LEBRUN Mathilde

HAVELANGE Victor TD1

TP N°3 du 06/12/21

TP3 : Représentation visuelle d'objets

Introduction :

Grâce à différentes fonctions, nous allons apprendre à créer des objets en 3D. On commence par créer des formes très simples en 2D pour ensuite les représenter en 3D.

Ensuite, on utilise pygame er PyOpenGl pour représenter un mur en 3D.

II- Utilisation de Pygame

1) Après exécution de ce code, on peut observer un écran blanc qui apparait et disparaît très rapidement.

En effet, pygame.init permet d'afficher l'écran blanc et pygame.quit permet de fermer la fenêtre.

2) la fenêtre s'ouvre mais elle se ferme lorsqu'on clique n'importe où sur le clavier.

Explication : le terme 'keydown' correspond à une touche de clavier pressée. Soit le code signifie que si n'importe quelle touche du clavier est pressée alors 'pygame.quit' s'exécute, la fenêtre se ferme.

Utilisation de Pyopengl pour représenter des objets 3D

1) Dans configuration.py, on insère ce code pour gluPerspective :
2) Le même code en ajoutant les fonctions glBegin, glEnd, glColor3fv, glVertex3fv :

```
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 (45.0, (display[0] / display[1]), 0.1, 50.0)
    gl.glTranslatef(0,2,-5)
    gl.glRotatef(-90, 1, 0, 0)
    # Placer ici l'utilisation de gluPerspective.
    gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode l
    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]) # 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

    gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
    gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
    gl.glVertex3fv((0, 0, -1)) # 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()
```

Découverte de l'environnement du travail du TP

1.a) On ajoute la commande suivante à la fonction Q1a() dans le main.py :

```
def Q1a():
    return Configuration()
```

Cette commande va appeler la classe configuration du fichier configuration.py dont toutes les fonctions associées à cette classe.

1.b)

```
def Q1b_f():
    return Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
        setParameter('xAxisColor', [1, 1, 0]). \
        setParameter('yAxisCo lor', [0,1,1]). \
        display()
```

setParameter permet de changer la couleur de l'axe x (rouge → jaune) puis l'axe y (vert → bleu).

Display() permet d'afficher les axes.

1.c) Pour effectuer la position des axes :

```
def initializeTransformationMatrix(self):
    gl.glMatrixMode(gl.GL_PROJECTION)
    gl.glLoadIdentity()
    glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)

    gl.glMatrixMode(gl.GL_MODELVIEW)
    gl.glLoadIdentity()
    gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
    gl.glRotate(90,1,0,0)
```

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

1.d)

```
# 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.
        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)
```

La commande self.event.dict[‘unicode] == Z correspond à la condition touche Z pressée, elle indique que les axes ont une variation d’un angle de 2.5 degrés à chaque rotation. Le raisonnement est identique pour les autres touches.

1.E)

```
# Processes the MOUSEBUTTONDOWN event
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)
```

Le if self.event.button == 4 : correspond à la condition « scroller vers le haut » . Si cette condition est réalisée, la taille de chacun des axes est augmentée par un facteur de 1,1.

Le if self.event.button == 5 correspond à la condition « scroller vers le bas » . Si cette condition est réalisée, la taille de chacun des axes est diminuée par un facteur de 1,1 .

1.F) on s’intéresse ici aux déplacements des objets à l’aide des boutons de la souris.

Ici, si le bouton gauche de la souris est pressé, si nous déplaçons la souris horizontalement, cela fera rotationner les trois axes autour de l’axe x. Si nous la déplaçons verticalement, cela fera rotationner les trois axes autour de l’axe z.

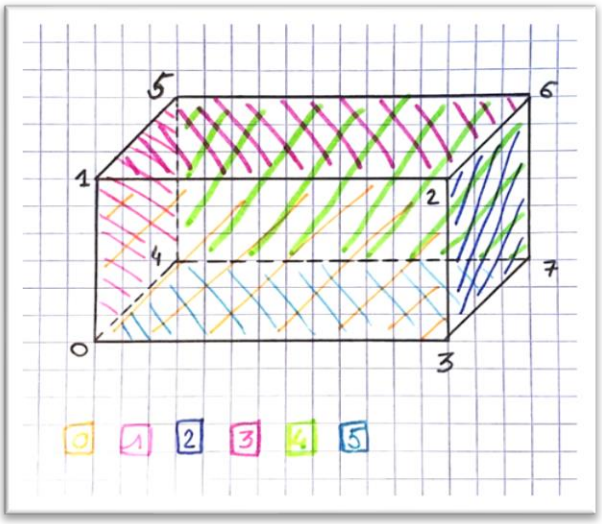
En terme de translation, si nous cliquons sur le bouton droit de la souris, les trois axes sont censés se déplacer vers l'endroit où on bouge la souris. Afin de mieux "contrôler" les trois axes, nous avons dû réduire la sensibilité du déplacement par rapport au mouvement de la souris sinon l’ensemble bougeait trop vite. Cela s’est fait en divisant par 20 la commande self.event.rel

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0] == 1:
        gl.glRotate(self.event.rel[0],1,0,0)
        gl.glRotate(self.event.rel[1],0,0,1)
    elif pygame.mouse.get_pressed()[2] == 1:
        gl.glTranslate((self.event.rel[0]/20),0,0)
        gl.glTranslate(0,0,self.event.rel[1]/20)
```

2.A) A partir de l'ébauche et du schéma que nous avons fait ci-dessous, on détermine sommets et faces dans la fonction 'generate'.

On peut voir les différents noms des sommets et les faces que nous avons listé dans def generate.

Plus précisément, les sommets dans self.vertices et les faces dans self.face.



Nous avons mis des noms à chaque sommet et faces pour mieux se repérer dans notre code.

Voici notre code :

