

Compte rendu TP3

Introduction :

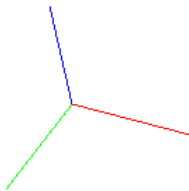
L'objectif de ce Tp est de nous apprendre à manipuler une perspective et de pouvoir créer un objet dans un plan 3D. Pour cela nous utiliserons Pygame et PyOpenGL.

1a :

On a ici une fonction main qui va appeler la classe configuration et va lui affecter des paramètres définis dans la classe.

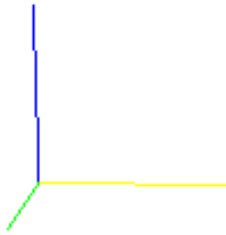
La classe configuration possède différentes fonctions :

- : - La définition de la taille de la fenêtre et la distance d'affichage
- Un affichage d'un repère avec 3 axes
- Une fonction qui permet à l'utilisateur de créer une rotation autour de l'axe z en appuyant soit sur z ou Z selon le sens qu'il choisit



1b:

La modification apportée permet ici de changer la couleur de l'axe sélectionné avec la couleur sélectionnée.



1c:

```
def
initializeTransformationMa
trix(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'])
```

1d:

On utilise ici PAGEUP et PAGEDOWN pour effectuer un changement d'échelle

```
elif self.event.dict['unicode'] == 'Page up' or self.event.key == pygame.K_PAGEUP:
    gl.glScalef(1.1,1.1,1.1)

elif self.event.dict['unicode'] == 'Page down' or self.event.key == pygame.K_PAGEDOWN:
    gl.glScalef(1/1.1, 1/1.1, 1/1.1)
```

1e:

On utilise ici un bouton pour effectuer le changement d'échelle

```
# 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)
```

1f:

Ici on se concentre sur l'action qui s'effectue quand l'utilisateur appuie sur le bouton droit de la souris [2] ou sur le bouton gauche [1].

```
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],0,0)
        gl.glTranslate(0,0,self.event.rel[1])
```

2a :

L'objectif maintenant est de créer une forme de rectangle. Pour cela on crée des sommets en indiquant leurs coordonnées et ensuite on crée des faces à ce rectangle en 3D.

```

def generate(self):
    self.vertices = [
        [0, 0, 0 ], #0
        [0, 0, self.parameters['height']], #1
        [self.parameters['width'], 0, self.parameters['height']], #2
        [self.parameters['width'], 0, 0], #3
        [0,self.parameters['thickness'],0], #4
        [0,self.parameters['thickness'],self.parameters['height']], #5
        [self.parameters['width'],self.parameters['thickness'],self.parameters['height']], #6
        [self.parameters['width'],self.parameters['thickness'],0] #7
    ]
    self.faces = [
        [0,3,2,1],
        [4,7,6,5],
        [3,7,6,2],
        [0,4,5,1],
        [0,3,7,4],
        [1,2,6,5],
    ]

```

2b :

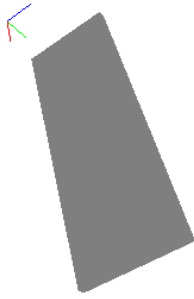
Dans cette question on effectue la même chose que pour la question 1. Cependant, on ajoute la classe section la forme géométrique qu'on vient de définir dans section. On utilise alors la fonction draw

```

def draw(self):
    if self.parameters['edges']: #si le paramètre "edges" prend la valeur "true" alors le dessin des faces peut avoir lieu
        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)
    for x in self.faces:
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv([0.5, 0.5, 0.5])
        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()

```



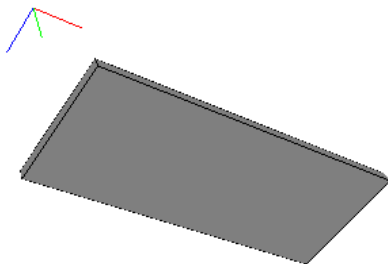
2c:

On ajoute alors la fonction drawEdges pour avoir les contours de nos arrêtes.

```
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)
    for x in self.faces:
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv([0, 0, 0])
        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() #permet de restaurer les coordonnées du système précédent
```

On ajoute à notre draw une conditions afin d'afficher la fonction drawEdges

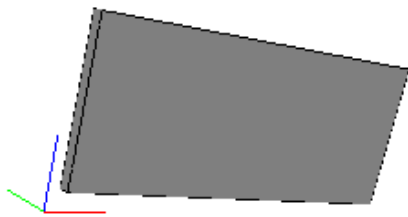
```
if self.parameters['edges']: #si le paramètre "edges" prend la valeur "true" alors le dessin des faces peut avoir lieu
    self.drawEdges()
```



3a :

Dans la classe Wall il y a la méthode self.parentSection qui va appeler le programme section et ainsi créer le parallélépipède.

