PEYROUX Alexis
BEAURAIN Hugo
15/12/2021

Rapport de TP3 – Représentation visuelle d'objet

I. Introduction

1. Question (1).

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

Ces quatre lignes de programme permettent d'importer le module pygame et de modéliser une fenêtre de taille 300 par 200.

2. Question (2).

```
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()
```

Ce programme permet d'ouvrir une fenêtre. Tant que l'on appuie sur aucune touche du clavier, la fenêtre « écran » reste ouvre. Lorsqu'on appuie sur la croix en haut à droite ou n'importe quel autre touche, la fenêtre se ferme. Cela permet de la garder à l'écran pour faire une rotation ou une translation pour mieux voir la modélisation.

II. Préparation

1. Question (1).

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
```

Cette ligne nous permet de gérer la perspective de la fenêtre de modélisation. Les différents réglages sont fait comme indiqué : « On utilisera un Field of View de 45° Et comme plan de clipping near la valeur 0.1 Et le plan de clipping far, la valeur 50. »

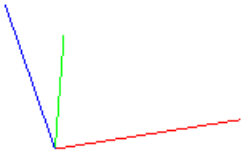
2. Question (2).

```
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 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.glEnd() # Find du tracé

gl.glBegin(gl.GL_LINES)
gl.glColor3fv([0, 255, 0])
gl.glVertex3fv((0,0, -2))
gl.glVertex3fv((0, 1, -2))
gl.glEnd()
```

```
gl.glBegin(gl.GL_LINES)
gl.glColor3fv([0, 0, 255])
gl.glVertex3fv((0,0, -2))
gl.glVertex3fv((0, 0, -1))
gl.glEnd()
pygame.display.flip()
```

Le but de ce code est d'afficher les axes sur la fenêtre. Pour chaque axe on indique que l'on va faire des lignes, puis on indique la couleur à l'aide de son code RGB. Ensuite on indique le départ de la ligne et la fin de la ligne pour afficher le tracé de l'axe. Toutes les coordonnées sont données par rapport à l'origine de la fenêtre.

3. Question (3).



```
gl.glTranslatef(0.0, 2, -5)
gl.glRotatef(-45, 2, 1, 0)
```

La commande glTransaltef(0.0,2,-5) permet de faire une translation de la position à l'écran de 0 sur l'axe x, 2 sur y et -5 sur z.
gl.glRotatef(-45, 2, 1, 0) permet une rotation de 45° par rapport à l'axe x et y.

4. Question (1).

A. Dans Configuration.py, on retrouve le tracé des différents axes. On retrouve aussi les fonctions qui sont attribuées au clavier.

Dans le fichier Configuration on retrouve deux instructions permettant d'initialiser pygame et OpenGL. A la fin du programme nous pouvons remarquer des instructions correspondant à la gestion du clic droit et gauche ainsi que la molette de la souris. D'autres instructions permettent de gérer la croix en haut à droite et les touches au clavier.

B. Le chainage de setter et de display ne pose de problème car display va permettre d'afficher le résultat des setter. Les opération se faisant dans l'ordre et non en même temps cela ne pose pas problème.

Dans le setter de screenposition il est précisé que si la position n'est pas donnée on lui donnera par défaut la position -10 pour éviter des problèmes d'affichage.
On utilise la chaîne de setter pour changer les paramètres du trait. On change ainsi sa couleur.

C. gl.glRotatef(-90, 1, 0, 0), cette instruction nous permet la rotation des axes de 90° autour de l'axe x :

Avant la rotation

Après la rotation

III. Mise en place des interactions avec l'utilisateur

1. Question (1).

D.

```
elif self.event.dict['unicode'] == 'p' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_p):
    gl.glScalef(1.1, 1.1, 1.1)

elif self.event.dict['unicode'] == 'm' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_m):
    gl.glScalef(0.9,0.9,0.9)
```

Ici le but était de gérer le zoom grâce aux touches p (pour plus) et m (pour moins). Nous n'avons pas pu le configurer sur les touches page up et page down à cause d'un problème de clavier.
Pour zoomer en avant il faut alors détecter l'appui sur la touche p ou P, car la méthode permet de gérer les majuscules. Une fois la pression sur la touche détecté on fait alors un zoom de 1.1 sur chaque axe afin que le zoom soit uniforme. Par analogie il suffit de presser sur la touche m pour zoomer en arrière de 0.9 sur chaque axe.

E.

