Rapport de TP
FI-3 INFO501 2021-2022
Polytech Annecy-Chambéry – Université Savoie Mont-Blanc

DORNIER Lisa
LEGLISE Cloé
26/11/2021

Rapport du TP3 – Représentation visuelle d'objets

I. Introduction :

Les objectifs de ce TP sont les suivants :

- Représentation d'objets 3D à l'écran

Nous devons tout d'abord importé deux librairies importantes :

- Pygame : Permet de construire des jeux en Python.
- PyOpenGL : Permet d'accéder en Python aux très nombreuses fonctions de visualisation de la bibliothèque OpenGL dédiée à l'affichage de scènes tridimensionnelles à partir de simples primitives géométriques.

II. Préparation :

Utilisation de Pygame :

Pygame nous permet de gérer une fenêtre d'affichage graphique avec laquelle nous pouvons interagir.

Question 1 :

Dans cette question, nous allons tester l'utilisation de Pygame avec le code suivant :

```
1  import pygame
2  pygame.init()
3  ecran = pygame.display.set_mode((300, 200))
4  pygame.quit()
```

A l'exécution de ce code, une fenêtre s'ouvre et se referme immédiatement.

Explication :

On commence par importer le module pygame que l'on va par la suite lancer. Ensuite, le code configure la fenêtre d'affichage. La dernière ligne fait quitter pygame.

Question 2 :

Nous allons maintenant afficher une fenêtre sans qu'elle se ferme immédiatement. Pour cela, il faut demander à Pygame de gérer un événements comme « Attends que je clique sur la croix pour fermer ma fenêtre ».

Nous allons donc tester le code suivant :

```
1  import pygame
2
3
4  pygame.init()
5  ecran = pygame.display.set_mode((300, 200))
6
7  continuer = True
8  while continuer:
9      for event in pygame.event.get():
10         if event.type == pygame.KEYDOWN:
11             continuer = False
12
13 pygame.quit()
```

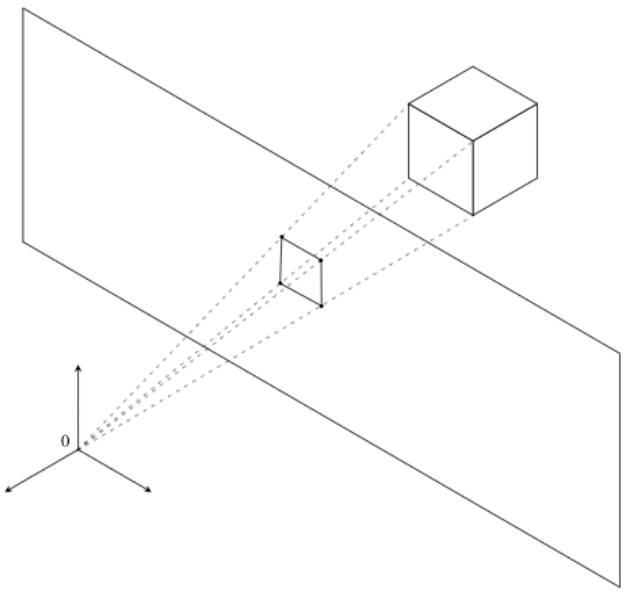
Lors de l'exécution du code, une fenêtre s'ouvre et rien ne se passe. A l'appui d'une touche du clavier, la fenêtre se referme.

Explication :

Après avoir importé pygame, nous le lançons. Ensuite nous configurons l'affichage. A la ligne 7, nous attribuons une valeur à la variable 'continuer'. Ensuite une boucle while est exécutée, tant que *continuer* ne change pas de valeur, la boucle continue. A la ligne 9, une boucle *if* est écrite au cas où un événement se produirait dans pygame. Si l'événement est un appui sur une touche du clavier, on attribue la valeur False à la variable continuer. Une fois la boucle terminée, on ferme pygame avec la ligne 13.

- Utilisation de Pyopengl :

Pyopengl nous permet de représenter des objets 3D. Nous allons essayer de représenter des objets 3D sur la fenêtre graphique crée par le module Pygame.
Nous allons considérer un repère de coordonnées pour lequel l'observateur se situe à l'origine et regarde la scène devant lui au travers d'un écran. Un objet 3D (constitué d'un grand nombre de points appelés vertices) est représenté sur l'écran par un point obtenu en prenant l'intersection de la droite partant de l'œil et allant au vertice avec l'écran.
Le principe est illustré par le schéma suivant :



Nous allons utiliser des coordonnées homogènes :

- Au lieu de considérer un vecteur de coordonnées $[x, y, z]^T$ de dimension 3 pour représenter la position des points sur le repère, nous allons définir un vecteur de dimension 4 $[x_h, y_h, z_h, w]^T$ afin que la composante soit une composante de normalisation. On utilise les relations suivantes :

$$\begin{cases} x_h = x/w \\ y_h = y/w \\ z_h = z/w \end{cases}$$

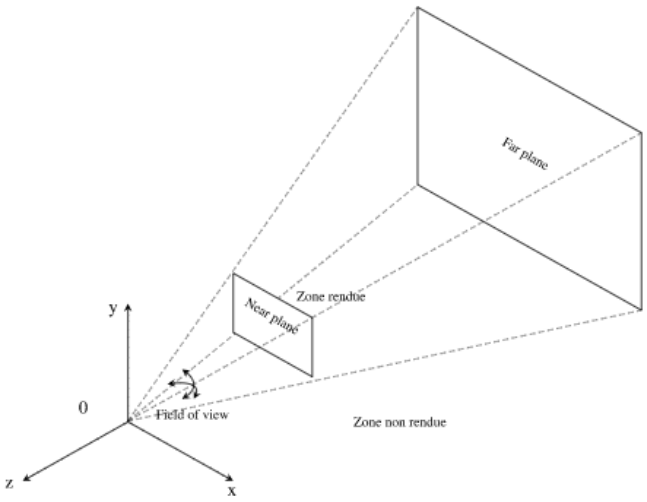
- L'intérêt de ce système de coordonnées est de pouvoir se balader sur le plan de perspective en modifiant le facteur w.

