

# TP3 (les deux séances) :

## Représentation visuelle des objets

### Introduction

Pour ce TP3, nous devons représenter des objets en 3D dans une fenêtre graphique grâce à OpenGL et pygame. Avec la séance 2, nous devons terminer le TP.

## I. Préparation

### Utilisation de pygame

#### Question 1

On crée une fenêtre en utilisant pygame avec une hauteur de 200 et une largeur de 300. Mais elle disparaît immédiatement.

#### Question 2

Dans un premier temps, on crée la fenêtre.

Ensuite, on crée une boucle d'exécution while.

Dans cette boucle d'exécution, on récupère tous les événements dans un tableau avec `pygame.event.get()`.

On vérifie si l'un des événements correspond à `pygame.KEYDOWN`. Cet événement est activé lorsqu'une touche quelconque du clavier est enfoncée.

Si une touche est enfoncée, on arrête la boucle d'exécution et on ferme la fenêtre.

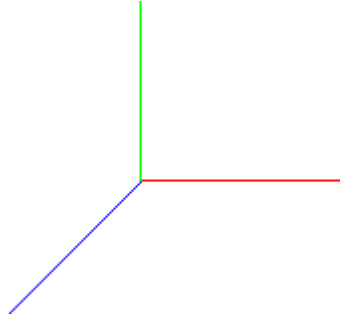
### Utilisation PyOpengl

## Question 1

On remarque que la couleur du fond est définie par `gl.ClearColor`.

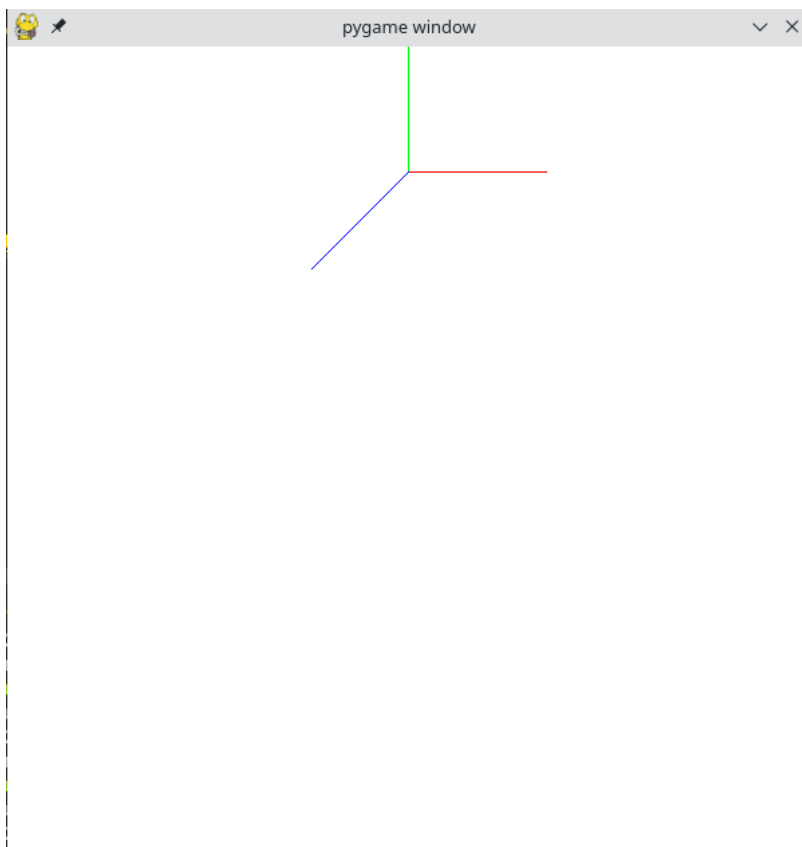
## Question 2

On obtient ce genre d'image :



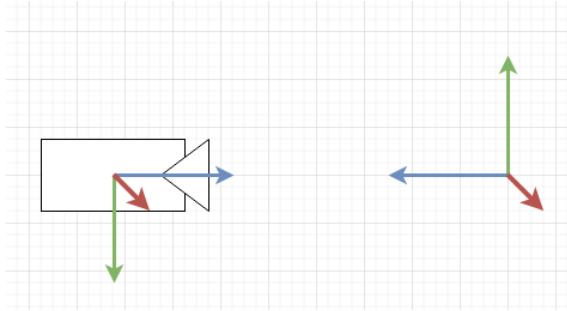
1.

