Gonez Arthur (@arthurgonay)
Descomps Alexandre

Compte rendu TP3 Part 2/2

IV/ Création d'une section :

2/

a) Méthode generate(self) de la classe Section qui crée les sommets et les faces d'une section :

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

Nous travaillons sur un parallélépipède, il fallait donc créer 8 sommets et 6 faces. Les coordonnées dépendent de la largeur (x), la profondeur(y) et la hauteur(z)

b) Dessin d'une section

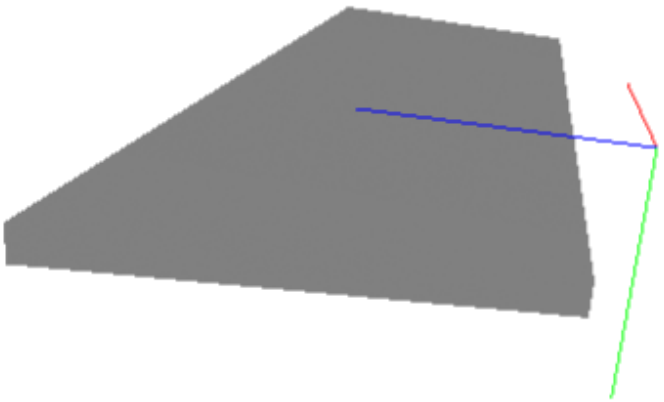
```
def Q2b():
    # Ecriture en utilisant le chaînage
    return Configuration().add(
                Section({'position': [1, 1, 0], 'width':7, 'height':2.6})
                )
```

Cette fonction permet d'ajouter les paramètres "largeur = 7" , "hauteur = 2.6", et "position = [1,1,0]" à notre objet section dans la configuration initiale. Cela nous permettra de la visualiser lorsque l'on aura écrit la méthode "draw"

On dessine notre section à 6 faces en utilisant les paramètres de la question Q2b) et notre précédente méthode generate pour générer les faces:

```
# Draws the faces
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([0.5, 0.5, 0.5]) # Couleur gris moyen
    for face in self.faces:
      for vertex in face:
        gl.glVertex3fv(self.vertices[vertex])
    gl.glEnd()
    gl.glPopMatrix()
```

On peut visualiser cette section en exécutant la question Q2b) du main

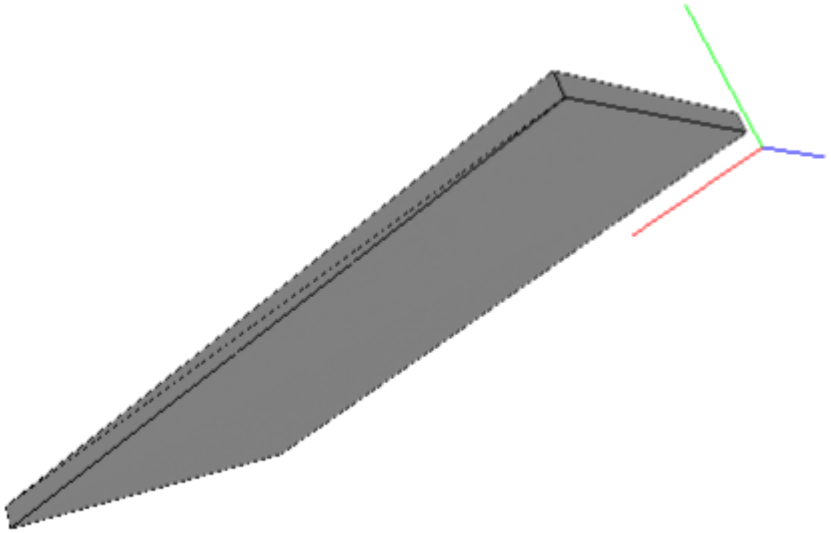


c) Dessin des arêtes :

```
# Draws the edges
def drawEdges(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_LINE)
    gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
    gl.glColor3fv([0, 0, 0]) # Couleur noir
    for face in self.faces:
        for vertex in face:
            gl.glVertex3fv(self.vertices[vertex])
    gl.glEnd()
    gl.glPopMatrix()
```

```
# Draws the faces
def draw(self):
    gl.glPushMatrix()
    self.drawEdges()
    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([0.5, 0.5, 0.5]) # Couleur gris moyen
    for face in self.faces:
        for vertex in face:
            gl.glVertex3fv(self.vertices[vertex])
    gl.glEnd()
    gl.glPopMatrix()
```