```
# Defines the vertices and faces
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, 4, 5, 1],
            [2, 3, 7, 6],
            [1, 2, 5, 6],
            [4, 5, 6, 7],
            [0, 4, 7, 3]
            ]
```

2.B)

Afin de pouvoir représenter un parallélépipède avec une couleur grise sur chaque surface, il est nécessaire d'utiliser plusieurs méthodes de la librairie OpenGL.GL comme

```
gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL,
gl.glBegin(gl.GL_QUADS),  gl.glColor3fv([0.5, 0.5, 0.5]) #
```

qui permettent de tracer les différentes faces du sommet. Ensuite, on a la méthode glPopMatrix une fois que l'on a fini qui permet de représenter le parallélépipède.

```
def draw(self):
    if self.parameters['edges']:
        self.drawEdges
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for x in self.faces:
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
        gl.glVertex3fv(self.vertices[x[0]])
        gl.glVertex3fv(self.vertices[x[1]])
        gl.glVertex3fv(self.vertices[x[2]])
        gl.glVertex3fv(self.vertices[x[3]])
        gl.glEnd()
    gl.glPopMatrix()
```

2.C)

La méthode qui permet de tracer les différentes arêtes du parallélépipède dans une certaine couleur est assez similaire à la méthode précédente. On utilise juste la méthode `gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)` à la place de `gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL` car on souhaite donner une couleur à une ligne plutôt qu'un remplissage. On donne une autre couleur avec la méthode `l.glColor3fv([1, 1, 1])`.

```
def drawEdges(self):
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
    for x in self.faces:
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv([1, 1, 1]) # Couleur noire
        gl.glVertex3fv(self.vertices[x[0]])
        gl.glVertex3fv(self.vertices[x[1]])
        gl.glVertex3fv(self.vertices[x[2]])
        gl.glVertex3fv(self.vertices[x[3]])
        gl.glEnd()
    gl.glPopMatrix()
```

+ quelques lignes dans DRAW :

```
if self.parameters['edges']:
    self.drawEdges
```

3.A) dans WALL :

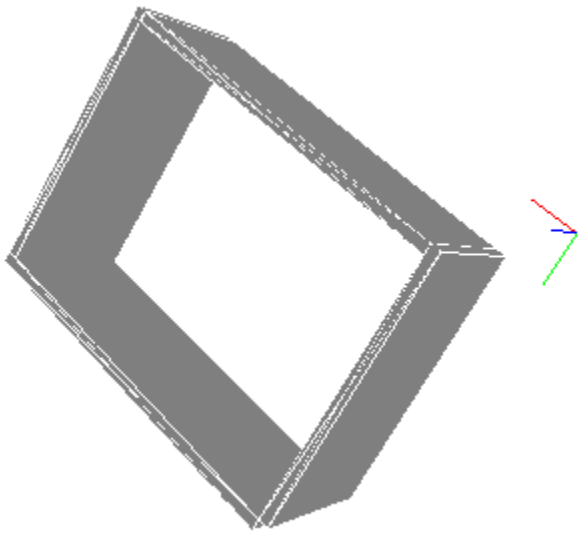
```
# Draws the faces
def draw(self):
   for i in self.objects:
       i.draw()
       i.drawEdges()
```

On l'appelle dans MAIN :

```
def Q3a():
    wall = Wall({'position': [1,0,0], 'width':7, 'height':2.6, 'edges': True})
    Configuration().add(wall).display()
```

4.A)

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    house=House()
    wall1 = Wall({'position': [1,1,0], 'width':7, 'height' : 2, 'edges' : True})
    wall2 = Wall({'position': [1,6,0], 'width':7, 'height' : 2, 'edges' : True})
    wall3 = Wall({'position': [1,1,0], 'width':5, 'height' : 2, 'edges' : True, 'orientation' : 90})
    wall4 = Wall({'position': [8,1,0], 'width':5, 'height' : 2, 'edges' : True, 'orientation' : 90})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house).display()
```



5.A) ouverture

```
    def Q5a():
        # Ecriture avec mélange de variable et de chaînage
        opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
        opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
        return Configuration().add(opening1).add(opening2)

    def Q5b():
        # Ecriture avec mélange de variable et de chaînage
        section = Section({'width':7, 'height':2.6})
        opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
        opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
        opening3 = Opening({'position': [4, 0, 1.7], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})

        print(section.canCreateOpening(opening1))
        print(section.canCreateOpening(opening2))
        print(section.canCreateOpening(opening3))
        return Configuration()

def Q5c1():
    section = Section({'width':7, 'height':2.6})
    opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createOpening(opening1)
    configuration = Configuration()
    for x in sections:
        configuration.add(x)
    return configuration

def Q5c2():
    section = Section({'width':7, 'height':2.6})
    opening2 = Opening({'position': [4, 0, 1.2], 'width':1.25, 'height':1, 'thickness':0.2, 'color': [0.7, 0.7, 0.7]})
    sections = section.createNewSections(opening2)
    configuration = Configuration()
    for section in sections:
        configuration.add(section)
    return configuration
```