Dans le cadre de ce TP, **w = 1.**

Pour chaque opération de translation, rotation ou de changement d'échelle d'un point du repère, nous utilisons une matrice de dimensions 4x4.

- Changement d'échelle : glScalef(a, b, c)
- Translation : glTranslatef(a, b, c)
- Rotation : glRotatef0(theta, a, b, c)

Pour utiliser OpenGL, nous allons définir une **matrice de perspective**. On ajoute à cela le principe de **clipping** qui consiste à ne pas calculer la projection de tous les objets de l'espace mais seulement ceux définis dans un espace 3D définit par deux angles et deux plans de la façon suivante :



On obtient la matrice perspective avec gluPerspective(fovy, aspect, zNear, zFar).

Question 1 :

Nous faisons un copier/coller du code donné dans un nouveau fichier et nous avons ajouté la fonction gluPerspective à la ligne 17. Nous avons testé la fonction et elle marche. Il est maintenant possible de fermer la fenêtre avec la croix.

Question 2 :

Le but de cette question est de tracer des axes. Nous allons utiliser les fonctions glBegin, glColor3fv, glVertex3fv et glEnd. En s'aidant du code fourni, nous allons tracer les axes x, y et z.

Nous obtenons le code suivant :

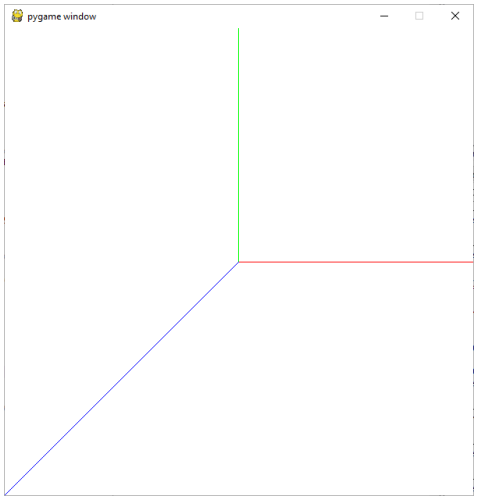
```
23      # Placer ici l'utilisation de gluPerspective.
24
25      glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
26
27      gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments)
28      gl.glColor3fv([1, 0, 0]) # Indique la couleur du prochian segment en RGB
29      gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
30      gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne
31
32      gl.glColor3fv([0, 1, 0])
33      gl.glVertex3fv((0,0, -2))
34      gl.glVertex3fv((0, 1, -2))
35
36      gl.glColor3fv([0, 0, 1])
37      gl.glVertex3fv((0,0, -2))
38      gl.glVertex3fv((0, 0, -1))
39
40      gl.glEnd() # Find du tracé
41      pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
42
```

Explication :

De la ligne 27 à 30 nous traçons le premier segment (rouge), de la ligne 32 à 34 nous traçons le deuxième segment (vert), et pour finir, nous traçons le troisième segment (bleu) de la ligne 36 à 38. On ne voit pas l'axe bleu car nous sommes dans le plan et pas dans l'espace, en effet l'axe bleu est dirigé vers nous, il est orthogonal à la fenêtre d'affichage.

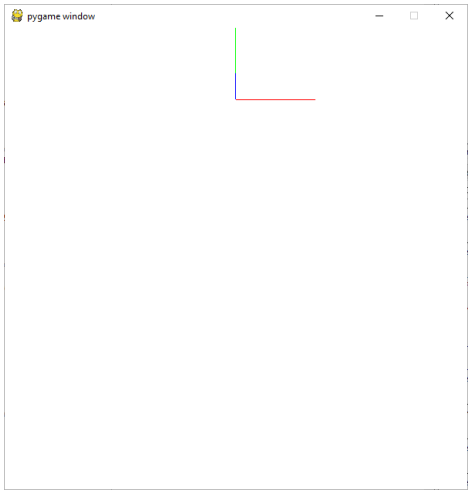
Nous obtenons le résultat suivant :

Pour afficher l'axe bleu, nous modifions la ligne 38 de cette façon : gl.glVertex3fv((-1, -1, -1)).
On obtient l'affichage suivant :

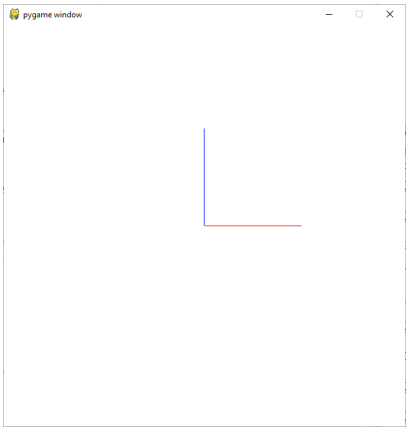


Question 3 :

Nous voulons maintenant déplacer la position de l'écran, nous allons faire cela grâce à une translation.
On utilise la fonction glTranslatef après la fonction gluPerspective.
Nous obtenons l'affichage suivant :



On remarque que le segment bleu apparaît en perspective, aligné sur le segment vert.
On effectuera ensuite une rotation avec la fonction glRotatef. On fera une rotation de 90° autour de l'axe x. On obtient l'affichage suivant :



Suite à la rotation autour de l'axe x, l'axe bleu apparaît et l'axe vert se retrouve orthogonal au plan d'affichage, il n'apparaît donc plus à l'écran.

Nous avons utilisé dans cette partie le code suivant :

```
glu.gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
gl.glTranslatef(0.0, 2, -5)
gl.glRotatef(-90, 1, 0, 0)
```

- Découverte de l'environnement du travail du TP :

Le code sera organisé de la manière suivante :

- La classe Configuration qui gère l'affichage de la fenêtre graphique avec Pygame, les évènements au clavier/souris et une liste d'objets à dessiner sur la fenêtre. Les objets sont des instances de classes à réaliser dans d'autres fichiers.
- Les classes d'objets dessiner dans les fichiers Section.py, Wall.py, Door.py, Window.py et House.py.
- Le fichier main.py est le fichier d'exécution principal pour lequel on importe la classe Configuration et à laquelle on ajoute les éléments à dessiner.

