JOLLET Léo
CRISCUOLO Vincent
13/01/22

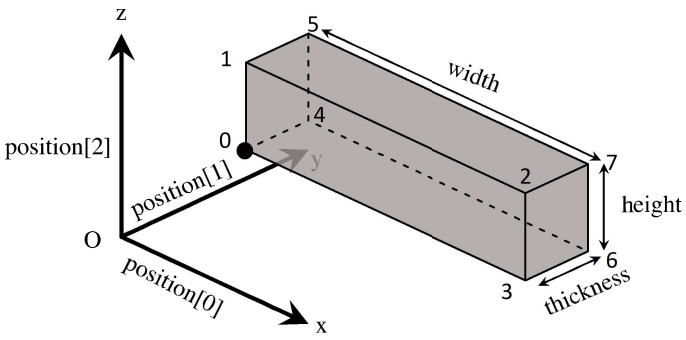
Rapport de TP3 Partie 2
Représentation visuelle d'objets.

I. Introduction

Ce TP reprend ce que nous avons commencé à faire le mois dernier sur la représentation visuelle d'objets, plus particulièrement d'une maison avec ses murs. Ce TP commence donc à la partie 4, les 3 premières parties étant sur le rapport précédent. Nous avons effectué ce TP sur Replit.

II. Création d'une section

Cette partie consiste à créer une section avec ses différents paramètres. En voici une représentation avec ses paramètres :



Question 2a) On nous demande d'écrire la méthode **generate(self)** de la classe Section permettant de créer les sommets et les faces d'une section par rapport au point [0,0,0] en 0. On écrit donc toutes les coordonnées des sommets par rapport à ce repère en utilisant les paramètres de la section :

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

Question 2b) Une fois avoir généré les sommets et les faces de la section il nous faut pouvoir les voir sur l'interface graphique, c'est pourquoi on nous demande d'écrire la méthode **draw()** dans Section qui permet cela. Nous créons ici d'abord une matrice de projection propre à la section, et qui sera relative par rapport à la matrice de projection globale de l'application, en utilisant Opengl.

L'instruction Configuration.add(section).display() permet de créer une section avec des paramètres définit de la classe Configuration. Ainsi :

```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
                Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
                )
```

Cette fonction nous renvoie une section à la position en x=1, en y=1, z=0, de longueur=7 et de hauteur=2,6. Cependant, pour qu'elle puisse s'afficher nous avons du écrire la méthode **draw()** de cette manière :

```
def draw(self):
  gl.glPushMatrix() #Matrice de projection
  gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],
  self.parameters['position'][2]) # Permet la translation suivant les 3 axes
  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(self.parameters['color'])
  for face in self.faces:
    for vertex in face:
      gl.glVertex3fv(self.vertices[vertex])
  gl.glEnd()
  gl.glPopMatrix()
```

On effectue une boucle pour chaque vertices dans la liste face

Cela nous renvoie :

On constate que cela nous renvoie bien un parallélépipède, or, nous ne pouvons pas distinguer ses arrêtes. C'est pourquoi, il nous est demandé dans la question suivante de les dessiner.

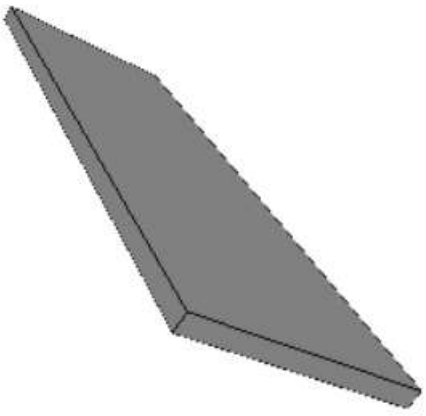
Question 2c) Pour cela, on va écrire la méthode **drawEdges()** en appliquant le même procédé que précédemment, et mettre une couleur différente aux arrêtes afin de bien distinguer la face et les arrêtes. On a donc :

```
def drawEdges(self):
  gl.glPushMatrix() #Déplacement de ['position'] selon x  #Déplacement de ['position'] selon y  #Déplacement de ['position'] selon z
  gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) #On trace des lines donc GL_LINE
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv([self.parameters['color'][0]*0.1,self.parameters['color'][1]*0.1,self.parameters['color'][2]*0.1])
  for face in self.faces:
    for vertex in face:
      gl.glVertex3fv(self.vertices[vertex])
  gl.glEnd()
  gl.glPopMatrix()
```

