PONTAUD Thomas
MANOUKIAN Maxime
Date du TP : 22/11/2021

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Ce document présente un modèle de rapport pour les TPs sur lequel il faut vous baser pour le rendu. Présenter d'abord ici l'objet du TP et la méthode générale proposée dans le cadre du TP.

II. Section 1 du TP

1. Utilisation de Pygame

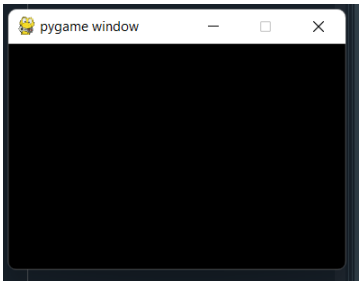
```python
import pygame
pygame.init()
ecran = pygame.display.set_mode((300, 200))
pygame.quit()
```

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
```

```python
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.



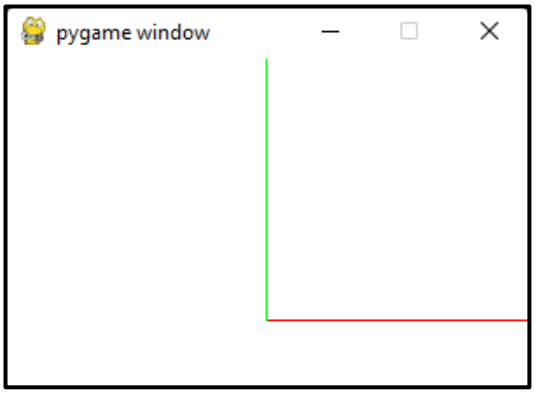
2. Utilisation de PyopenGL

```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)

    # Placer ici l'utilisation de gluPerspective.
    glu.gluPerspective(45, 1, 0.1, 50)

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

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.

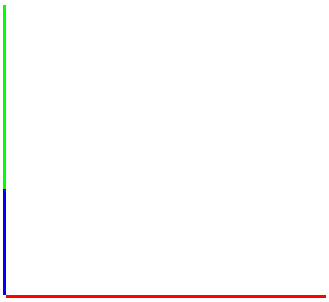
Les lignes 26 à 30 permettent de quitter la fenêtre Pygame en cliquant sur la croix.

```python
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
```



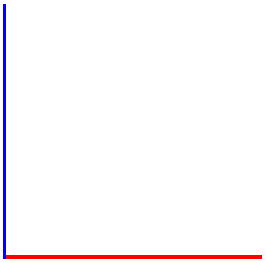
2) On ajoute alors les différents vertices correspondant aux différents axes, précédé de leur couleur au format RGB. Ce qui nous donne l'affichage suivant.

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)
```

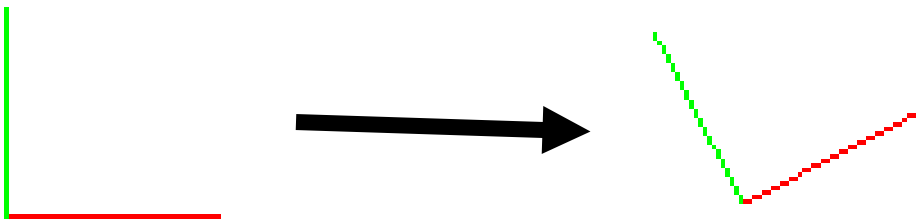


On peut également observer le résultat de la rotation de 90° autour de l'axe x.

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()
```

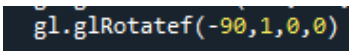

Cette méthode setParameter() permet de changer les paramètres des axes x, y et z. Ces fonctions sont accessibles depuis la console car le fichier Configuration est un fichier qui est utilisé dans le main. Le main va aller chercher la fonction setParameter () dans la classe Configuration grâce à cette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

```
from Configuration import Configuration
```

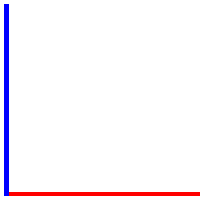
Donc le fait de mettre un point à la fin de Configuration(...). suivis de la fonction setParameter() permet de changer les paramètres de cette configuration, puis de l'afficher avec display(). Ce chainage est possible car setParameter() renvoie un objet de type Configuration.

```
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)
```



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)
```

III. 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.