

## Rapport de TP3 (séance 1 et 2) – Représentation visuelle d'objets

### I. Présentation du TP

Le but de ce TP est d'afficher des images en 3 dimensions.

### II. Préparation du TP

#### Utilisation de Pygame

##### 1. Question (1)

Nous avons implémenté le code fourni dans le sujet.

Le première ligne sert à importer la librairie Pygame, la seconde permet d'initialiser tout ce qui a été importé et de signifier que l'on va utiliser la librairie. Avec la troisième ligne on crée une fenêtre graphique (écran), d'une taille 300\*200. Cette dernière est pour le moment vide. Enfin, la dernière ligne permet de signifier que nous avons fini de travailler avec la librairie et de quitter la fenêtre que nous avons créé.

En effet, lorsque l'on exécute le programme, une fenêtre s'ouvre et se ferme presque instantanément.

##### 2. Question (2)

Nous avons implémenté le nouveau code.

Cette fois-ci une fenêtre s'ouvre mais ne se ferme que lorsque l'on appuie sur une touche du clavier.

Ceci est possible grâce à la boucle while : tant qu'il n'y a pas d'événement de type : pygame.KEYDOWN (appui sur une touche du clavier), la fenêtre reste ouverte.

#### Utilisation de Pyopengl pour représenter des objets 3D

##### 1. Question (1)

Nous avons exécuté le code.

Une fenêtre blanche s'ouvre. Elle ne se ferme plus quand on appuie sur une touche du clavier, mais quand on clique sur la croix avec la souris.

On a rajouté la fonction gluPerspective. Cela ne change rien au programme pour le moment.

##### 2. Question (2)

Nous avons complété notre code avec ce que fourni le sujet.

Nous obtenons un axe noir dans la fenêtre lorsque nous exécutons le programme. Nous apportons des modifications pour faire apparaître les 3 axes de 3 couleurs différentes. L'origine de nos axes est en (0, 0, -2).

##### 3. Question (3)

Nous avons ensuite ajouté les fonctions glTranslatef et glRotatef.

Avec ces fonctions, nous obtenons une nouvelle fenêtre où les axes sont translatés et/ou tournés par rapport à la fenêtre originale.

### III. Découverte de l'environnement du TP

##### 1. Question (1.a)

La ligne de commande donnée permet de faire tourner les axes créés précédemment selon l'axe z lorsque l'on appuie sur une touche (en l'occurrence la touche z pour tourner dans un sens et shift + z pour tourner dans l'autre sens). Nous pouvons aussi faire disparaître les axes en appuyant sur la touche a

C'est la fonction processKeyDownEvent(self) qui permet de définir un événement contenu dans la librairie pygame d'effectuer une action. Ici par exemple, l'événement event.key est un appui de touche de clavier.

## 2. Question (1.b)

La ligne de code proposée dans le sujet permet deux choses. Premièrement elle permet de réduire la taille de la figure dans la fenêtre grâce au 'screenPosition'. Ensuite elle permet de colorer l'axe de x (dans le cas présent en mélangeant du rouge et du vert, ce qui donne du jaune).

Il est possible de chaîner les méthodes 'setParameter' et 'display' car la méthode 'setParameter' ne fait que modifier les paramètres de figure. Paramètres qui ont de toute façon déjà une valeur par défaut et qui sont nécessaires pour l'afficher la figure à l'écran. La méthode 'setParameter' n'influence pas le fonctionnement de 'display' (à ceci près qu'elle affiche une figure différente en fonction des paramètres).

Dans la méthode setParameter, un traitement particulier est donné à 'screenPosition'. Ce traitement est nécessaire car la méthode fonctionne avec un dictionnaire. Lorsque l'on rentre un paramètre, cela revient à créer une clé et à lui associer la valeur que l'on souhaite pour ce paramètre. En ce qui concerne 'screenPosition', lorsque l'on rentre une nouvelle valeur, on ne tient pas compte de valeur précédente et on souhaite repartir de zéro pour effectuer la modification. On a donc besoin d'initialiser la position de la figure avant d'appliquer les nouveaux changements, d'où l'utilisation de la méthode 'initializeTransformationMatrix'.

