VERMOT DESROCHES Leo
LECORVAISIER Elias
08/12/2021

Rapport de TP3 – Représentation visuelle d'objets

I-Introduction

Le but final de ce troisième TP est la représentation d'objet 3D à l'écran à l'aide d'un programme Python. On va progressivement construire les objets simples composants une maison tel que des murs ou des fenêtres. Nous allons utiliser deux nouvelles librairies : PyGame et PyOpenGL, qui serviront respectivement à utiliser des touches et se déplacer, et à visualiser les scènes en trois dimensions.

II- Préparation

Utilisation de Pygame

1- On commence par importer la librairie Pygame, puis on importe tous ses modules. Ensuite, nous définissons la taille de la nouvelle fenêtre Spyder et enfin nous la fermons.

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

Après avoir lancé le programme, une page Spyder se lance et se ferme rapidement.

2- Une nouvelle page Spyder s'ouvre et lorsque l'on appuie sur un bouton du clavier elle se ferme.

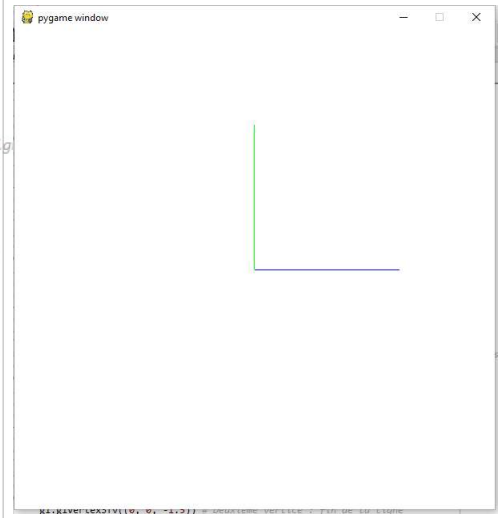
On ouvre une nouvelle fenêtre de la même façon que précédemment, puis on crée une boucle : tant que l'on n'a pas appuyé sur une touche (KEYDOWN) la fenêtre reste ouverte.

```
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()
```

Utilisation de PyOpenGL pour représenter des objets 3D

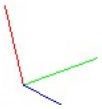
2-

```python
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)


    glu.gluPerspective(45,1,0.1,50)
    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lig
    gl.glColor3fv([0, 0, 250]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0.5, 0, -2)) # Deuxième vertice : fin de la ligne


    gl.glColor3fv([0, 250, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0.5, -2)) # Deuxième vertice : fin de la ligne


    gl.glColor3fv([250, 0, 0]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0, 0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0, -1.5)) # Deuxième vertice : fin de la ligne
    gl.glEnd() # Find du tracé
    pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique


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



3-

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



Découverte de l'environnement du travail du TP

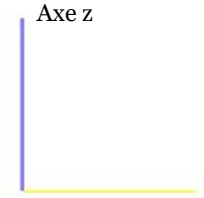
1)

a. Le fichier Configuration permet de mettre en place les axes et leurs paramètres, tel que leur couleur, leur position. De plus, il permet de d'ouvrir la nouvelle page affichant ces axes. Enfin, il permet de contrôler notre nouvelle affichage en attribuant des fonctions à certaines touches.
Le fichier Main permet d'utiliser les autres fichiers pour l'affichage.
Avec la touche « a », on peut faire disparaitre et réapparaitre l'image, avec la touche « z », on peut faire tourner les axes selon le sens anti-horaire et avec le « Z », les axes tournent dans le sens horaire.

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

Cette modification permet de changer la position de l'écran et de modifier la couleur de l'axe x.

c. Nous avons rajouté cette ligne de code dans la fonction initializeTransformationMatrix() : `gl.glRotatef(-90,1,0,0)`



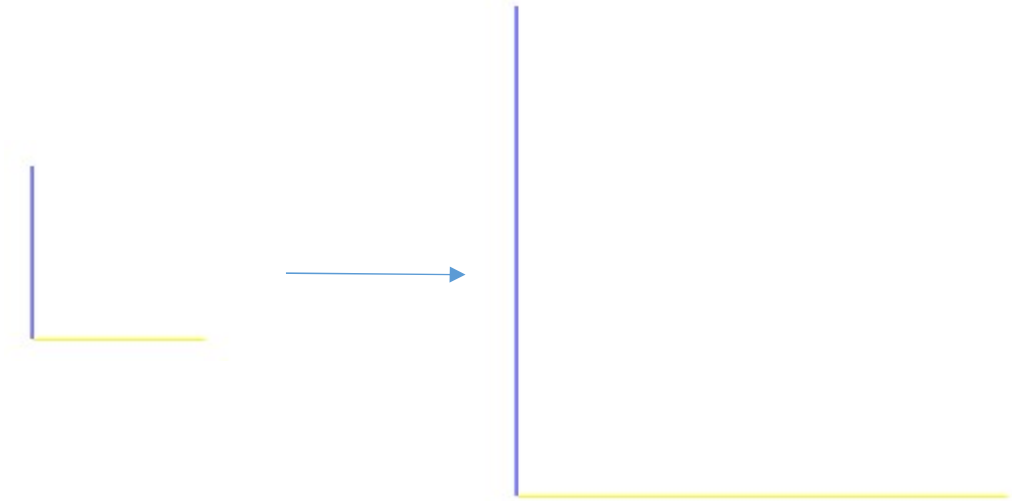
III- Mise en place des interactions avec l'utilisateur avec Pygame

d.

```
elif self.event.dict['unicode'] == 'l' or self.event.key == pygame.K_l:
    gl.glScalef(1.1,1.1,1.1)


elif self.event.dict['unicode'] == 'm' or self.event.key == pygame.K_m:
    gl.glScalef(1/1.1,1/1.1,1/1.1)
```

e.

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

    elif self.event.button == 5:
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```



f.