```
if self.event.type == pygame.MOUSEBUTTONDOWN and self.event.button == 4:
    gl.glScalef(1.1, 1.1, 1.1)
elif self.event.type == pygame.MOUSEBUTTONDOWN and self.event.button == 5:
    gl.glScalef(0.9, 0.9, 0.9)
```

On adopte le même raisonnement pour la molette de la souris afin de zoomer en avant et en arrière.

F.

```
if pygame.mouse.get_pressed()[0] == 1:
    x= self.event.rel[0]
    y= self.event.rel[1]
    gl.glRotate(x+y, 1, 0, 1)

elif pygame.mouse.get_pressed()[2] == 1:
    x= self.event.rel[0]
    y= self.event.rel[1]
    gl.glTranslate(x*0.05, 0, y*0.05)
```

Il faut maintenant gérer les clics droit et gauche de la souris. Le clic gauche va nous permettre de faire une rotation dans la fenêtre d'affichage. On récupère alors le signal lorsque la touche est pressé ainsi que la valeur du déplacement que l'on met en x et en y. On additionne alors les deux valeurs et on alloue ces valeurs à l'angle de rotation autour de l'axe x et z.

On utilise le même raisonnement pour le clic droit de la souris pour faire une translation selon l'axe y et z. Nous avons juste multiplié les valeurs de déplacement par 0.05 afin de gérer mieux la sensibilité.

IV. Création d'une section

1. Question (2)

A.

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

Pour la création d'une section il nous faut implémenter la fonction generate qui permet de créer les lignes et les faces de notre section avec comme argument la longueur, largeur et la profondeur.

B.

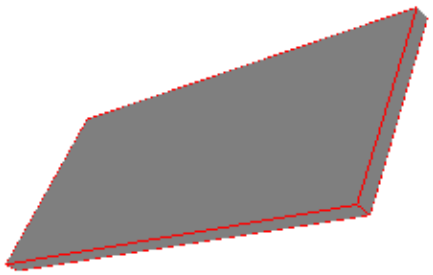
```
return Configuration().add(
        Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
        )
```

Cette fonction permet de créer une section d'origine [1,1,0] (coin inférieur gauche), de longueur 7 et de hauteur 2,6.

C.

```
82
83    # Draws the edges
84    def drawEdges(self):
85        # A compléter en remplaçant pass par votre code
86        gl.glPushMatrix()
87        gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
88        gl.glRotate(self.parameters['orientation'],0,0,1)
89        gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
90        for i in self.faces :
91            gl.glBegin(gl.GL_QUADS)
92            gl.glColor3fv([1,0,0])
93            for j in i:
94                gl.glVertex3fv(self.vertices[j])
95            gl.glEnd()
96        gl.glPopMatrix()
97
98
99    # Draws the faces
100   def draw(self):
101
102       if self.parameters['edges']:
103           self.drawEdges()
104
105       gl.glPushMatrix()
106       gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
107       gl.glRotate(self.parameters['orientation'],0,0,1)
108       gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
109       for i in self.faces :
110           gl.glBegin(gl.GL_QUADS)
111           gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
112           for j in i:
113               gl.glVertex3fv(self.vertices[j])
114           gl.glEnd()
115       gl.glPopMatrix()
116
117
```

Voici le code permettant de dessiner à l'écran une section et ses arrêtes. La ligne de code 87 et 106 permettent de faire une translation de la valeur de la position. La ligne 88 et 107 permette de gérer la rotation autour de l'axe z. Ensuite la ligne 89 permet de dire que nous allons tracer en mode ligne. On parcours ensuite les différentes coordonnées des coins de la face et on trace dans la couleur souhaité (grâce à la ligne 92 et 111) les lignes pour la méthode drawEdges et les faces pour la méthode draw.



V. Création d'un mur

1. Question (3)

A.

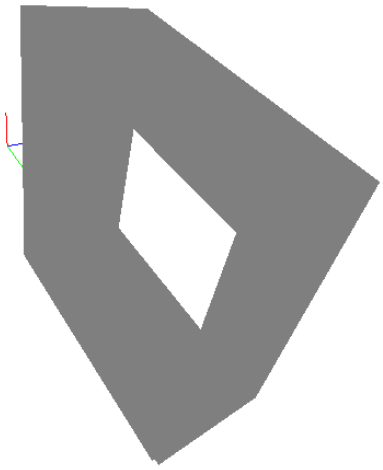
```
def draw(self):
    gl.glPushMatrix()
    gl.glRotate(self.parameters['orientation'],0,0,1)
    for i in self.objects:
       i.draw()
    gl.glPopMatrix()

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

Pour créer un mur dans la section maison il suffit d'appeler la méthode draw écrite précédemment et l'utiliser en parcourant objects qui est la liste des faces des murs de la maison.