On modifie la méthode **draw()** de la classe Section en ajoutant une boucle permettant à l'utilisateur d'affiché où non les arrêtes de la section, ainsi :

```
def draw(self):
  if self.parameters['edges']: #Boucle d'affichage des arrêtes
    self.drawEdges()
  gl.glPushMatrix()
  gl.glTranslatef(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
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv(self.parameters['color'])
  for face in self.faces:
    for vertex in face:
      gl.glVertex3fv(self.vertices[vertex])
  gl.glEnd()
  gl.glPopMatrix()
```

En relançant la fonction précédente, on observe :



Par conséquent, on distingue désormais les arrêtes de la face.

III. Création des murs

Cette partie se consacre désormais à la création de murs pour pouvoir former une maison. Après avoir configurer la classe Section, nous allons faire celle se nommant Wall. Cette classe à des paramètres qui sont identiques à celle de la Classe Section mais d'autre qui sont nouveaux :

```
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]
```

#Identiques à la classe Section

#Nouveaux paramètres

De la même manière, il va nous falloir afficher les murs que nous allons créer, par conséquent, il nous faut modifier la méthode **draw()** de la classe Wall en suivant le même raisonnement que précédemment. Ainsi, on écrit :

```
def draw(self):
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'],0,0,1) *
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    self.parentSection.drawEdges()
    for I in self.objects:
        I.draw()
    gl.glPopMatrix()
```

#On dessine aussi les objets éventuellement contenus dans l'attribut objects

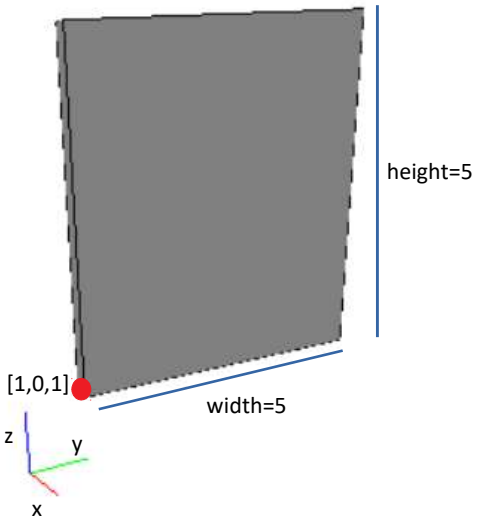
* On utilise Opengl pour effectuer une rotation du mur selon l'axe z suivant le paramètre initial. Nous avons pas eu besoin d'effectuer de translation puisqu'elle y est déjà dans la classe Section.

Question 3a) Pour créer un mur constitué d'une section, on peut écrire la fonction suivante :

```
def Q3a():
    return(Configuration().add(Wall({'position':[1,0,1],'width':5, 'height':5,'orientation':90})))
```

Ce mur est à la position [1,0,1], de longueur=5, de hauteur=5 et orienté de 90° par rapport à l'axe z. En exécutant la fonction on observe :

IV. Création d'une maison



Maintenant qu'il nous est possible de créer des murs nous pouvons constituer les 4 murs d'une maison. Pour cela nous allons utiliser la classe House et modifier sa méthode **draw()**, permettant son affichage. La maison est constituée de murs qui peuvent translater et tourner. Par conséquent, la méthode draw() s'écrit de la façon suivante :

```
def draw(self):
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'],0,0,1)
    gl.glTranslatef(self.parameters['position'][0],self.parameters['position'][1],0)
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for I in self.objects:
        I.draw()
    gl.glPopMatrix()
```

#Permet la rotation suivant l'axe z et la translation suivant les 3 axes.