Question 1 a :

On ajoute à la fonction Q1a() la commande « return Configuration() ».
On analyse le fichier Configuration :

- Ligne 14 à 50 : Initialisation de classe, paramétrisation des axes et de la fenêtre d'affichage, initialisation des modules pygame et OpenGL puis des fonctions de transformation et de la liste d'objets, génération de coordonnées.
- Lige 53 à 58 : Fonction d'initialisation de pygame (lancement, paramétrage d'affichage et affichage).
- Ligne 61 à 67 : Fonction d'initialisation de glOpen (paramétrage de la couleur blanche de l'écran, mise en place de l'effet de perspective afin de voir que les surfaces apparentes à l'observateur situé derrière l'écran.
- Ligne 70 à 77 : Fonction d'initialisation des matrices de transformation à l'aide des fonctions de openGL et paramétrage de la position de l'écran.
- Ligne 80 à 81 : Fonction permettant d'afficher les paramètres sans faire appel au constructeur.
- Ligne 84 à 88 : Fonction permettant de paramétrer et effectuer une transformation selon les coordonnées de l'écran.
- Ligne 90 à 102 : Génération des coordonnées des vertices et edges (sommet et arrêtes).
- Ligne 105 à 107 : Surcharge de l'opération addition.
- Ligne 110 à 135 : Affichage des axes x, y et z.
- **Lignes 138 à 148** : Fonction qui gère certaines interactions du clavier : Un appui sur la touche z fait tourner la figure selon l'axe y dans le sens trigonométrique, un appui au même moment sur la touche majuscule et la touche z fait tourner la figure selon l'axe y dans le sens inverse, un appui sur la touche a fait disparaître ou apparaître le dessin.
- Ligne 151 à 156 : Fonctions qui gèrent les événements liés à des manipulations de la souris. (Elle n'est pas encore écrite)
- Ligne 159 à 194 : Affichage et coordination des différentes fonctions et des événements.

Analyse du fichier main :

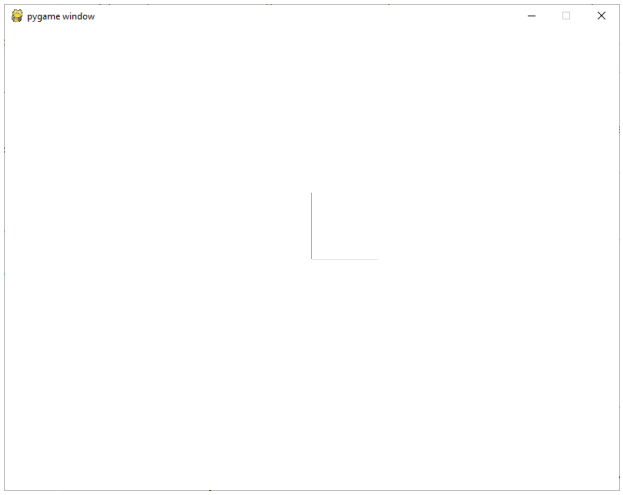
- Liste des fonctions des réponses aux questions du TP (elles ne sont pas encore écrites).
- En bas, l'activation des fonctions-réponses est commentée pour le moment.

Question 1 b :

Nous devons analyser l'effet de la modification suivante :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

Elle entraîne un changement de couleur de l'axe des x. Le mélange de rouge et vert donne du jaune, donc l'axe des x se retrouve coloré en jaune.



On modifie les couleurs des axes x et y à l'aide de la méthode setParameter() :

```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}). \
    setParameter('xAxisColor', [1, 1, 0]). \
    setParameter('yAxisCo lor', [0,1,1]). \
    display()
```

Remarque : les \ permettent d'écrire une instruction sur plusieurs lignes

Le chaînage des fonctions setParameter et display est possible car setParameter nous renvoie un objet et le chaînage des deux fonctions permet d'afficher l'objet en question.

Nous effectuons un traitement particulier dans le setter pour le paramètre screenPosition. Si la fonction est appliquée à la position de l'écran screenPosition, il faut faire appel à la fonction initializeTransformationMatrix(self) qui paramètre l'écran.

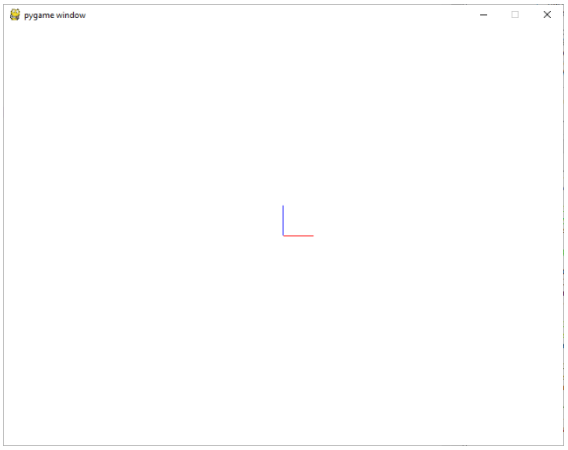
Question 1 c :

On ajoute une instruction à la méthode initializeTransformationMatrix() afin que l'axe z soit représenté verticalement sur l'écran et que l'axe x soit représenté horizontalement. L'axe y devient la profondeur.

On ajoute l'instruction à la ligne 78 :

```
69      # Initializes the tranformation matrix
70      def initializeTransformationMatrix(self):
71          gl.glMatrixMode(gl.GL_PROJECTION)
72          gl.glLoadIdentity()
73          glu.gluPerspective(70, (self.screen.get_width()/self.screen.get_height()), 0.1, 100.0)
74
75          gl.glMatrixMode(gl.GL_MODELVIEW)
76          gl.glLoadIdentity()
77          gl.glTranslatef(0.0,0.0, self.parameters['screenPosition'])
78          gl.glRotatef(-90, 1, 0, 0)
79
```

Nous obtenons l'affichage suivant (pour des raisons de visibilité, nous avons remodifié la fonction Q1a pour que l'axe x reste rouge) :



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

Nous allons gérer dans cette partie les interactions possibles de notre application avec l'utilisateur.