VI. Création d'une maison

1. Question(4)

On a alors réutiliser à peu de chose près la méthode draw de la section précédente afin de créer les murs de la maison. La principale difficulté était d'afficher plusieurs sections en même temps. Une fois ce problème résolu nous avons pu afficher 4 murs. Nous avons fait une rotation de 90° pour les murs selon dont la longueur est selon y.



VII. Création d'ouvertures

Dans cette partie nous devons créer des ouvertures nous permettant de placer les fenêtres et les portes de la maison. Le but est de dissocier une face en plusieurs petites section afin d'encadrer l'ouverture.

Code dans opening :

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

# Draws the faces
def draw(self):
    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # TracÃ© dâ€™un quadrilatÃ¨re
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for j in i:
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

Dans le fichier Opening.py nous avons écrit la fonction generate et draw comme dans la fonction section car les ouvertures sont représentable de la même façon que les sections

Code dans section :

```
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    return (self.parameters['height']>=x.getParameter('height') + x.getParameter('position')[2] - self.parameters['position'][2]
            and x.getParameter('position')[2]>= self.parameters['position'][2]
            and self.parameters['width']>= x.getParameter('width') + x.getParameter('position')[0] - self.parameters['position'][0]
            and x.getParameter('position')[0] >= self.parameters['position'][0])

# Creates the new sections for the object x
def createNewSections(self, x):
    if self.canCreateOpening(x):
        section = [Section(copy.copy(self.parameters))
            .setParameter('width', x.getParameter('position')[0] - self.getParameter('position')[0]),
            Section(copy.copy(self.parameters))
            .setParameter('height', self.getParameter('height') - x.getParameter('height') - x.getParameter('position')[2] + self.getParameter('position')[2])
            .setParameter('width', x.getParameter('width'))
            .setParameter('position',[x.getParameter('position')[0], self.getParameter('position')[1],x.getParameter('position')[2] + x.getParameter('height')
            ]),
            Section(copy.copy(self.parameters))
            .setParameter('height', x.getParameter('position')[2] -self.getParameter('position')[2])
            .setParameter('width', x.getParameter('width'))
            .setParameter('position', [x.getParameter('position')[0], self.getParameter('position')[1], self.getParameter('position')[2]
            ]),
            Section(copy.copy(self.parameters))
            .setParameter('width', self.getParameter('width') -x.getParameter('width') -x.getParameter('position')[0] +self.getParameter('position')[0])
            .setParameter('position', [x.getParameter('position')[0] + x.getParameter('width'),self.getParameter('position')[1], self.getParameter('position')[2]
            ])
        ]
        res=[]
        for i in section:
            print(i.getParameter('width'))
            print(i.getParameter('height'))
            print(i.getParameter('position'))
            if (i.getParameter('width')!= 0.0 and i.getParameter('height')!=0.0):
                i.generate()
                res.append(i)
            return res
```

Nous avons d'abord créé ici la méthode canCreateOpening afin de venir si l'ouverture était possible. Cette méthode vérifie que l'ouverture n'est pas plus longue, plus large ou plus haute que la section en elle-même. Si l'ouverture fait 3,5 alors que le mur n'en fait que 3 cela n'a pas de réel intérêt.

Ensuite nous avons donc écris la méthode createNewSection qui permet de partitionner la section initial afin d'y créer une ouverture par la suite.

VIII. Conclusion

Pour conclure ce tp nous a permis de nous familiarisé avec la programmation d'un modèle 3D en python. Ce tp était intéressant car lors de nos années précédents l'aspect graphique n'a jamais été abordé en programmation. Pourtant cet aspect se révèle très utile dans l'industrie, nous pouvons penser notamment à l'impression 3D qui pourrait en découler. Malheureusement nous n'avons pas eu le temps de finir ce tp, ce qui est dommage car nous aurions aimé rendre une maison avec de belles fenêtres et portes.