#On dessine aussi les objets éventuellement contenus dans l'attribut objects

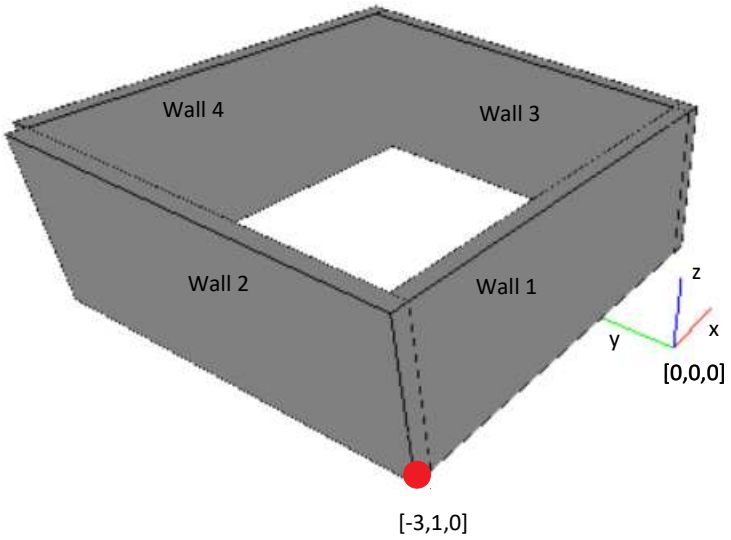
Désormais, il ne nous reste plus qu'à créer 4 murs de manière à ce qu'il forme un rectangle, bâti de notre maison.

Pour cela, nous avons écrit :

```
def Q4a():
    wall1 = Wall({'position':[0,0,0],'width':5,'height':2,'orientation':0,'thickness':0.2})
    wall2 = Wall({'position':[0,0,0],'width':5,'height':2,'orientation':90,'thickness':0.2})
    wall3 = Wall({'position':[0,-5,0],'width':5,'height':2,'orientation':90,'thickness':0.2})
    wall4 = Wall({'position':[0,5,0],'width':5,'height':2,'orientation':0,'thickness':0.2})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

#Repère de la maison par rapport au repère en [0,0,0]

Représentation de la maison :



IV. Création d'une d'ouvertures

Une maison est généralement accompagnée de portes et de fenêtres, c'est pourquoi nous allons devoir faire des ouvertures sur les murs.

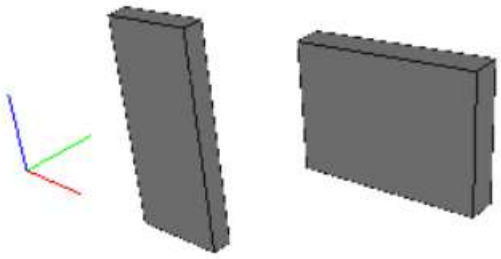
Pour cela, nous allons utiliser la classe Opening. Dans un premier temps, nous allons écrire la méthode **draw()** de la classe Opening permettant de dessiner les ouvertures. On a, en suivant le même procédé que précédemment :

```
def draw(self):
  gl.glPushMatrix()
  gl.glTranslatef(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
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv([self.parameters['color'][0]*0.6,self.parameters['color'][1]*0.6,self.parameters['color'][2]*0.6])
  for face in self.faces:
    for vertex in face:
      gl.glVertex3fv(self.vertices[vertex])
  gl.glEnd()

  gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les faces : GL_FILL
  gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
  gl.glColor3fv([self.parameters['color'][0]*0.1,self.parameters['color'][1]*0.1,self.parameters['color'][2]*0.1])
  for face in self.faces:
    for vertex in face:
      gl.glVertex3fv(self.vertices[vertex])
  gl.glEnd()

  gl.glPopMatrix()
```

#Trace les faces de l'ouverture

#Trace les arrêtes de l'ouverture

Nous avons modifié la couleur entre les arrêtes et la face de façon à bien les distinguer. On peut exécuter le contenu de la fonction **Question 5a)**. On obtient :



On observe une porte et une fenêtre.

V.Conclusion

En conclusion, nous n'avons pas réussi à aboutir à la fin de ce TP car les questions étaient assez complexes par la suite et l'écriture de nos méthodes nous donnait des affichages incohérents. Cependant, ce TP nous a permis d'appréhender Opengl, les classes et la syntaxe python dans son ensemble. De plus, il nous a été intéressant de partir de simples points dans un repère pour aboutir à créer un ensemble de murs et de pouvoir se déplacer dans celui-ci.