Pour gérer trois types d'événements (clavier ou souris), on appelle les méthodes suivantes :

- processKeyDownEvent : évènements de type appuie sur une touche du clavier
- processMouseButtonEvent : évènements où on clique sur un bouton de la souris
- processMouseMotionEvent : évènements où l'on bouge la souris

Question 1 d :

Pour l'instant, la méthode processKeyDownEvent gère les touches suivantes :

- « a » : afficher ou supprimer les axes
- « z » : rotation de 2.5 degrés autour de l'axe z
- « Z » : rotation de -2.5 degrés autour de l'axe z

Nous devons ajouter la gestion des touches « Page Up » et « Page Down » afin de grossir ou réduire l'affichage, nous effectuons un changement d'échelle.

Remarque : Le facteur d'échelle pour un zoom positif devra être de 1.1, et de 1/1.1 pour un zoom négatif (dans les trois axes).

```
138     # Processes the KEYDOWN event
139     def processKeyDownEvent(self):
140         # Rotates around the z-axis
141         if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
142             gl.glRotate(-2.5, 0, 0, 1)
143         elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
144             gl.glRotate(2.5, 0, 0, 1)
145 
146         # Draws or suppresses the reference frame
147         elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
148             self.parameters['axes'] = not self.parameters['axes']
149             pygame.time.wait(300)
150 
151         #Zoom
152         elif self.event.dict['unicode'] == 'K_PAGEUP' or self.event.key == pygame.K_PAGEUP :
153             gl.glScalef(1.1, 1.1, 1.1)
154 
155         elif self.event.dict['unicode'] == 'K_PAGEDOWN' or self.event.key == pygame.K_PAGEDOWN :
156             gl.glScalef(1/1.1, 1/1.1, 1/1.1)
157 
```

Les lignes 152 à 153 correspondent au zoom positif et les lignes 155 à 156 au zoom négatif.

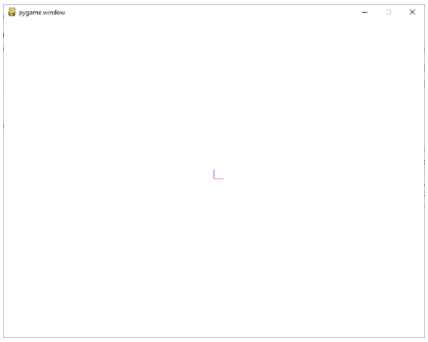
Question 1 e :

Nous reconnaissons un évènement sur la souris lorsque le type de l'évènement est MOUSEBUTTONDOWN. Nous voulons dans cette question zoomer avec la mollette de la souris. Le sens de l'action sur la mollette est donnée par l'attribut button, il prend les valeurs 4 ou 5.

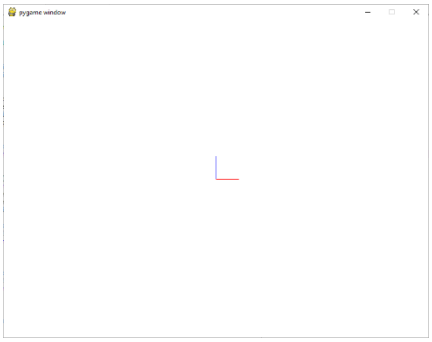
On remplace l'instruction pass dans la méthode processMouseButtonDownEvent() :

```
158     # Processes the MOUSEBUTTONDOWN event
159     def processMouseButtonDownEvent(self):
160         if self.event.button == 4 :
161             gl.glScalef(1.1, 1.1, 1.1)
162         elif self.event.button == 5 :
163             gl.glScalef(1/1.1, 1/1.1, 1/1.1)
164 
```

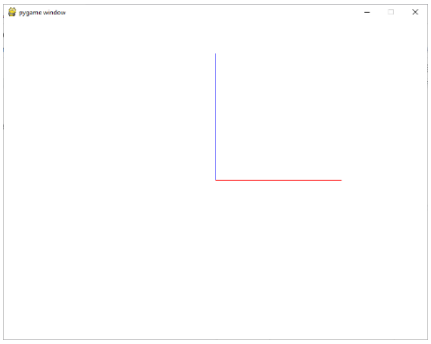
Nous obtenons les affichages suivants :



Zoom négatif



Etat normal



Zoom positif

Question 1 f :

Nous souhaitons maintenant déplacer les objets affichés en bougeant la souris lorsque le bouton de droite ou de gauche est activé. Le bouchon de gauche servira aux rotations et le bouton de droite aux translations.

L'état des boutons de la souris est détecté grâce à la méthode get_pressed(), cette méthode retourne l'état du bouton de gauche, de la molette et du bouton de droite. Pour avoir juste l'état du bouton de

gauche on tape l'instruction suivante : pygame.mouse.get_pressed()[0]. L'instruction pygame.mouse.get_pressed()[2] retourne l'état du bouton de droite. Si la valeur est 1, le bouton est enfoncé, si la valeur est 0, le bouton est relâché.

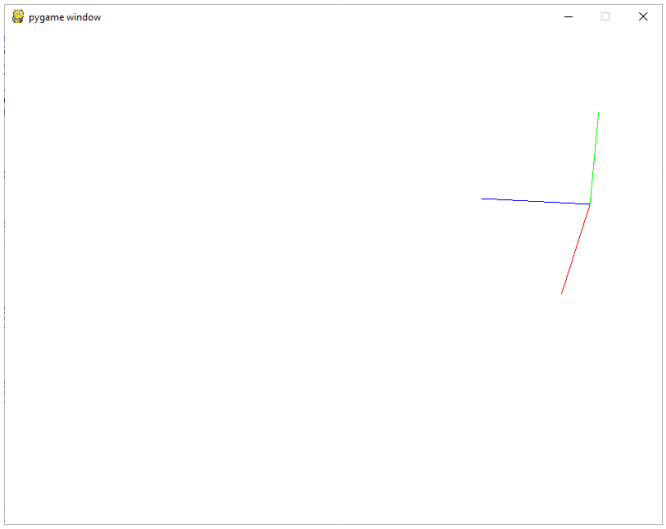
Lorsque la souris est déplacée, des évènements de type MOUSEMOTIONS sont engendrés. L'attribut rel de l'évènement fournit un tuple contenant le déplacement en x et en y de la souris. Ainsi, self.event.rel[0] est le déplacement en x et self.event.rel[1] est le déplacement en y.

