OUKZIZ Salma
IBNABDELJALIL Adam
Date du TP (24/11/2021)

Rapport de TP3 – Représentation visuelle d'objets

Table des matières

Introduction 2
Préparation à faire avant le TP 2
I. Utilisation de pygame 2
1. Question (1) 2
2. Question (2) 2
II. Utilisation de pyOpenGL pour représenter des objets 3D 2
1. Essai d'un code 2
2. Tracer les axes 3
3. Utilisation d'une transformation de translation et rotation 3
III. Découverte de l'environnement du travail du TP 4
1. Question 1.a 4
2. Question 1.b 4
3. Question 1.c 4
Travail à faire pendant le TP 5
IV. Mise en place des interactions avec l'utilisateur avec Pygame 5
1. Question 1.d 5
2. Question 1.e 5
3. Question 1.f 5
V. Création d'une section 5
1. Question 2.a 5
2. Question 2.b 6
3. Question 2.c 6
VI. Création des murs : 7
VII. Création d'une maison : 9
VIII. Création d'une ouverture 10
1. Question 5.a 10
CONCLUSION 11

POLYTECH ANNECY-CHAMBÉRY
UNIVERSITÉ SAVOIE MONT BLANC

Introduction

On s'intéresse dans ce TP à la représentation d'objets 3D sur une fenêtre graphique , en utilisant les librairies

Pygame pour gérer l'affichage de la fenêtre et Py OpenGL pour l'affichage des scènes tridimensionnelles

Préparation à faire avant le TP

I. Utilisation de pygame

1. Question (1)

Lorsque on exécute ce programme on voit une fenêtre de 300*200 qui se ferme sur le champs.

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

Dans ce programme on a importé le module pygame, ensuite on la initialisé , après on a créé une fenêtre pygame de dimension 300x 200 et enfin la dernière ligne est faite pour arrêter l'utilisation du module pygame.

2. Question (2)

Lorsqu'on exécute ce code on a une fenêtre pygame qui se ferme si on appuie sur une touche du clavier.

```
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()
```

Dans ce programme on a créé une fenêtre pygame de taille 300*200 comme dans le programme précèdent. Après on a utilisé la variable booléenne initialisé à true qui nous permet de quitter pygame lorsqu'elle devient false. Ensuite on passe à la boucle while qui lorsqu'on touche un bouton du clavier la fenêtre se ferme

II. Utilisation de pyOpenGL pour représenter des objets 3D

1. Essai d'un code

Lorsqu'on exécute ce code, on obtient une fenêtre pygame, cependant on ne peut fermer cette fenêtre qu'en cliquant sur la croix.

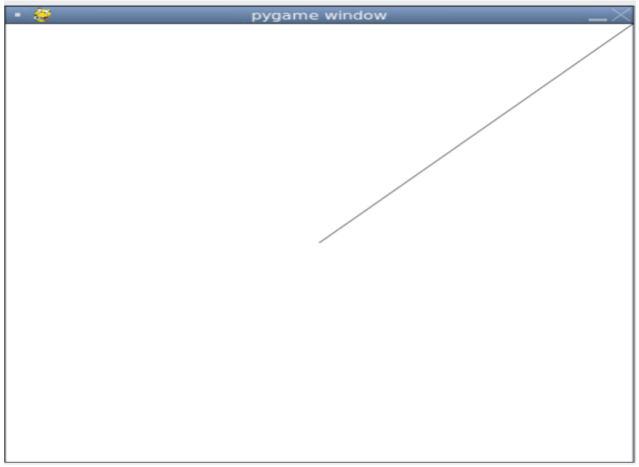
La fonction gluPerspective permet d'initialiser une matrice de perspective avec des paramètre donnés dans l'énoncé : fovy = 45, aspect = display [0] / display[1], zNear = 0.1 ,zFar = 50

2. Tracer les axes

On a copié le code suivant :

