z » majuscule la figure tournera dans le sens contraire toujours selon l'axe z et de 2.5° à chaque fois. La touche « a » nous permet quant à elle de cacher la figure sur le graphique puis de la faire réapparaitre lorsque l'on rappuie sur « a ».

```
# Processes the KEYDOWN event
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)

    # Draws or suppresses the reference frame
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)
```



1.b) Cette commande permet de changer la couleur de l'axe x pour la passer en jaune.

```
In [27]: Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```


Cette méthode setParameter() permet de changer les paramètres des axes x, y et z. Ces fonctions sont accessibles depuis la console car le fichier Configuration est un fichier qui est utilisé dans le main. Le main va aller chercher la fonction setParameter () dans la classe Configuration grâce à cette ligne :

```
from Configuration import Configuration
```

Donc le fait de mettre un point à la fin de Configuration(…). suivis de la fonction setParameter() permet de changer les paramètres de cette configuration, puis de l'afficher avec display(). Ce chainage est possible car setParameter() renvoie un objet de type Configuration.


```
In [41]: Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    ...:     setParameter('xAxisColor', [1, 1, 0]). \
    ...:     setParameter('yAxisColor', [0,1,1]). \
    ...:     display()
```

1.c) Il suffit d'ajouter une fonction de rotation autour de l'axe x à la fonction initializeTransformationMatrix() pour avoir l'axe des z vers le haut (en bleu), l'axe des y en profondeur (en vert) et l'axe des x vers la droite (en rouge). On obtient alors :

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

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

1.d) On ajoute les fonctions de zoom et dézoom sur la figure avec les lignes de code suivantes.

```
elif self.event.key == pygame.K_UP:
    gl.glScalef(1.1, 1.1, 1.1)
elif self.event.key == pygame.K_DOWN:
    gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

1.e) Il est également possible de faire la même chose avec la mollette de la souris en utilisant la fonction ci-contre :

```
def processMouseButtonDownEvent(self):
    if self.event.button==4:
        gl.glScalef(1.1, 1.1, 1.1)
    if self.event.button==5:
        gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

1.f) Nous avons modifié la fonction processMouseMotionEvent() pour que l'on puisse utiliser la souris pour déplacer le graphique et effectuer des rotations.

Pour chaque cas on va regarder si l'utilisateur appuie sur clique droit ou clique gauche et on va utiliser les fonctions glRotate(theta,a,b,c) et glTranslatef(a,b,c) en fonction. On applique un coefficient devant le déplacement de la souris pour ajuster la vitesse de rotation ou de déplacement.

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

    if pygame.mouse.get_pressed()[2]==1:
        gl.glTranslatef(self.event.rel[0]/40, 0, 0)
        gl.glTranslatef( 0, 0,-self.event.rel[1]/40)
```

IV. Création d'une section

2.a) On peut définir chaque arrête du parallélépipède rectangle selon chacun de ses sommets, puis définir chaque face de ce dernier selon ses points. On part de la position initiale de la section et on ajoute les différentes dimensions en fonction de coordonnées.

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

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

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) #
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    for i in range(0,len(self.faces)):
        face=self.faces[i]
        gl.glVertex3fv(self.vertices[face[0]])
        gl.glVertex3fv(self.vertices[face[1]])
        gl.glVertex3fv(self.vertices[face[2]])
        gl.glVertex3fv(self.vertices[face[3]])

    gl.glEnd()


    gl.glPopMatrix()
    return self
```

2.b)
On peut donc créer une boucle for pour créer les différentes faces de la section.

2.c)
Dans la fonction drawEdges nous reprenons la fonction draw en remplaçant gl.GL_FILL par gl.GL_LINE.
Ensuite nous exécutons drawEdges loraque le paramètre « edges » est True.

V. Création des murs

3.a)

Le constructeur de la classe Wall vérifie si chaque paramètre est dans self et si ce n'est pas le car, le paramètre en question est initialisé. Ensuite une liste « objects » qui regroupe différentes sections, qui elles-mêmes sont des dictionnaires de tous les paramètres d'entrée.

Pour la méthode draw, on parcourt les objets de la classe Wall et on les dessine.

```
75      # Draws the faces
76      def draw(self):
77          # A compléter en remplaçant pass par votre code
78          for i in self.objects:
79              i.draw()
```

On a géré la rotation autour de l'axe z directement dans la méthode draw de la classe section :

```
180     def draw(self):
181
182
183         if self.parameters['edges']==True:
184           self.drawEdges()
185         # A compléter en remplaçant pass par votre code
186         gl.glPushMatrix()
187
188         gl.glMatrixMode(gl.GL_MODELVIEW)
189         gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1], self.parameters
            ['position'][2])
190         gl.glRotate(self.parameters['orientation'],0,0,1)
191
```

ici

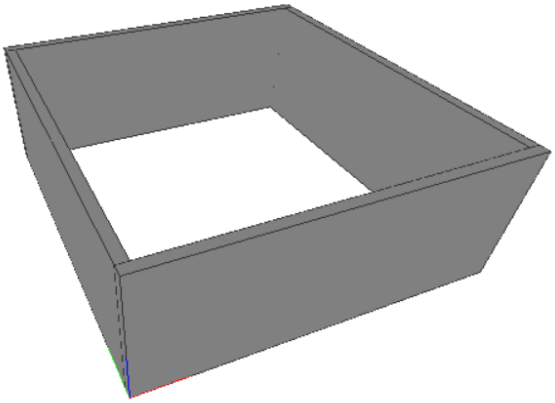
Nous avons ensuite ajouté un Wall dans la classe Configuration dans le fichier main :