Lors de la translation, on obtient :

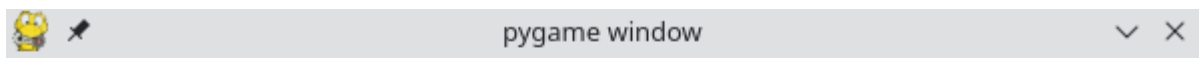


Ce résultat nous apprend beaucoup de choses sur le fonctionnement de la Caméra et les axes que la caméra suit.

En effet, si on veut retrouver le comportement de la caméra ci dessus, il faut que les axes soit comme cela :

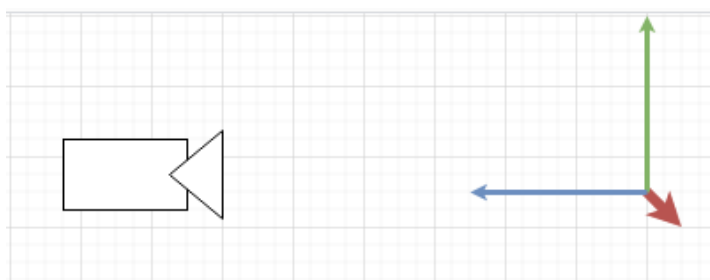


Voici le résultat pour la rotation de  $90^\circ$  :



Ce résultat est logique, on effectue une rotation sur la caméra de  $90^\circ$ , lorsque l'on fait cela, c'est normal que l'on ne voit plus l'objet, on regarde le ciel :

Avant rotation :



Après rotation :



## Découverte de l'environnement de travail de TP

### Question 1

#### a. Question 1)a

Le fichier main.py récupère toutes les classes du dossier "src" afin de les utiliser. Les différentes configurations sont contenues dans des fonctions notées de Q1a à Q6. Elles sont appelées à la toute fin du fichier, dans la fonction main(). La fonction main est ensuite appelée lors de l'exécution du fichier.

Le fichier Configuration.py contient la classe "Configuration". Cette classe contient toutes les informations liées au GUI généré grâce à pygame. Elle contient également toutes les informations relatives à l'espace dans lequel les objets sont créés grâce à OpenGL.

Le constructeur permet de définir un bon nombre d'options :

- L'option d'afficher les axes ou non
- L'option de couleur pour chaque axe
- et enfin l'option de la position de la caméra par rapport au [0,0,0] de l'espace.

Durant la construction de l'objet, les éléments relatifs à Pygame et OpenGL sont initialisés.

initializePygame crée une fenêtre de 800 de largeur et 600 de hauteur et la configure pour fonctionner avec OpenGL.

initializeOpenGL définit la couleur utilisée en background et d'autres paramètres utilisés dans l'espace.

La classe contient également un getter(getParameter) et un setter(setParameter). Le setter a un comportement particulier par rapport à la position de la caméra dans l'espace.

La méthode add permet d'ajouter un élément à la collection d'objets à afficher à l'écran.

La méthode draw permet, comme son nom l'indique, de dessiner les objets contenus dans la collection.

#### b. Question 1)b

La modification de code : permet d'approcher la caméra de l'objet, car la position de base est -1. On met également la couleur de l'axe X en Jaune.

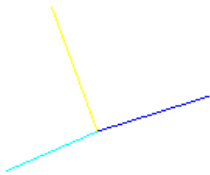
```
Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display()
```

Pour ce morceau de code :

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

display()
```

Les paramètres passés permettent de mettre la couleur de l'axe x en jaune et l'axe y en cyan



Ce chaînage d'appel est possible car la méthode setParameter retourne self.

Il y a un traitement particulier pour le paramètre : 'screenPosition' car il doit être appliqué à la matrice de translation à chaque modification.

### c. Question 1)c

Je vois ce problème comme une simple rotation : on retourne la caméra de -90 degré autour de l'axe x. Cela permet de passer l'axe de cette position :



à cette position :



## II. Mise en place des interactions avec l'utilisateur avec Pygame

### Question 1)d

Nous devons faire en sorte qu'avec les touches "Page up" et "Page down", on puisse grossir ou réduire l'affichage.

Pour cela, nous avons rajouté ces lignes dans la fonction processKeyDownEvent :