## 3. Question (1.c)

On remarque que pour que l'axe z soit vertical et l'axe x horizontal, il suffit juste de « placer l'axe z à la place de l'axe x », c'est-à-dire faire une rotation de 90° de l'ensemble des axes selon l'axe y.

```
# Initializes the tranformation matrix
def initializeTransformationMatrix(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'])
    gl.glRotatef(-90, 1,0,0)

# Getter
```

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

### 1. Question (1.d)

On utilise la même méthode que celle que l'on voit dans l'exemple donné avec les touches z et a. On crée une condition if qui est validée que lorsque l'événement « appui sur la touche K\_PAGEUP » (respectivement K\_PAGEDOWN) est validé. On change toutefois l'action à réaliser qui n'est plus une translation ou une rotation mais un zoom avec la fonction Scalef. On utilise pour le zoom les valeurs données dans le sujet.

```
elif self.event.key == pygame.K_PAGEUP :
    gl.glScalef(1.1, 1.1, 1.1)

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

### 2. Question (1.e)

On garde encore la même structure que précédemment. Cette fois, la condition à valider est un coup de molette de souris (BUTTON\_WHEELUP/BUTTON\_WHEEL\_DOWN). On crée deux conditions, une pour le zoom et une pour le dé-zoom.

```
if self.event.button == pygame.BUTTON_WHEELUP :  
    gl.glScalef(1/1.1, 1/1.1, 1/1.1)  
  
if self.event.button == pygame.BUTTON_WHEELDOWN :  
    gl.glScalef(1.1, 1.1, 1.1)
```

### 3. Question (1.f)

On cherche tout d'abord à tourner le repère selon les axes x et z avec les mouvements de la souris lorsque le clic droit est enfoncé. Pour cela, on crée tout d'abord la condition sur le clic droit, l'objet ne tournera pas sans un appui du clic droit. Pour le faire tourner en bougeant la souris, on utilise comme vitesse de rotation la méthode rel qui nous permet d'avoir la vitesse de la souris. Ainsi, le repère tournera avec la souris.

Pour la translation, le résultat est similaire. On prend le clic gauche en condition du if et les mouvements de la souris comme vitesse de translation. Ici nous avons divisé la vitesse de la souris par 100 car le repère étant très petit, le moindre petit coup de souris faisait sortir nos axes de l'écran. C'est un problème que nous n'avons pas eu avec la rotation, les angles ne dépendant pas du « zoom » du repère.

```
# Processes the MOUSEMOTION event  
def processMouseEvent(self):  
  
    if pygame.mouse.get_pressed()[0] == 1 :  
        gl.glRotatef(10, self.event.rel[0], 0, self.event.rel[1])  
  
    if pygame.mouse.get_pressed()[2] == 1 :  
        gl.glTranslatef(self.event.rel[0]/100, 0, self.event.rel[1]/100)
```

## V. Création d'une section

### 1. Question (2.a)

Nous nous sommes penchés ici sur la création d'une section dans la fenêtre de pygame. Nous avons d'abord pris la fonction exemple en tant que référence et complété celle-ci afin d'obtenir la liste des points de notre futur parallélépipède. Le schéma sur le sujet de TP nous a aidé à rentrer les coordonnées de nos points.

Pour les faces, il suffisait de rentrer les numéros des points par ordre d'implémentation et de les relier entre eux.

```
# Defines the vertices and faces  
def generate(self):  
    self.vertices = [  
        # Définir ici les sommets  
        [0,0,0],  
        [0, 0, self.parameters['height']],  
        [0, self.parameters['thickness'], self.parameters['height']],  
        [0, self.parameters['thickness'], 0],  
        [self.parameters['width'], 0, 0],  
        [self.parameters['width'], 0, self.parameters['height']],  
        [self.parameters['width'], self.parameters['thickness'], self.parameters  
        ['height']],  
        [self.parameters['width'], self.parameters['thickness'], 0]  
    ]  
  
    self.faces = [  
        # définir ici les faces  
        [0,1,2,3],  
        [4,5,6,7],  
        [0,1,5,4],  
        [0,3,7,4],  
        [2,3,7,6],  
        [1,2,6,5]  
    ]
```

## 2. Question (2.b)

La partie de remplissage des faces du parallélépipède nous a posé problème. En effet, la méthode demandée dans le sujet est de remplir une face avec le code donné et utiliser des matrices de transposition pour transformer cette face afin de pouvoir remplir toutes les faces de notre rectangle. Nous n'avons pas bien compris cette consigne. Nous avons donc, comme le code qui définissait une face était donné, défini les 5 autres faces de la même manière, en adaptant le code avec les bonnes coordonnées.