```
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0] == 1 :
      gl.glRotate(2.5, 0, 0, 1)
      gl.glRotate(2.5, 1, 0, 0)
    elif pygame.mouse.get_pressed()[2] == 1 :
      gl.glTranslate(self.event.rel[0]/100, 0 ,0)
      gl.glTranslate(0, 0, -self.event.rel[1]/100)
```

IV - Création d'une section

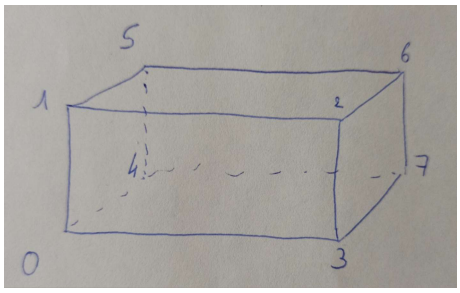
2)

a. Dans cette question, on cherche à définir les paramètres d'un bloc en écrivant les coordonnées des sommets et des faces. Chaque sommet est défini par trois coordonnées :

- widht → qui correspond à la largeur en x
- thickness → qui correspond à la profondeur en y
- height → qui correspond à la hauteur en z

Une fois les sommets crées, on peut définir les faces grâce à ces derniers.

```
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],
                [0, self.parameters['thickness'], self.parameters['height']],
                [self.parameters['width'], self.parameters['thickness'], self.parameters
                ['height']],
                [self.parameters['width'], self.parameters['thickness'], 0],
                ]
    self.faces = [
                [0, 3, 2, 1],
                [0, 1, 5, 4],
                [0, 4, 7, 3],
                [3, 2, 6, 7],
                [4, 7, 6, 5],
                [1, 5, 6, 2],
                ]
```



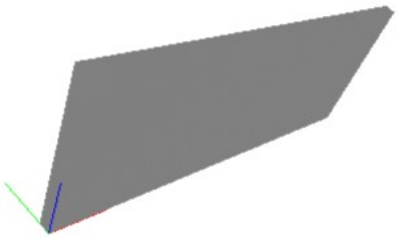
b.

```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
            Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
            )
```

L'instruction Configuration().add(Section).display() dessine les axes et les objets.

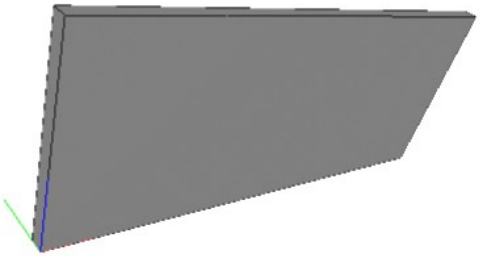
On écrit ensuite le code permettant de dessiner le bloc, pour cela on crée une boucle permettant de colorier en gris l'espace entre les angles définis au préalable.

```
# Draws the faces
def draw(self):
  if self.parameters['edges'] == True :
    self.drawEdges()
  gl.glPushMatrix()
  gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters
  ['position'][2])
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
  gl.glBegin(gl.GL_QUADS)
  gl.glColor3fv([0.5, 0.5, 0.5])

  for i in self.faces :
    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()
```



c.

```
# Draws the edges
def drawEdges(self):
  gl.glPushMatrix()
  gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters
  ['position'][2])
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)
  gl.glBegin(gl.GL_QUADS)
  gl.glColor3fv([0.1, 0.1, 0.1])

  for i in self.faces :
    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()
```



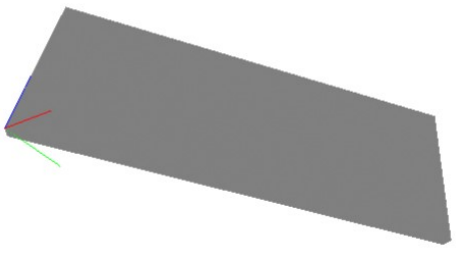
```
# Draws the faces
def draw(self):
  if self.parameters['edges'] == True :
    self.drawEdges()
  gl.glPushMatrix()
  gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters
  ['position'][2])
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
  gl.glBegin(gl.GL_QUADS)
  gl.glColor3fv([0.5, 0.5, 0.5])

  for i in self.faces :
    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()
```

V - Création des murs

3)

a.

```
# Draws the faces
def draw(self):
  gl.glPushMatrix()
  gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
  for x in self.objects :
    x.draw()
  gl.glPopMatrix()
```



```
def Q3a():
    return Configuration().add(Wall({'position' : [1, 1, 0], 'width':7, 'height':2.6,
    'orientation' : 45}))
```

VI - Création d'une maison

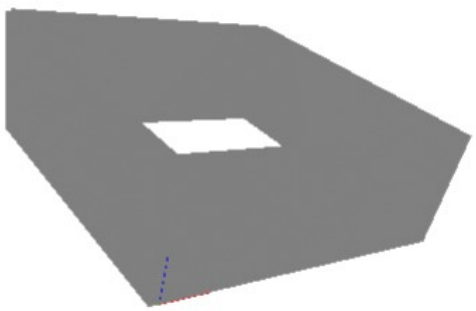
4)

a.

```
def draw(self):
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)

    for x in self.objects :
        x.draw()
    gl.glPopMatrix()
```

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

Resultat :



Conclusion

Bien que nous n'ayons pas pu finir le TP, nous avons pu commencer la construction et l'affichage d'objets tel que des murs sur l'outil de visualisation. Nous avons donc appris à réaliser des objets 3D sur le langage Python. Nous avons eu quelques difficultés, notamment pour comprendre l'utilisation de la bibliothèque gl et de certaines de ses fonctions.