Nous remplaçons l'instruction pass dans la méthode processMouseotionEvent() afin de gérer le déplacement des objets :

```
165     # Processes the MOUSEMOTION event
166     def processMouseMotionEvent(self):
167         if pygame.mouse.get_pressed()[0] == 1 :
168             gl.glRotatef(self.event.rel[1], 1, 0, 0)
169             gl.glRotatef(self.event.rel[0], 0, 0, 1)
170         elif pygame.mouse.get_pressed()[2] == 1 :
171             gl.glTranslatef(self.event.rel[0], 0, -self.event.rel[1])
```

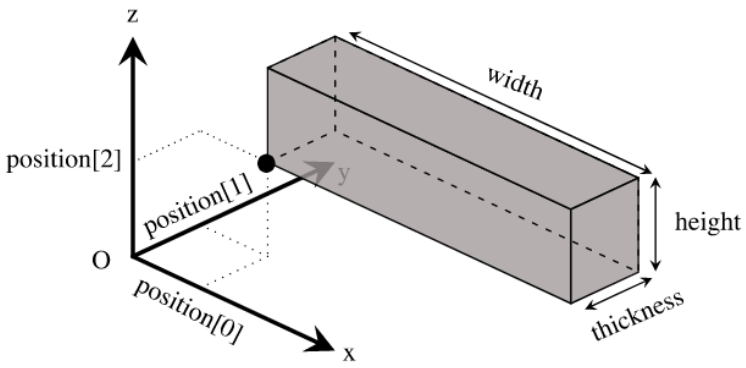
On met un signe négatif pour que le mouvement aille dans le même sens que la souris.

Nous obtenons le résultat suivant :



IV. Création d'une section :

On veut dans cette partie créer une section. Une section correspond à un parallélépipède à 8 sommets et 6 faces :



Une section est constituée du dessin de ses arrêtes et du remplissage des faces par une couleur. Les paramètres d'une section sont :

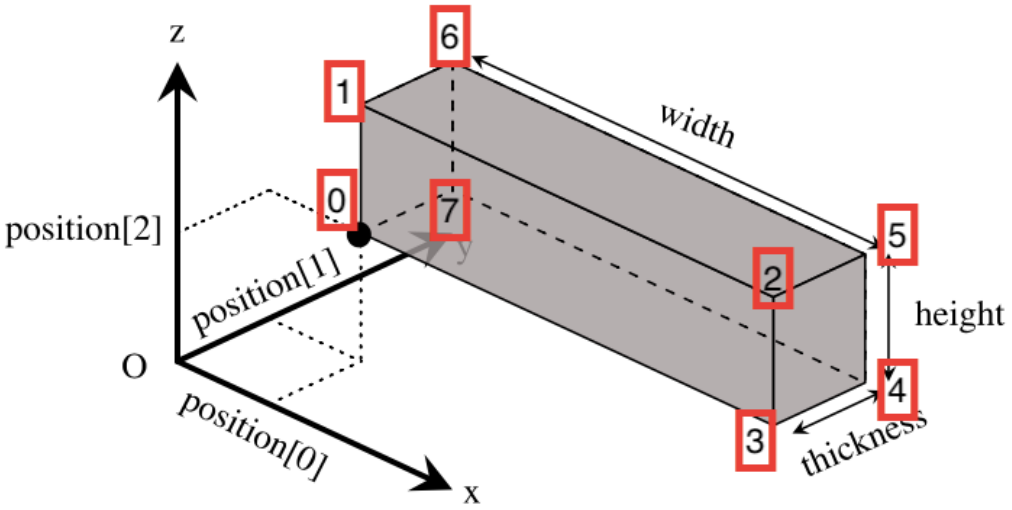
- La largeur dans des x dénommée width
- La hauteur dans l'axe des z dénommée height
- La profondeur dans l'axe des y dénommée thickness
- La position du coin inférieur gauche à l'aide de trois coordonnées stockées dans une liste position

Nous allons compléter le fichier section.py au fur et à mesure.

Question 2 a :

Nous allons écrire la méthode generate(self) dans la classe Section. Cette méthode crée les sommets et faces d'une section orientée selon l'axe x et dont le coin bas gauche de la face externe est en (0,0,0).

Chaque face sera représentée par les numéros de ses quatre sommets :



Nous avons le code suivant :

```
53      # Defines the vertices and faces
54      def generate(self):
55          self.vertices = [
56                  [0, 0, 0 ],
57                  [0, 0, self.parameters['height']],
58                  [self.parameters['width'], 0, self.parameters['height']],
59                  [self.parameters['width'], 0, 0],
60                  [self.parameters['width'], self.parameters['thickness'], 0],
61                  [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
62                  [0, self.parameters['thickness'], self.parameters['height']],
63                  [0, self.parameters['thickness'], 0]
64                  ]
65
66          self.faces = [
67                  [0, 3, 2, 1],
68                  [1, 2, 5, 6],
69                  [6, 5, 4, 7],
70                  [7, 4, 3, 0],
71                  [0, 7, 6, 1],
72                  [2, 3, 4, 5]
73                  ]
```

Explication :

Afin d'écrire ce programme, nous nous sommes aidés d'un schéma annoté, et nous avons trouvé les coordonnées de chaque sommet. Nous avons après cela représenté les faces grâce à ces quatre sommets.