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

On a obtenu le tracé suivant

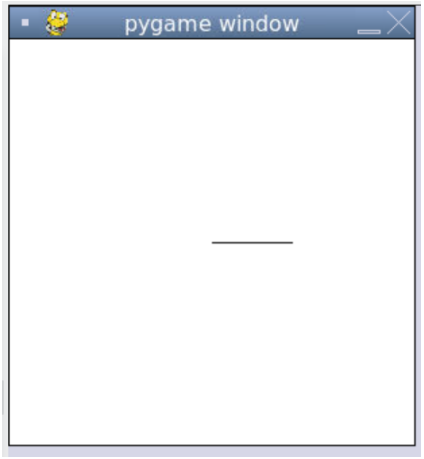


3. Utilisation d'une transformation de translation et rotation

On rajoute les trois lignes suivante avant la fonction glBegin() .

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

On obtient alors cela :



III. Découverte de l'environnement du travail du TP

1. Question 1.a

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

On retourne un objet de la classe configuration avec les valeurs par défaut qui correspond à trois axe unitaire formant un repère orthonormé.

2. Question 1.b

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

Le chaînage de l'appel des méthodes setParameter() et display() est possible car elles appartiennent à la classe configurations.

Un traitement particulier est effectué dans le « setter » pour le paramètre screenPosition. Ce traitement est nécessaire car il est utilisé pour initialiser la positon de l'objet on fait donc appel à la méthode initializeTransformationMatrix() afin de le mettre à jour.

3. Question 1.c

Pour que l'axe z soit représenté verticalement sur l'écran et que l'axe x soit représenté horizontalement on effectue une rotation de -90° autour du vecteur x.

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

Travail à faire pendant le TP

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

1. Question 1.d

Dans la méthode **processKeyDownEvent()** on ajoute les instructions suivantes :

```
if self.event.key == pygame.K_PAGEUP:
    gl.glScalef(1.1, 1.1, 1.1)
elif self.event.key == pygame.K_PAGEDOWN:
    gl.glScalef(1 / 1.1, 1 / 1.1, 1 / 1.1)
```

2. Question 1.e

Dans la méthode **processMouseButtonDownEvent()** on ajoute les instructions suivantes :

```
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)
```

3. Question 1.f

Dans la méthode **processMouseMotionEvent ()** on ajoute les instructions suivantes :

```
if pygame.mouse.get_pressed()[0]==1:
  gl.glRotatef( self.event.rel[1], 1, 0, 0)
  gl.glRotatef( self.event.rel[0], 0, 0, 1)
if pygame.mouse.get_pressed()[2]==1:
  gl.glTranslatef(self.event.rel[0]/100,0,0)
  gl.glTranslatef(0, 0, -self.event.rel[1]/100)
```

V. Création d'une section

1. Question 2.a

Cette fonction nous permet de définir les faces et les sommets .

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

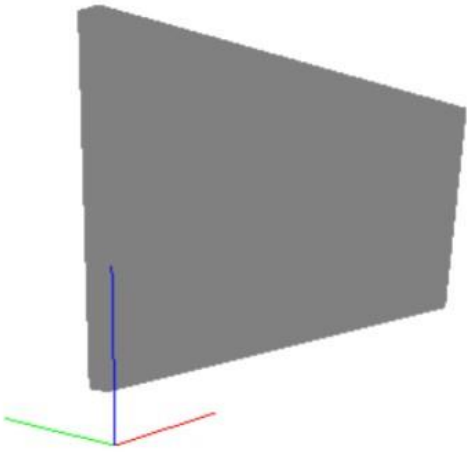
Afin de mieux comprendre cette question on a dessiné un parallélépipède puis on a numéroté les sommets, puis on a regroupé les 4 sommets qui forment chaque face.

