

Rapport de TP3 – Représentation visuelle d'objets.

I. Introduction

On s'intéresse dans ce TP à la représentation d'objets 3D à l'écran dans une fenêtre graphique qui permet des opérations de zoom, rotation et translations. On a choisi la représentation de maisons à partir d'objets simples que l'on va construire progressivement comme les murs, les portes, les fenêtres...

II. Préparation à faire avant le TP

- Expliquer pourquoi le chaînage de l'appel des méthodes *setParameter()* et *display()* est possible.

Puisque *display()* permet d'afficher alors on peut le chaîner avec *setParameter()*, ce chaînage permet d'afficher les paramètres.

- Un traitement particulier est effectué dans le « setter » pour le paramètre *screenPosition*. Expliquer pourquoi ce traitement particulier doit être effectué.

Ce traitement est effectué pour sauvegarder les paramètres avant la transformation par-la matrice.

- Ajouter une seule instruction à la méthode *initializeTransformationMatrix()* pour que l'axe z soit représenté verticalement sur l'écran et que l'axe x soit représenté horizontalement. L'axe y devient alors la profondeur. Garder cette transformation pour la suite.

```
# Initializes the transformation matrix
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.glRotatef(-90, 1, 0, 0)
```

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

1. Ajouter la gestion des touches « Page Up », « Page Down » pour, respectivement, grossir ou réduire l'affichage, c'est-à-dire effectuer un changement d'échelle (effet de zoom).

```
# 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)

    elif self.event.dict['unicode'] == 'u' or self.event.key == pygame.K_u:
        gl.glScalef(1.1, 1.1, 1.1)
    elif self.event.dict['unicode'] == 'd' or self.event.key == pygame.K_d:
        gl.glScalef(0.9, 0.9, 0.9)
```

2. Remplacer l'instruction pass dans la méthode processMouseButtonDownEvent() pour gérer l'effet de zoom avec la souris.

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if pygame.mouse.get_pressed()[1]==4 :
        gl.glScalef(1.1, 1.1, 1.1)
    elif pygame.mouse.get_pressed()[1]==5 :
        gl.glScalef(0.9, 0.9, 0.9)
```

3. Remplacer l'instruction pass dans la méthode processMouseMoveEvent() pour gérer le déplacement des objets.

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

IV. Création d'une section

2.a En s'inspirant du fichier Configuration.py, écrire la méthode generate(self) de la classe Section qui crée les sommets et les faces d'une section orientée selon l'axe x et dont le coin bas gauche de la face externe est en (0, 0, 0).

```
# 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],
        [self.parameters['width'], self.parameters['thickness'], 0],
        [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
        [0, self.parameters['thickness'], self.parameters['height']],
    ]
    # Définir ici les sommets
    self.faces = [
        [0, 3, 2, 1],
        [0, 3, 5, 4],
        [4, 7, 6, 5],
        [7, 6, 2, 1],
        [0, 5, 6, 1],
        [3, 4, 2, 7],
    ]
    # définir ici les faces
```

2.b Ainsi à l'aide de ces informations et de la documentation de PyOpenGL:

- Analyser la fonction Q2b() dans le fichier main.py et expliquer l'instruction Configuration().add(section).display()

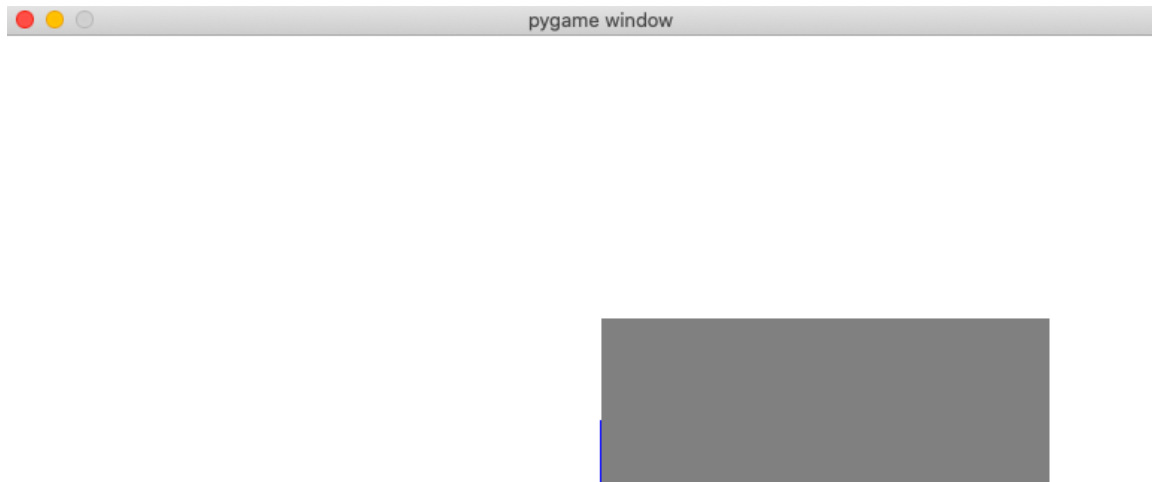
Cette instruction permet de créer et d'afficher une section à partir de ses paramètres renseigner : l'origine, la longueur, la largeur.

- Ecrire la méthode draw() pour la classe Section afin de tracer les faces de la section en gris

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glPushMatrix()
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glEnd()
    gl.glPopMatrix()

    gl.glPushMatrix()
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, self.parameters['thickness'], 0])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glEnd()
    gl.glPopMatrix()
```