```
if self.event.key==pygame.K_PAGEUP:
    gl.glScalef(1.1,1.1,1.1)
if self.event.key==pygame.K_PAGEDOWN:
    gl.glScalef(1/1.1,1/1.1,1/1.1)
```

Si l'événement quand j'appuie sur une clé correspond à la touche "Page up", alors j'utilise une fonction d'OpenGL, glScalef, qui va me permettre de faire un zoom selon les données de l'énoncé.

On fait ensuite la même chose pour "Page Down", en changeant simplement les valeurs pour avoir un effet de "de-zoom".

### Question 1)e

On veut la même chose que dans la question précédente, sauf que cette fois, cela va dépendre de la molette de la souris.

On obtient ce code :

```
# 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)
```

Si on utilise la molette de la souris dans le sens "avant", alors on va effectuer un zoom. Et si on utilise la molette dans le sens arrière, on va "de-zoom".

### Question 1)f

Ici, on veut faire une rotation autour de x composée avec une rotation autour de z lorsque le bouton gauche est enfoncé et que la souris est déplacée et une translation selon l'axe x composée à une translation selon l'axe z lorsque le bouton droit est enfoncé et que la souris est déplacée.

Pour la rotation, voici le code :

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):
    if pygame.mouse.get_pressed()[0]:
        gl.glRotatef(self.event.rel[0],0,0,1)
        gl.glRotatef(self.event.rel[1],1,0,0)
```

On a une condition pour vérifier si le bouton gauche est bien enfoncé. Puis, on utilise la fonction `glRotatef(angle,x,y,z)` pour effectuer la rotation en mettant l'angle à 1° (pour être plus précis). x et y sont à 1 et z va dépendre du déplacement horizontal de la souris

Pour la translation, voici le code :

```
if pygame.mouse.get_pressed()[2]:
    gl.glTranslatef(self.event.rel[0]/25,0,-self.event.rel[1]/25)
```

On vérifie si on presse bien le bouton droit, puis on va effectuer une translation avec la fonction `glTranslatef` en utilisant le déplacement horizontal (x) et le déplacement vertical (y, qui sera ici z) de la souris. y, lui, sera à 0 car nous ne voulons pas faire de translation avec cet axe.

Voici aussi un screenshot des tests pour cette partie (qui sont tous validés) :

```
C:\Users\etien\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\Scripts\python.exe "D:\deep_learning\PyCharm 2021.2.3\plugins\python\helpers\pycharm\jb_unittest_runner.py" --target test_interaction_utilisateur.TestQuestionID
Testing started at 20:13 ...
Launching unittests with arguments python -m unittest test_interaction_utilisateur.TestQuestionID in D:\test\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\tests

pygame 2.0.0 (SDL 2.0.12, python 3.9.9)
Hello from the pygame community. https://www.pygame.org/contribute.html

Ran 3 tests in 0.012s

OK

Process finished with exit code 0
```

```
✓ Tests passed: 3 of 3 tests - 3ms
C:\Users\etien\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\Scripts\python.exe "D:\deep_learning\PyCharm 2021.2.3\plugins\python\helpers\pycharm\jb_unittest_runner.py" --target test_interaction_utilisateur.TestQuestionIF
Testing started at 20:14 ...
Launching unittests with arguments python -m unittest test_interaction_utilisateur.TestQuestionIF in D:\test\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\tests

pygame 2.0.0 (SDL 2.0.12, python 3.9.9)
Hello from the pygame community. https://www.pygame.org/contribute.html

Ran 3 tests in 0.000s

OK
```

```
✓ Tests passed: 1 of 1 test - 4ms
C:\Users\etien\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\Scripts\python.exe "D:\deep_learning\PyCharm 2021.2.3\plugins\python\helpers\pycharm\jb_unittest_runner.py" --target test_interaction_utilisateur.TestQuestionIF
Testing started at 20:15 ...
Launching unittests with arguments python -m unittest test_interaction_utilisateur.TestQuestionIF in D:\test\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3\tests

pygame 2.0.0 (SDL 2.0.12, python 3.9.9)
Hello from the pygame community. https://www.pygame.org/contribute.html

Ran 1 test in 0.005s

OK
```