2. Question 2.b

La fonction « draw » permet de faire l'assemblage des faces afin de construire une section.

```
def draw(self):
  if self.parameters['edges']==True:
    self.drawEdges
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
  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'],1,1,1)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    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()
```

On a obtenu cette section :



3. Question 2.c

Grace à la fonction « drawEdges » on peut visualiser les arêtes de la section.

```
    # Draws the edges
    def drawEdges(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'],1,1,1)
        gl.glBegin(gl.GL_LINES) # Tracé d'une ligne
        gl.glColor3fv([0, 0, 0]) # Couleur noire

        gl.glVertex3fv(self.vertices[0])
        gl.glVertex3fv(self.vertices[3])

        gl.glVertex3fv(self.vertices[3])
        gl.glVertex3fv(self.vertices[2])

        gl.glVertex3fv(self.vertices[2])
        gl.glVertex3fv(self.vertices[1])

        gl.glVertex3fv(self.vertices[1])
        gl.glVertex3fv(self.vertices[0])
```

```
        gl.glVertex3fv(self.vertices[0])
        gl.glVertex3fv(self.vertices[4])

        gl.glVertex3fv(self.vertices[4])
        gl.glVertex3fv(self.vertices[5])

        gl.glVertex3fv(self.vertices[5])
        gl.glVertex3fv(self.vertices[1])

        gl.glVertex3fv(self.vertices[5])
        gl.glVertex3fv(self.vertices[6])

        gl.glVertex3fv(self.vertices[6])
        gl.glVertex3fv(self.vertices[7])

        gl.glVertex3fv(self.vertices[7])
        gl.glVertex3fv(self.vertices[3])

        gl.glVertex3fv(self.vertices[7])
        gl.glVertex3fv(self.vertices[4])

        gl.glEnd()
        gl.glPopMatrix()
```

Après exécution on a obtenu :

VI. Création des murs :

la classe Wall permet de générer des murs à partir de section. Dans le constructeur de la classe, des paramètres sont indiqués. Ensuite, un objet Section est créé puis cette section est ajouté aux objets à afficher.

```
class Wall:
    # Constructor
    def __init__(self, parameters = {}) :
        # Parameters
        # position: position of the wall
        # width: width of the wall - mandatory
        # height: height of the wall - mandatory
        # thickness: thickness of the wall
        # color: color of the wall

        # Sets the parameters
        self.parameters = parameters

        # Sets the default parameters
        if 'position' not in self.parameters:
            self.parameters['position'] = [0, 0, 0]
        if 'width' not in self.parameters:
            raise Exception('Parameter "width" required.')
        if 'height' not in self.parameters:
            raise Exception('Parameter "height" required.')
        if 'orientation' not in self.parameters:
            self.parameters['orientation'] = 0
        if 'thickness' not in self.parameters:
            self.parameters['thickness'] = 0.2
        if 'color' not in self.parameters:
            self.parameters['color'] = [0.5, 0.5, 0.5]

        # Objects list
        self.objects = []

        # Adds a Section for this object
        self.parentSection = Section({'width':
self.parameters['width'], \
                                      'height':
self.parameters['height'], \
                                      'thickness':
self.parameters['thickness'], \
                                      'color':
self.parameters['color'],
                                      'position':
self.parameters['position']})
                                          'position':
 self.parameters['position']})
        self.objects.append(self.parentSection)

```

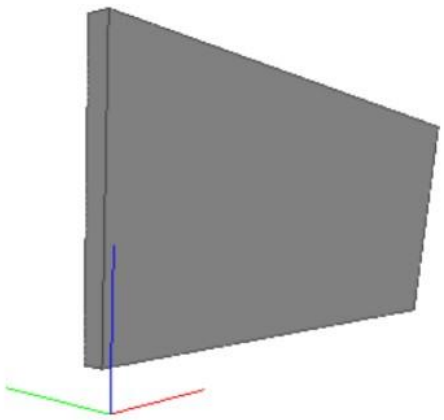
Grace à la méthode draw de la classe Configuration, on a réalisé une suite d'instruction pour tracer les murs, les arêtes et les objets .

```
    def draw(self):
        # A compléter en remplaçant pass par votre code
      gl.glPushMatrix()
      gl.glRotatef(self.parameters['orientation'],0,0,1)
      self.parentSection.drawEdges()
      for x in self.objects:
        x.draw()
      gl.glPopMatrix()
```

La fonction Q3() du fichier Main.py :

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

Voici ce que donne après execution :