Pour dessiner les arêtes nous avons réutilisé le code pour créer les faces en remplaçant “gl.GL_FILL” par “gl.GL_LINE”. Ainsi seulement les lignes seront tracées sans remplissage. Pour bien constater cette différence nous avons aussi changé la couleur pour que les arêtes puissent ressortir.
Enfin, il fallait appeler cette méthode dans la méthode draw pour qu’elle soit exécutée en premier.
On visualise ensuite le résultat :



V/ Création des murs :

3/
a) Création des murs

Analyse du fichier wall.py :
Après avoir récupéré ou set des paramètres par défaut, on crée une liste “objects” dans laquelle nous allons ajouter la section parente (contenant les valeurs de width,thickness …).

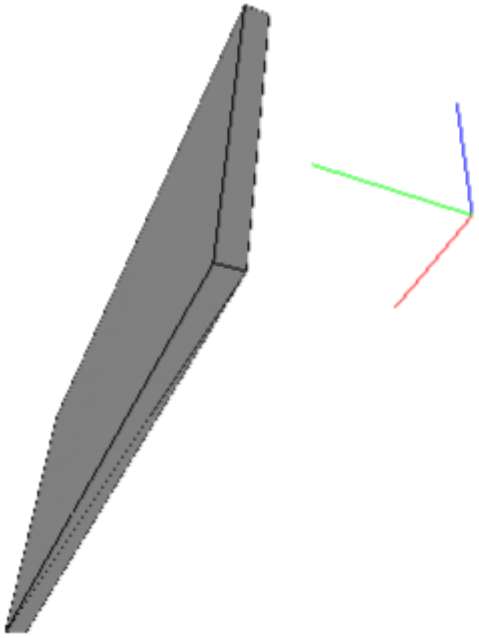
Pour dessiner tous les objets de notre liste “objects”, nous utilisons une boucle dans notre méthode “draw” :

```
def draw(self):
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'],0,0,1)
    for section in self.objects:
        section.draw()
    gl.glPopMatrix()
```

De façon analogue à la fonction draw de la section on utilise est matrice afin de réaliser des opérations dessus, telles que des rotations comme ci-dessus.
Enfin, on ajoute le code dans la fonction Q3a) pour ajouter un objet mur :

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

Cela nous donne ce mur :



VI/ Création d'une maison :

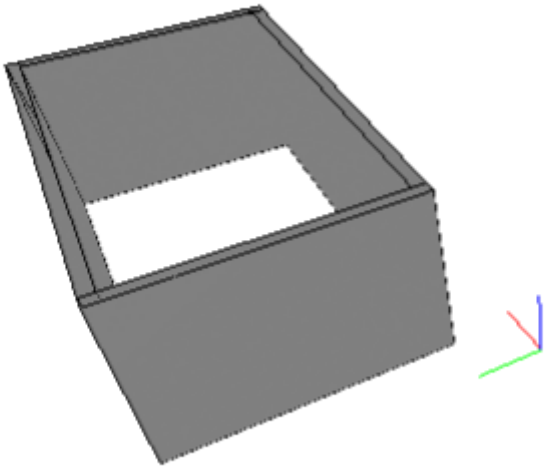
on réalise la fonction *draw()* de la classe *house* de façon analogue à la fonction *draw()* de *wall :*

```
def draw(self):
  for wall in self.objects:
    wall.draw()
```

On paramètre les murs de la maison directement dans la main. Pour réaliser un semblant de maison, on tourne 2 murs à 90°. Il faut ensuite ajouter chacun des murs à la maison avec la fonction *add().*

```
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, 5, 0], 'width':7, 'height':2.6, 'edges':True})
    wall3 = Wall({'position': [1, -1, 0], 'width':4.2, 'height':2.6, 'edges':True,'orientation':90})
    wall4 = Wall({'position': [1, -8, 0], 'width':4.2, '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)
```

On obtient le résultat ci-dessous:



VII/ Création d’ouvertures :