### III. Création d'une section

#### Question 2)a

Dans un premier temps, il faut déterminer comment les faces seront construites, c'est-à-dire qu'il faut finir le tableau "self.faces".

```
self.faces = [
    [0, 3, 2, 1], # la face en face de l'utilisateur
    [1, 2, 6, 5], # la face en haut
    [5, 6, 7, 4],
    [0, 3, 7, 4],
    [2, 3, 7, 6],
    [1, 0, 4, 5]
    # définir ici les faces
]
```

Ensuite, il faut s'occuper de la création des vertices et surtout de leur positionnement. Ici 2 approches sont possibles : soit on implémente les coordonnées de la section directement dans la méthode "generate" comme suit :

```
# Defines the vertices and faces
def generate(self):
    self.vertices = [ # A revoir, il manque l'utilisation de self.position
        self.parameters['position'],
        [self.parameters['position'][0], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
        [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]+self.parameters['height']],
        [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1], self.parameters['position'][2]],
        [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]],
        [self.parameters['position'][0], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
        [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]+self.parameters['height']],
        [self.parameters['position'][0]+self.parameters['width'], self.parameters['position'][1]+self.parameters['thickness'], self.parameters['position'][2]]
    ]
    # Définir ici les sommets
    self.faces = [
        [0, 3, 2, 1], # la face en face de l'utilisateur
        [1, 2, 6, 5], # la face en haut
        [5, 6, 7, 4],
        [0, 3, 7, 4],
        [2, 3, 7, 6],
        [1, 0, 4, 5]
        # définir ici les faces
    ]
```

L'autre approche ne demande l'addition que 3 lignes de code à l'étape du dessin de la section.

On utilise la méthode `gl.glTranslatef` afin de placer la section qui vient d'être créée à la bonne position :

```
gl.glTranslatef(
    self.parameters['position'][0],
    self.parameters['position'][1],
    self.parameters['position'][2]
)
```

On oublie pas de push une matrice et de la supprimé juste après.

La deuxième méthode a été retenue.



## Question 2)b

Ce morceau de code affiche une face normal à l'axe y allant de  $x=0$  à  $x=1$  et de  $z=0$  à  $z=1$

Afin de créer toutes les faces de la section, il suffit de boucler sur `self.faces` et d'associer les différents sommets entre eux.

```
# Draws the faces
def draw(self):
    gl.glPushMatrix()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    gl.glTranslatef(
        self.parameters['position'][0],
        self.parameters['position'][1],
        self.parameters['position'][2]
    )
    for f in self.faces:
        gl.glBegin(gl.GL_QUADS)
        gl.glColor3fv(self.parameters['color'])
        for sommet in f :
            gl.glVertex3fv(self.vertices[sommet])
        gl.glEnd()
    gl.glPopMatrix()
```

## Question 2)c

On a rencontré un problème : les arêtes ne sont pas stockées dans la classe, elles doivent être calculées.

Afin de trouver toutes les arêtes, nous avons 2 approches, soit on marque toutes les possibilités d'arêtes en dur, ça fonctionne et ça ne demande aucun calcul, ou alors, on trouve toutes les arêtes de manière algorithmique, le codes est donc plus court et lisible. Nous avons opté pour la seconde option.

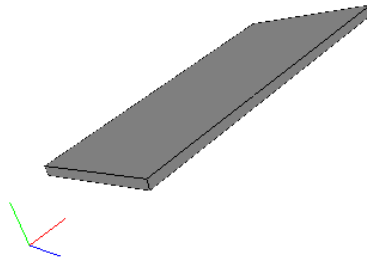
```
# Draws the edges
def drawEdges(self):
    lines = []
    for x in self.faces :
        for i in range(len(x)):
            lines.append([x[i], x[i+1 if i+1 < len(x) else 0]])

    for l in lines :
        gl.glBegin(gl.GL_LINES)
        gl.glColor3fv([0,0,0])
        for sommet in l :
            gl.glVertex3fv(self.vertices[sommet])
        gl.glEnd()
```

Nous obtenons la liste des sommets pour les lignes en associant les sommets présents dans les faces 2 à 2. Cela revient à lire toutes les lignes dont les faces sont composées.