VII. Création d'une maison :

La methode draw permet de dessiner une maison en parcourant les objets qui la constituent :

```
70    def draw(self):
71        # A compléter en remplaçant pass par votre code
72        gl.glPushMatrix()
73        gl.glRotatef(self.parameters['orientation'],0,0,1)
74        self.parentSection.drawEdges()
75        for x in self.objects:
76            x.draw()
77        gl.glPopMatrix()
78
```

Pour créer une maison de 4 murs, nous avons defini la longueur, la hauteur et l'orientation de ces murs.

```
42  def Q4a():
43      # Ecriture en utilisant des variables : A compléter
44      wall1 = Wall({'position': [0, 0, 0], 'width':5, 'height':3 ,
    'edges': True , 'orientation':0})
45      wall2 = Wall({'position': [0, 0, 0], 'width':4, 'height':3 ,
    'edges': True , 'orientation':90})
46      wall3 = Wall({'position': [0, 3.8 , 0], 'width':5, 'height':3,
    'edges': True , 'orientation':0})
47      wall4 = Wall({'position': [-4, 5,0], 'width':4, 'height':3,
    'edges': True , 'orientation':-90})
48      house = House({'position': [1, 4, 4], 'orientation':33})
49      house.add(wall1).add(wall3).add(wall4).add(wall2)
50      return Configuration().add(house)
```

Le résultat après execution :

VIII. Création d'une ouverture

1. Question 5.a

Il s'agit de créer un mur mais en ne remplissant pas 2 face devant et derrière.
J'ai rajouté une fonction fusion ce qui permet de prendre directement en compte la position de l'ouverture.

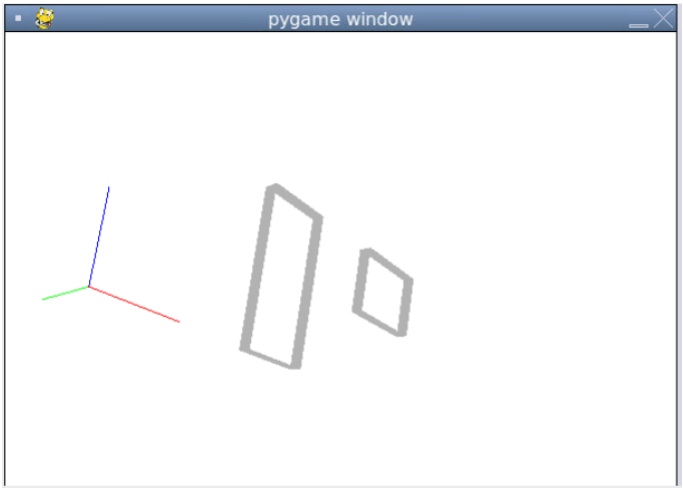
Voici le code et le résultat :

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

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

# Draws the faces
def draw(self):
  gl.glPushMatrix()
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
   #on trace les faces : GL_FILL
  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()
```

```
def fusion(list1,list2):
  #deux coordonné entre elle et en donne le résultat
  list=[]
  list.append(list1[0]+list2[0])
  list.append(list1[1]+list2[1])
  list.append(list1[2]+list2[2])
  return list
```



CONCLUSION

Nous n'avons pas pu terminer le TP mais nous avons quand même une esquisse de maison avec 4 murs. Ce TP a été très enrichissant pour nous. En effet nous connaissons maintenant les bibliothèques OpenGL et PyGame et nous avons une meilleure compréhension de la construction d'un programme informatique.

Concernant les difficultés qui nous ont amené à ne pas pouvoir finir le TP, il y a je pense un manque de partage du travail, c'est-à-dire que nous travaillons ensemble sur les mêmes tâches. De plus on n'a compris que trop tard l'importance d'avoir un cahier avec des schémas 3D. Nonobstant il est vrai que si nous avions commencer le TP avant la séance, nous aurions pu finir.

Personnellement nous avons vraiment aimer ce TP car c'était notre première expérience en programmation 3D et nous sommes fière du résultat que l'on a obtenu.