```
50  def Q3a():
51      return Configuration().add(
52          Wall({
53              'position': [1, 1, 0],
54              'width': 7,
55              'height': 2.6,
56              'orientation': 45
57          }))
58
```

VI. Création d'une maison

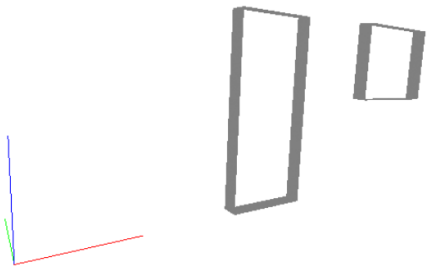
Pour créer une maison, nous avons créé dans le fichier main quatre murs en ajustant leurs positions et leurs orientations afin qu'ils se rejoignent.

Puis nous les avons ajoutés à la classe house que nous avons ajouté à la classe configuration.



```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({
        'position': [0, 0, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
    wall2 = Wall({
        'position': [0.2, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
    wall3 = Wall({
        'position': [0, 7.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 0,
        'edges': True
    })
    wall4 = Wall({
        'position': [7, 0.2, 0],
        'width': 7,
        'height': 2.6,
        'orientation': 90,
        'edges': True
    })
    house = House({'position': [-3, 1, 0], 'orientation': 0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

VII. Création d'ouverture



5 a)

Afin de créer les contours de porte ou fenêtres, nous avons repris les fonctions generate et draw de la classe Section en enlevant les deux grandes faces des murs.

5 b)

La fonction renvoie True pour les deux premières ouvertures et False pour la dernière car la position en hauteur plus la hauteur d'ouverture dépasse la hauteur de la section.

5 c)

Nous avons ensuite divisé la section en sous sections. Pour cela on vérifie que l'ouverture peut bien être créé, et on crée la nouvelle section aux dimensions souhaité et à l'emplacement souhaité en vérifiant qu'une des dimensions de la section n'est pas nulle, cas dans lequel la sous-section ne sera pas créé. On ajoute ensuite ces sections dans une liste avec la fonction append() pour retourner la liste des sous sections voulu.

5 d)

La méthode enumerate permet de parcourir la liste passé en paramètre tout en gardant le compte sur le nombre d'itérations effectué. Elle renvoie ensuite une liste de deux éléments, le premiers étant le numéro de l'itération (l'indexation) et le deuxième étant l'objet dans la liste passé en paramètre que la boucle parcours.

Section[0] vaut 1 et section[1] vaut <Section.section object at 0x7f2f05c06f70>, soit l'objet section.

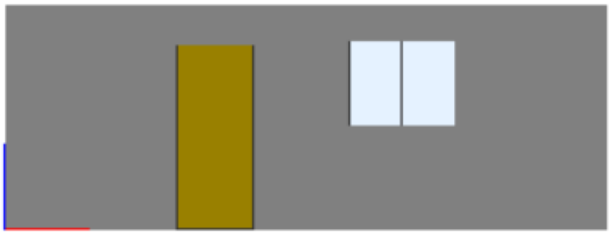
6 a)

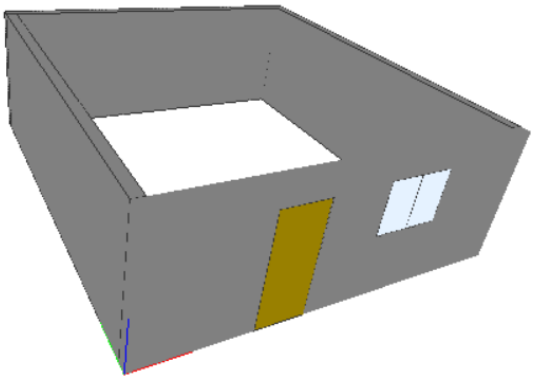
La classe Door crée une ouverture dans un mur de la même manière que la classe Opening puis ajoute une porte dans cette ouverture comme on ajoute les murs et celle-ci est simplement plus fine et de couleure differente.

La classe Windows fait la même chose mais divise la section en « wings » qui sont les ailes de la fenêtre. Une fenêtre est donc une association de deux sections d'epaisseur fine et de couleure différente des murs.

Il suffit alors de créer notre mur, la porte et la fenetre, d'ajouter avec la fonction add(), la porte et la fenêtre dans le mur, puis d'ajouter le mur, la porte et la fenetre à la configuration pour afficher le tout. Il est aussi possible d'ajouter 3 autres murs comme à la question 4 pour obtenir une maison complète.

```
214 def Q6():
215     wall1 = Wall({
216         'position': [0, 0, 0],
217         'width': 7,
218         'height': 2.6,
219         'orientation': 0,
220         'edges': True
221     })
222
223     porte = Door({'position': [2, 0, 0]})
224
225     fenetre = Window({'position': [4, 0, 1.2], 'width': 1.25,
        'height': 1})
226
227     wall1.add(porte).add(fenetre)
228     configuration = Configuration()
229     configuration.add(wall1).add(porte).add(fenetre)
230     return configuration
```





Il est également possible d'ajouter des murs, portes et fenêtres pour afficher une seconde maison, collé à celle-ci et enfin afficher deux maisons côte à côte comme le GIF en début de sujet de TP :