Ensuite, il suffit de parcourir la liste des lignes et de les tracer sur l'écran grâce à OpenGL.

Nous obtenons ce résultat :



L'option "edges" rentrées par l'utilisateur dans la section est prise en compte dans la fonction draw() de la classe "Section".

Voici les tests obtenus pour la partie section :

```
C:\Users\Damien\Desktop\temp\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3>C:/Users/Damien/Desktop/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_section.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
.....
-----
Ran 9 tests in 0.002s

OK
```

## IV. Création des murs

### Question 3)a

Le constructeur de Wall a une particularité par rapport aux autres : lors de la création d'un mur, une section est également créée. En effet, un mur est défini par une section de base qui correspond aux dimensions du mur. On remarque également qu'un mur peut être composé de plusieurs objets, ainsi, dans la méthode draw, il suffit de dessiner tous les objets du mur en appliquant la rotation désirée.

Pour la méthode draw, ça a été plus compliqué : nous avons essayé de gérer la rotation des objets de manière mathématique, sans utiliser les fonctions de OpenGL. Cela n'a pas fonctionné. Nous avons adopté une nouvelle approche utilisant les méthodes glPushMatrix et glPopMatrix.

Ce que l'on fait c'est :

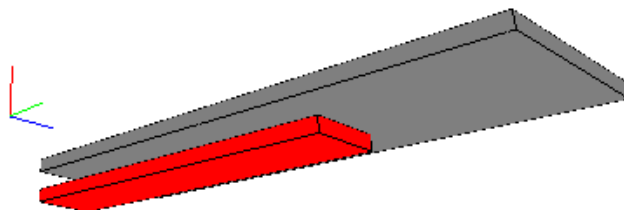
- On crée une matrice,
- On applique la rotation,
- On dessine les objets ,
- On enlève la matrice afin de ne plus travailler sur elle.

Afin de tester la méthode, On a ajouté ce code dans la fonction Q3a :

```
4 def Q3a():
5     return Configuration().add(
6         Wall({'position': [1, 1, 0], 'width':7, 'height':2.6, 'edges': True, 'orientation':90}).\
7         add(Section({'position':[1, 1.5, 0], 'width':5, 'height':1, 'edges': True, 'color':[1,0,0]}))
8     )
```

Nous devons obtenir 2 sections : une grise et une rouge qui auront une rotation de 90 degrés autour de l'axe z.

Voici le résultat du test :



Nous avons ensuite enlevé la 2ème section (la barre rouge) afin de faire passer les tests unitaires :

```
B8/python.exe c:/Users/Damien/Desktop/temp/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_wall.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
...
-----Ran 3 tests in 0.001s
OK
```

## V. Création d'une maison

Pour créer la maison, on a commencé par la partie la plus simple : appliquer une translation pour positionner la maison comme elle devrait être et appliquer la rotation correspondant à l'orientation de la maison.

```
gl.glPushMatrix()  
gl.glRotatef(self.parameters['orientation'], 0, 0, 1)  
gl.glTranslatef(self.parameters['position'][0],  
self.parameters['position'][1], self.parameters['position'][2])  
... #Affichage objets  
gl.glPopMatrix()
```