- Exécuter la fonction `Q2b()` dans le main pour visualiser la section.



Malheureusement je ne peux en demander plus au logiciel car sur mac il charge indéfiniment. Je peux donc juste voir en 2D car j'ai juste la première image du logiciel après il plante.

2.c

- Ecrire la méthode `drawEdges()` dans la classe `Section`.

```
# Draws the edges
def drawEdges(self):
    # A compléter en remplaçant pass par votre code
    for i in range(len(self.vertices)-1):
        for j in range(i):
            gl.glPushMatrix()
            gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)
            gl.glBegin(gl.GL_QUADS) # Tracé d'une ligne
            gl.glColor3fv([1, 1, 0]) # Couleur gris moyen
            gl.glVertex3fv(self.vertices[j])
            gl.glEnd()
            gl.glPopMatrix()
```

- **Modifier la méthode draw() de la question (2). b) pour que la méthode drawEdges() soit exécutée en premier lorsque le paramètre edges, fourni au constructeur ou via le « setter » setParameter(), prend la valeur True.**

```
# Draws the faces
def draw(self):
    if self.parameters['edges']==True:
        self.drawEdges()

    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, 0])
    gl.glVertex3fv([self.parameters['width'], 0, self.parameters['height']])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glEnd()
    gl.glPopMatrix()

    gl.glPushMatrix()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
    gl.glVertex3fv([0, 0, 0])
    gl.glVertex3fv([0, self.parameters['thickness'], 0])
    gl.glVertex3fv([0, self.parameters['thickness'], self.parameters['height']])
    gl.glVertex3fv([0, 0, self.parameters['height']])
    gl.glEnd()
    gl.glPopMatrix()
```

- **Exécuter la fonction Q2c() du fichier main.py pour visualiser la section avec arrêtes**

V. Création des murs

3.a

- Analyser le fichier **Wall.py** (notamment le constructeur de la classe **Wall**) et expliquer comment cette opération est réalisée.

L'objet mur possède comme paramètre : une longueur, une largeur, une épaisseur, un couleur et une position.

Ainsi le mur est construit avec ses paramètre et une section est aussi créée avec ses paramètres qui permet de trouver l'endroit où insérer le nouvel objet.

- Écrire ensuite dans la méthode **draw** de la classe **Wall**

```
# Draws the faces
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 i in self.objects:
        i.draw()
    gl.glPopMatrix()
```

- Écrire enfin dans la fonction **Q3ba** du fichier **Main.py** des instructions pour créer un mur constitué d'une section parente et le tracer.

```
def Q3a():
    wall1=Wall({'width':2,'height':4})
    wall1.draw()
```

Conclusion

Nous nous sommes avancés jusqu'à la partie création d'une maison.

Ce TP nous a permis d'en apprendre davantage sur les logiciel 3D et à quel point les développeurs nous ont facilité la tâche pour les logiciels de création 3D tel Blender ou Sketchop.