Question 2 b :

Nous commençons par analyser la fonction Q2b(). Cette fonction fait appel à la fonction add() qui est dans Configuration afin de créer une Section avec les paramètres suivants :

- Position = [1,1,0]
- Width = 7
- Height = 2.6

L'instruction Configuration().add(section).display() suit le principe suivant : Nous allons chercher dans le fichier Configuration la fonction « add », nous lui faisons prendre comme argument une section définie précédemment. La fonction « display » permet d'afficher ce que nous venons de créer.

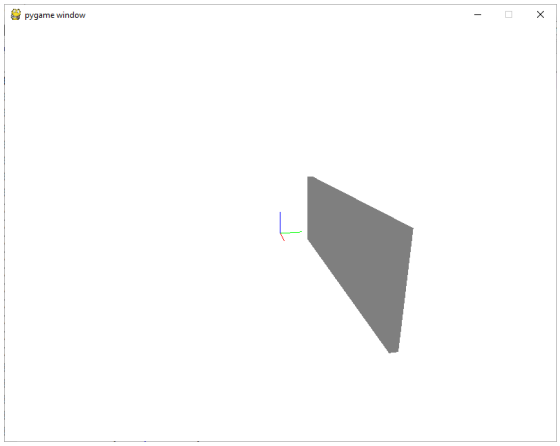
Nous allons maintenant écrire la méthode draw() pour la classe Section afin de tracer les faces de la section en gris :

```
90      # Draws the faces
91      def draw(self):
92          # A compléter en remplaçant pass par votre code
93
94          gl.glPushMatrix()
95          gl.glTranslatef(self.parameters['position'][1], self.parameters['position'][0], self.parameters['position'][2])
96
97          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
98
99          for i in self.faces :
100             gl.glBegin(gl.GL_QUADS)
101             gl.glColor3fv([0.5, 0.5, 0.5])
102             for j in i:
103                 gl.glVertex3fv(self.vertices[j])
104             gl.glEnd()
105
106         gl.glPopMatrix()
107
```

Explication :

Nous utilisons les fonctions données dans l'énoncé afin de sauvegarder et de récupérer une matrice de transformation dans une pile. Ces fonctions servent à afficher ou non les faces selon l'angle de vue. Afin d'afficher toutes les faces, nous commençons par parcourir les différentes faces et ensuite allons chercher dans vertice les paramètres nécessaires.

Nous obtenons le résultat suivant :



Question 2 c :

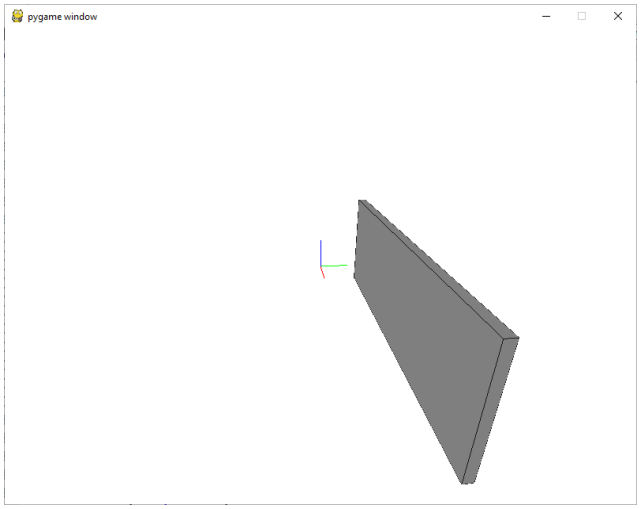
Nous allons maintenant dessiner les arêtes de la section. Nous allons utiliser le même principe que dans la question précédente mais il faudra changer la couleur et changer le mode par gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE).

On commence par écrire la méthode drawEdges() dans la classe Section :

```
85      # Draws the edges
86      def drawEdges(self):
87          # A compléter en remplaçant pass par votre code
88          gl.glPushMatrix()
89          gl.glTranslatef(self.parameters['position'][1], self.parameters['position'][0], self.parameters['position'][2])
90
91          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE) # on trace les lignes : GL_LINE
92
93          for i in self.faces :
94              gl.glBegin(gl.GL_QUADS)
95              gl.glColor3fv([0.1, 0.1, 0.1])
96              for j in i:
97                  gl.glVertex3fv(self.vertices[j])
98              gl.glEnd()
99
100         gl.glPopMatrix()
101
102     # Draws the faces
103     def draw(self):
104         # A compléter en remplaçant pass par votre code
105         self.drawEdges()
106         gl.glPushMatrix()
```

Nous avons ajouté la ligne 105 afin que la méthode drawEdges soit exécutée en premier lorsque le paramètre edges prend la valeur True.

Nous obtenons l'affichage suivant :



V. Création des murs :

Nous allons maintenant créer des murs. Le mur correspond à un objet contenant une liste d'éléments à tracer.

Question 3 a :

Nous commençons par analyser le fichier wall.py :

- Ligne 12 à 46 : Initialisation de la position, largeur, hauteur, orientation, épaisseur et couleur (comme dans la classe Section). Création d'une liste d'objets qui sera vide pour le moment (ligne 38). L'instruction self.parentSection (ligne 41 à 45) permet de créer la section principale (« pan du mur »). A la ligne 46, on ajoute la section principale à la liste des objets créés.
- Ligne 49 à 50 : Surcharge d'opérateur pour accéder aux paramètres du mur plus facilement.
- Ligne 53 à 55 : Surcharge d'opérateur pour modifier la valeur d'un paramètre plus facilement.
- Ligne 58 à 62 : Trouve l'emplacement où un objet peut être ajouté.
- Ligne 65 à 71 : Fonctions à compléter servant à ajouter l'objet voulu dans l'emplacement voulu et à dessiner les faces.