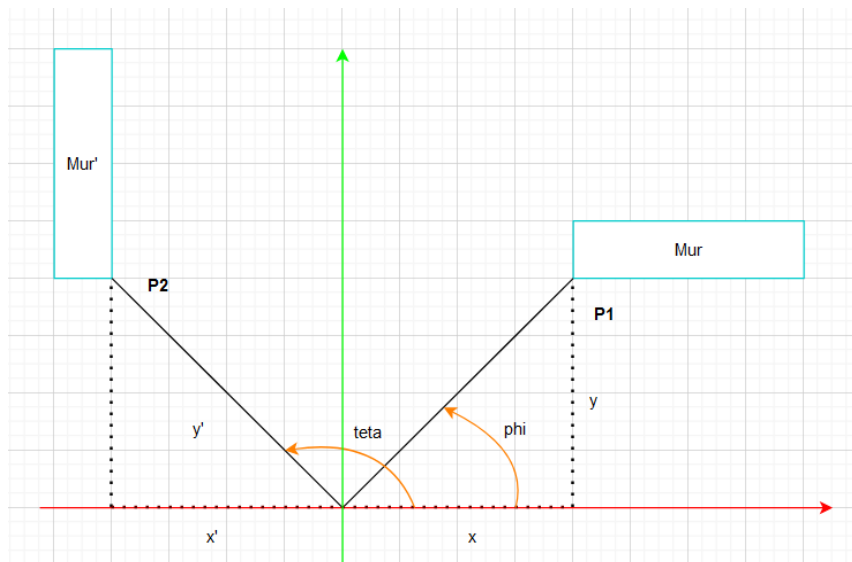
Dans un second temps, il faut afficher tous les murs de la maison.  
Cette partie a posé beaucoup de problèmes pour plusieurs raisons :

Le premier problème rencontré est la rotation des murs. En effet, lorsque l'on fait la rotation d'un mur grâce à la Classe Wall, la rotation est faite par rapport à l'axe Z, hors nous voulons faire une rotation sur place afin de placer tous les murs au bon endroit.

Après avoir fait des recherches nous avons déterminé qu'une solution possible était de faire une correction des mouvement fait par les murs lors de leur création.

On veut faire en sorte que la position du mur en coordonnées x et y ne change pas lors de la rotation. Ainsi il faut tradater le mur de manière à ce qu'il reste sur place.

On peut représenter cette situation sous forme d'un schéma.



Sur le schéma ci-dessus, nous pouvons voir que Mur' est l'image de Mur après une rotation de 90 degrés. La solution au problème serait de déterminer la translation à faire pour passer de P2 à P1 et de l'appliquer.

On trouve cette translation grâce à cette suite d'instruction :

```
x=o.parameters['position'][0]
y=o.parameters['position'][1]

phi = arctan(divide(y,x))*180/pi
teta = o.parameters['orientation']+phi

x_ = sqrt(pow(x,2)+ pow(y, 2)) * cos(teta*pi/180)
y_ = sqrt(pow(x,2)+ pow(y, 2)) * sin(teta*pi/180)

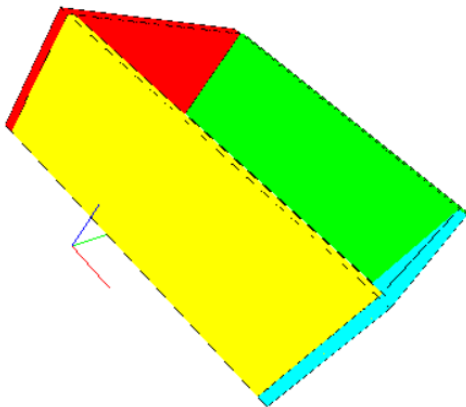
gl.glPushMatrix()
gl.glTranslatef(float(x-x_), float(y-y_),0)
o.draw()
gl.glPopMatrix()
```

Le second problème rencontré est directement lié à la solution trouvée juste au-dessus : lorsque x et y sont à 0, la fonction numpy “divide” n’est pas capable de renvoyer inf pour l’arctangente.

Ce problème est facilement résolu :

```
phi = arctan(divide(y,x))*180/pi if not(x==0) else 90
```

Ensuite nous avons obtenu ce résultat :



Voici les test unitaires pour cette partie :

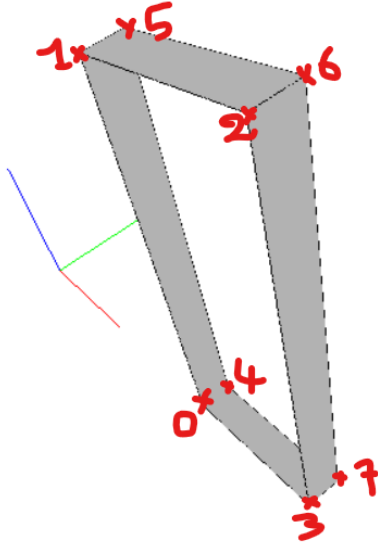
```
tp3/tests/test_house.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
...
-----
Ran 3 tests in 0.002s

OK
```

## VI. Création d'ouverture

### Question 5)a

A partir de cette figure :



Il devient assez simple de déduire la composition des faces.