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



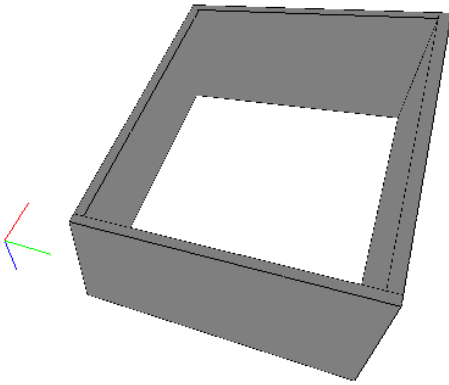
4a :

L'objectif ici est de créer maintenant 4 murs dans notre repère. On utilise maintenant le draw dans le programme house

```
# Draws the house
def draw(self):
    for obj in self.objects:
        obj.draw()
```

On va ensuite dans le main et on crée 4 murs avec une longueur différente selon le mur créé et une orientation pour les murs perpendiculaire. On ajoute 0,2 à deux murs pour combler l'épaisseur du mur.

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [1,1,0], 'width' : 7, 'height' : 2.6, 'edges' : True})
    wall2 = Wall({'position': [1,6,0], 'width' : 7.2, 'height' : 2.6, 'edges' : True})
    wall3 = Wall({'position': [8,6,0], 'width' : 5, 'height' : 2.6, 'edges' : True, 'orientation' : 270})
    wall4 = Wall({'position': [1,1,0], 'width' : 5.2, 'height' : 2.6, 'edges' : True, 'orientation' : 90})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```



5a :

L'objectif maintenant est de créer une ouverture pour notre mur. On commence alors à créer 2 formes ; une pour la fenêtre et une pour la vitre. Cela ressemble à la question 2, on supprimera juste 2 faces.

```

self.vertices = [
    [0, 0, 0 ], #0
    [0, 0, self.parameters['height']], #1
    [self.parameters['width'], 0, self.parameters['height']], #2
    [self.parameters['width'], 0, 0], #3
    [0,self.parameters['thickness'],0], #4
    [0,self.parameters['thickness'],self.parameters['height']], #5
    [self.parameters['width'],self.parameters['thickness'],self.parameters['height']], #6
    [self.parameters['width'],self.parameters['thickness'],0] #7
]
self.faces = [
    #[0,3,2,1],
    #[4,7,6,5],
    [3,7,6,2],
    [0,4,5,1],
    [0,3,7,4],
    [1,2,6,5],

]

```

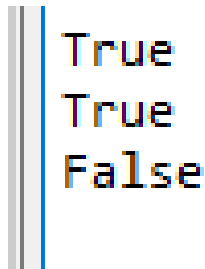


5b :

Maintenant on va instaurer des conditions à cette ouverture avec la fonction CanCreateOpening : l'objectif est de savoir si l'ouverture est incluse dans un mur déjà créé. On va donc vérifier si ces caractéristiques sont les mêmes.


```
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    if self.getParameter('position')[0] + self.getParameter('width') < x.getParameter('position')[0] + x.getParameter('width'):
        #si la position (en x) + la longueur du mur < position + longueur de l'ouverture : ce n'est pas possible
        return False
    elif x.getParameter('position')[0] + x.getParameter('width') < self.getParameter('position')[0]:
        #si la position (en x) + la longueur de la fenêtre < la position de départ du mur : ce n'est pas possible non plus (on ne peut pas faire d'ouverture dans le vide)
        return False
    elif self.getParameter('position')[2] + self.getParameter('height') < x.getParameter('position')[2] + x.getParameter('height'):
        #si la position (en z) + la hauteur du mur < position + hauteur de l'ouverture : ce n'est pas possible
        return False
    elif x.getParameter('position')[2] + x.getParameter('height') < self.getParameter('position')[2]:
        #si la position (en z) + la hauteur de la fenêtre < la position de départ du mur : ce n'est pas possible non plus (on ne peut pas faire d'ouverture dans le vide)
        return False
    elif self.getParameter('thickness') != x.getParameter('thickness'):
        #si l'épaisseur du mur est différente de celle de l'ouverture : on ne peut pas la faire
        return False
    else :
        return True
```

En lançant la fonction dans le main on obtient



```
True
True
False
```

Mais l'opening 3 est impossible car la position en z + la hauteur de l'ouverture > la hauteur du mur => $1+1.7=2.7 > 2.6$ (pas d'ouverture dans le vide). Ainsi le code semble être fonctionnel.

Pour le reste du tp nous n'avons malheureusement pas eu le temps de finir.

Conclusion :

Pour conclure, nous avons appris à gérer une structure en 3D et au plus nous avançons dans le tp au plus cela nous semblait concret et donc plus motivant. Notre principale difficulté a été de comprendre l'utilisation de ce nouveau langage et ses utilisations. Le temps de compréhension et d'application ont été un peu trop longs ce qui ne nous a pas permis d'aller au bout du sujet.