Il faut maintenant modifier la méthode draw de la classe Wall. Nous écrivons le code suivant :

```
70      # Draws the faces
71      def draw(self):
72          # A compléter en remplaçant pass par votre code
73          gl.glPushMatrix()
74          gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
75          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
76
77          for x in self.objects:
78              x.draw()
79
80          gl.glPopMatrix()
81
```

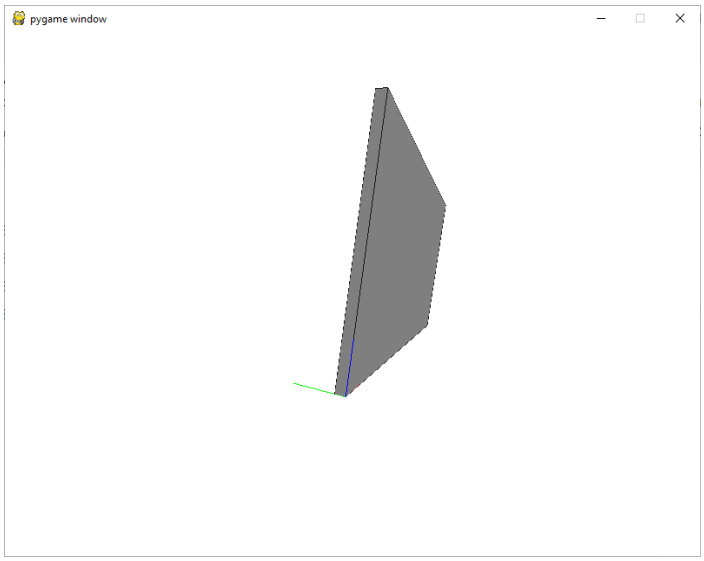
Explication :

Nous utilisons le même principe que dans la question 2c.

Nous écrivons maintenant des instructions dans le fichier main afin de créer et tracer un mur constitué d'une section parente :

```
40 def Q3a():
41     return Configuration().add(
42             Wall({'position' : [0, 0, 0], 'width' : 10, 'height' : 5})
43             )
44
```

Nous obtenons l'affichage suivant :



VI. Création d'une maison :

Nous allons maintenant créer une maison en créant quatre instances de mur. Chaque mur pourra contenir portes et fenêtres. La maison pourra être translatée et tournée.

Nous écrivons la méthode draw() de la classe House, nous avons repris celle écrite dans la question 3a :

```
42      # Draws the house
43      def draw(self):
44          # A compléter en remplaçant pass par votre code
45          gl.glPushMatrix()
46          gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
47          gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
48
49          for x in self.objects:
50              x.draw()
51
52          gl.glPopMatrix()
```

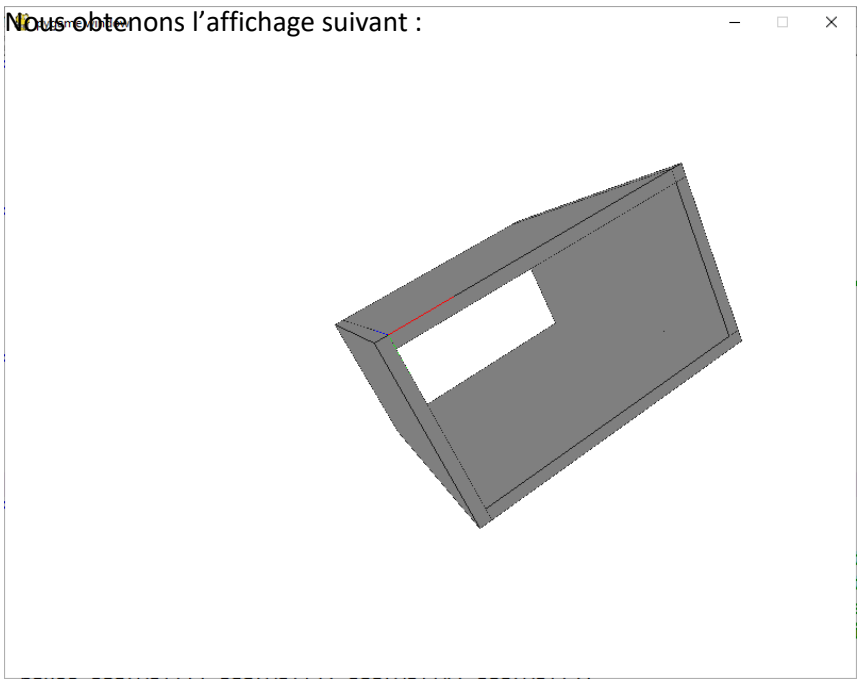
On modifie Q4a() dans le Main afin de créer une maison constituée de 4 murs :

```
45|
46 def Q4a():
47      # Ecriture en utilisant des variables : A compléter
48      wall1 = Wall({'position' : [0, 0, 0], 'width' : 5, 'height' : 5, 'orientation' : 0})
49      wall2 = Wall({'position' : [0, 0, 0], 'width' : 3, 'height' : 5, 'orientation' : 90})
50      wall3 = Wall({'position' : [2.8, 0, 0], 'width' : 5, 'height' : 5, 'orientation' : 0})
51      wall4 = Wall({'position' : [-5, 0, 0], 'width' : 2.8, 'height' : 5, 'orientation' : 90})
52      house = House({'position': [-3, 1, 0], 'orientation':0})
53      house.add(wall1).add(wall3).add(wall4).add(wall2)
54      return Configuration().add(house)
55
```

Explication :

Nous avons fait un dessin afin de comprendre les coordonnées que chaque mur devait avoir. Tous nos murs sont regroupés dans House. Nous avons parfois dû mettre 2.8 comme largeur afin de compenser la largeur du mur de 0,2.