Tout comme pour la création d'une section, la création des points est faite par rapport à la position d'origine (0,0,0) à laquelle on applique une matrice de translation.

### Question 5)b

Pour cette question, une simple suite de if est suffisante pour obtenir le résultat voulu.

On remarque que le dernier résultat du test est False.

Cela est justifié car l'ouverture que l'on veut créer sera trop haute par rapport à la section.

En effet, la position de cette ouverture en hauteur est de 1.7 m et la hauteur de l'ouverture est de 1m, hors la hauteur maximum de la section est de  $2.6\text{m} < 1.7+1$ .

Donc il n'est pas possible de mettre cette ouverture dans la section.

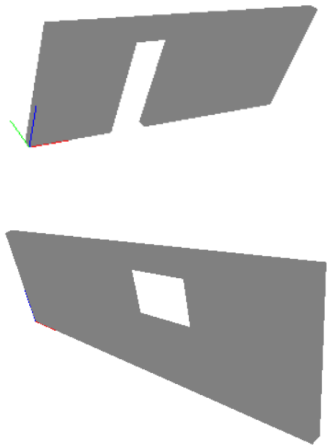
### Question 5)c

Pour cette partie de l'exercice, dans un premier temps, on vérifie que l'ouverture est possible dans cette section :

```
if self.canCreateOpening(x):  
    ...  
else:  
    return []
```

Il faut ensuite créer les 4 sections et ensuite déterminer si elles devraient être utilisées dans la construction du mur. Pour la section 1 et 4, il faut vérifier si la largeur n'est pas nulle et pour les section 2 et 3, il faut vérifier si la hauteur n'est pas nulle.

Voici le résultat :



### Question 5)d

La fonction `enumerate` permet de retourner un tuple de valeur : la première valeur est l'index de l'objet dans le tableau "objects", et la seconde valeur est l'objet en lui-même.

Pour ce code :

```
wall = Wall({'width':7, 'height':2.6,})
opening1 = Opening({'position': [2, 0, 0], 'width':0.9, 'height':2.15,
'thickness':0.2, 'color':[0.7, 0.7, 0.7]})
section = wall.findSection(opening1)
```

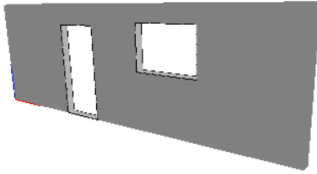
`section[0]` contient 0, car l'opening présent ici n'est possible que dans la section parente.  
`section[1]` contient la section parente en elle-même.

Lors de la création du mur avec 2 ouvertures, nous avons trouvé un problème de taille :

Les méthodes "canCreateOpening" et "createNewSections", utilisent toutes les 2 les coordonnées de l'ouverture relative à la section en question et non pas les coordonnées absolue. Cela nous a mis en difficulté pendant un certain temps.

Pour résoudre ce problème, la méthode "canCreateOpening" a été modifiée pour fonctionner dans un contexte plus large et la méthode "createNewSections" ne présente plus de vérification, car cela est fait dans la méthode "findSection".

Voici le résultat :



Voici le résultat des tests unitaires pour cette partie :

```
C:\Users\Damien\Documents\Polytech\2021-2022\INF0501_Numerisation\TPs\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3>c:/Users/Damien/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Damien/Documents/Polytech/2021-2022/INF0501_Numerisation/TPs/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_opening.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
.....
Ran 9 tests in 0.001s
OK

C:\Users\Damien\Documents\Polytech\2021-2022\INF0501_Numerisation\TPs\tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3>c:/Users/Damien/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Damien/Documents/Polytech/2021-2022/INF0501_Numerisation/TPs/tp3-representation-visuelle-d-objets-malacarne_dasseux_tp3/tests/test_house.py
pygame 2.1.0 (SDL 2.0.16, Python 3.8.6)
Hello from the pygame community. https://www.pygame.org/contribute.html
...
Ran 3 tests in 0.001s
```

## VII. Pour finir ...

### Question 6)

On remarque que dans les classes “Window” et “Door”, tout est déjà défini, de la largeur et la hauteur d'une porte jusqu'à la couleur des fenêtres Il suffit donc de donner les coordonnées en paramètre afin d'obtenir une porte ou une fenêtre correctement conçue.. Ainsi, pour réaliser un mur avec porte et fenêtre, il suffit d'écrire :

```
def Q6():
    wall = Wall({"position": [0, 0, 0], 'width': 7, 'height': 2.6, "edges": True})

    door = Door({'position': [2, 0.1, 0]})
    window = Window({'position': [4, 0.1, 1.25]})

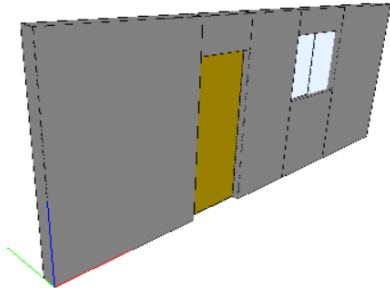
    wall.add(door)
    wall.add(window)

    configuration = Configuration()
    configuration.add(wall)

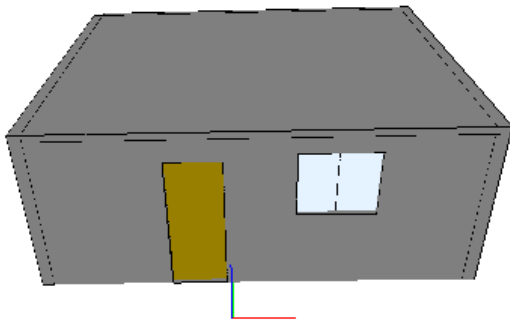
    return configuration
```



Voici le rendu du premier mur de la maison :



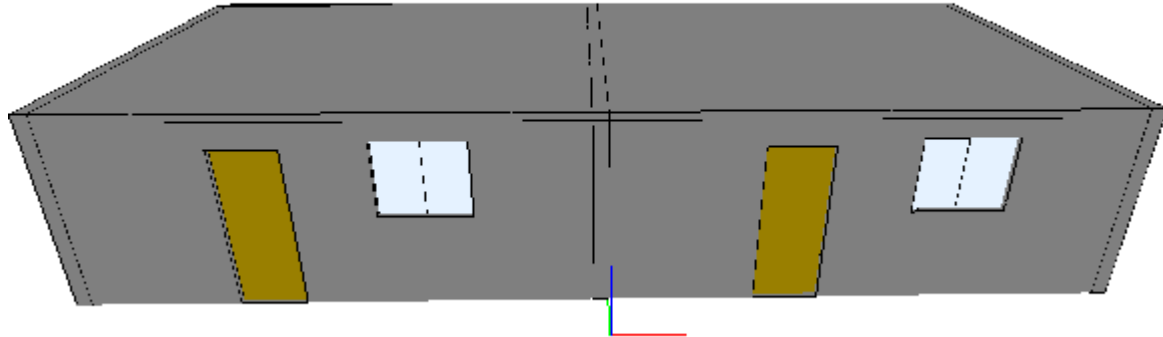
Ensuite, il faut ajouter les autres murs :



### Question 7)

Pour faire 2 maisons accolées, il suffit de dupliquer la maison précédemment créée puis de changer ses coordonnées.

Pour l'addition de la fenêtre dans la porte, nous n'avons pas trouvé comment le faire  
On obtient cela:



## Conclusion :

Suite au deux séances de TP , nous avons terminé toutes les questions du TP, excepté une partie de la question bonus. Tous les tests ont été effectués et validés. Ce TP nous aura permis de découvrir OpenGL, que nous ne connaissions que de nom. Il a été aussi intéressant de manipuler des objets 3D en python, sans utiliser des moteurs 3D comme Unity ou UnrealEngine.