Nous obtenons l'affichage suivant :



VII. Création d'ouvertures :

Le but de cette partie est de créer des ouvertures dans un mur afin d'y mettre des portes ou des fenêtres. Afin de créer une ouverture, nous allons devoir transformer notre section initiale en plusieurs sections. Comme l'explique le schéma suivant :

Question 5 a :

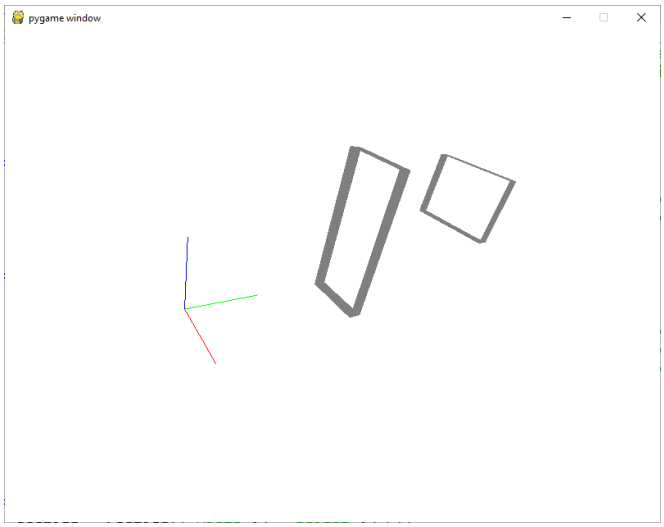
Nous avons écrit la méthode draw() comme vu précédemment. Nous complétons la classe Opening de la manière suivante :

```
47      # Defines the vertices and faces
48      def generate(self):
49          self.vertices = [
50                  [0, 0, 0 ],
51                  [0, 0, self.parameters['height']],
52                  [self.parameters['width'], 0, self.parameters['height']],
53                  [self.parameters['width'], 0, 0],
54                  [self.parameters['width'], self.parameters['thickness'], 0],
55                  [self.parameters['width'], self.parameters['thickness'], self.parameters['height']],
56                  [0, self.parameters['thickness'], self.parameters['height']],
57                  [0, self.parameters['thickness'], 0]
58                  ]
59          self.faces = [
60                  [1, 2, 5, 6],
61                  [7, 4, 3, 0],
62                  [0, 7, 6, 1],
63                  [2, 3, 4, 5]
64                  ]
65
```

Explication :

Nous réutilisons le même code que pour la section question 2a, mais nous supprimons les faces 0321 et 6547.

Nous obtenons l'affichage suivant :



Question 5 b :

Nous modifions la classe Section en ajoutant la méthode canCreateOpening(self, x). Cette méthode renvoie True si une ouverture, représentée par x, peut être ajoutée dans la section représentée par self. Sinon, elle renvoie False.

```
75    # Checks if the opening can be created for the object x
76    def canCreateOpening(self, x):
77        # A compléter en remplaçant pass par votre code
78        if x.parameters['width']+x.parameters['position'][0] <= self.parameters['position'][0] + self.parameters['width']:
79            if x.parameters['height'] + x.parameters['position'][2] <= self.parameters['position'][2] + self.parameters['height']:
80                return True
81        return False
```

Explication :

On commence par vérifier si l'ouverture ne dépasse pas du mur dans la largeur. On vérifie par la suite que l'ouverture n'est pas plus haute que le mur. Si l'ouverture est bien comprise à l'intérieur de la section, on renvoie True. Sinon, on renvoie False.

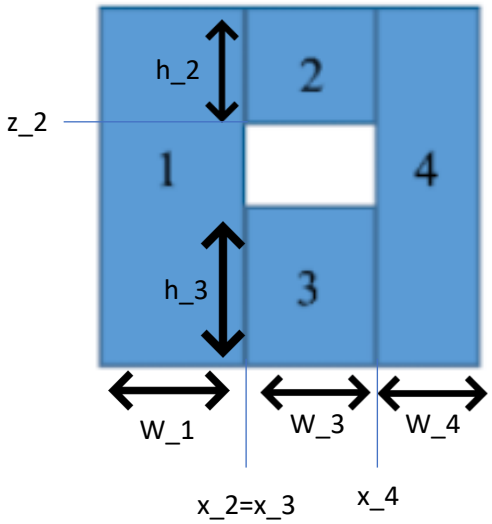
On obtient le résultat suivant :

```
In [29]: runfile('G:/tp3-representa
True
True
False
```

La dernière ouverture démarre bien à l'intérieur de la section mais elle finit en-dehors (1 + 1.7 > 2.6). C'est pour cela que l'on a False en dernier résultat.

Question 5 c :

Nous voulons maintenant décomposer notre section de base en plusieurs sections afin de créer l'ouverture. Comme le montre le schéma suivant :



Nous commençons par écrire la méthode createNewSections(self, x). Nous n'avons pas réussi à les aligner correctement, et les sections 1 et 4 sont mal créées.

Nous avons le code suivant :

```
83      # Creates the new sections for the object x
84      def createNewSections(self, x):
85          # A compléter en remplaçant pass par votre code
86          if self.canCreateOpening(x):
87              L = []
88              h_0 = self.parameters['height']
89              w_1 = self.parameters['width'] - x.parameters['position'][0]
90              x_2 = x.parameters['position'][0]
91              z_2 = x.parameters['position'][2] + x.parameters['height']
92              w_2 = x.parameters['width']
93              h_2 = self.parameters['height'] - z_2
94              x_3 = x_2
95              w_3 = w_2
96              h_3 = x.parameters['position'][2]
97              x_4 = x.parameters['width'] + x.parameters['position'][0]
98              w_4 = self.parameters['width'] - x.parameters['position'][0] - x.parameters['width']
99              if w_1 != 0:
100                 section1 = Section({'position' : [0, 0, 0], 'width' : w_1, 'height' : h_0})
101                 L.append(section1)
102             if h_2 != 0 :
103                 section2 = Section({'position' : [x_2, 0, z_2], 'width' : w_2, 'height' : h_2})
104                 L.append(section2)
105             if h_3 != 0 :
106                 section3 = Section({'position' : [x_3, 0, 0], 'width' : w_3, 'height' : h_3})
107                 L.append(section3)
108             if w_4 != 0 :
109                 section4 = Section({'position' : [x_4, 0, 0], 'width' : w_4, 'height' : h_0})
110                 L.append(section4)
111             return L
```

Nous obtenons le résultat suivant (qui n'est pas le résultat attendu) :

Conclusion :

Malheureusement, nous n'avons pas réussi à terminer le TP à temps, et la dernière question que nous avons effectuée n'a pas fonctionné. Mais il était très intéressant de voir comment créer et manipuler des objets en 3D, d'autant plus que c'est une fonctionnalité que nous côtoyons tous les jours sans réaliser le travail que cela demande.