```
gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
gl.glColor3fv([0.5, 0.5, 0.5]) # Couleur gris moyen
gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[1])
gl.glVertex3fv(self.vertices[2])
gl.glVertex3fv(self.vertices[3])

gl.glVertex3fv(self.vertices[4])
gl.glVertex3fv(self.vertices[5])
gl.glVertex3fv(self.vertices[6])
gl.glVertex3fv(self.vertices[7])

gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[1])
gl.glVertex3fv(self.vertices[5])
gl.glVertex3fv(self.vertices[4])

gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[3])
gl.glVertex3fv(self.vertices[7])
gl.glVertex3fv(self.vertices[4])

gl.glVertex3fv(self.vertices[2])
gl.glVertex3fv(self.vertices[3])
gl.glVertex3fv(self.vertices[7])
gl.glVertex3fv(self.vertices[6])

gl.glVertex3fv(self.vertices[1])
gl.glVertex3fv(self.vertices[2])
gl.glVertex3fv(self.vertices[6])
gl.glVertex3fv(self.vertices[5])

gl.glEnd()
```

Durant la séance 2, nous avons modifié la fonction d'affichage des faces de la section afin que celle-ci n'affiche que les faces choisies qui se trouvent dans la liste des faces. On utilise pour cela une boucle for qui va parcourir la liste des faces.

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    self.drawEdges()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv(self.parameters['color'])
        for j in i :
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

## 3. Question (2.c)

Pour les arrêtes, on utilise la même méthode en définissant le point de départ et le point d'arrivée de chaque arrête. Pour la couleur on met du noir (0,0,0) pour pouvoir différencier facilement les arrêtes des faces.



```
# A compléter en remplaçant pass par votre code

gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_LINE)
gl.glBegin(gl.GL_QUADS)
gl.glColor3fv([0.2,0.2,0.2])

gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[1])

gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[4])

gl.glVertex3fv(self.vertices[0])
gl.glVertex3fv(self.vertices[3])

gl.glVertex3fv(self.vertices[1])
gl.glVertex3fv(self.vertices[2])

gl.glVertex3fv(self.vertices[1])
gl.glVertex3fv(self.vertices[5])

gl.glVertex3fv(self.vertices[2])
gl.glVertex3fv(self.vertices[3])

gl.glVertex3fv(self.vertices[2])
gl.glVertex3fv(self.vertices[6])

gl.glVertex3fv(self.vertices[3])
gl.glVertex3fv(self.vertices[7])

gl.glVertex3fv(self.vertices[4])
gl.glVertex3fv(self.vertices[5])

gl.glVertex3fv(self.vertices[5])
gl.glVertex3fv(self.vertices[6])

gl.glVertex3fv(self.vertices[6])
gl.glVertex3fv(self.vertices[7])

gl.glVertex3fv(self.vertices[4])
gl.glVertex3fv(self.vertices[7])

gl.glEnd()
```

## VI. Création des murs

### 1. Question (3.a)

Pour construire un Wall, il est nécessaire de donner en paramètre un 'width' et un 'height'. Les autres paramètres sont optionnels car le constructeur de classe les initialise s'ils ne sont pas donnés.

La classe suit la même construction que la classe Section.

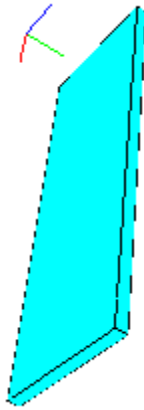
On écrit la méthode draw pour la classe Wall.

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1],
self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    for i in self.objects :
        i.draw()
    gl.glPopMatrix()
```

Les fonctions glPushMatrix et glPopMatrix servent à prendre en compte les matrices de translation et de rotation dans l'affichage du mur. On définit ces matrices avec l'orientation et la position du mur en entrée. La boucle for permet de dessiner plusieurs objets en une fois.



On modifie la couleur dans le main et on obtient le résultat suivant :



On remarque que le mur ne commence pas au point (0,0,0) car il est affecté par une matrice de translation. La couleur du mur est volontaire et relève d'un choix esthétique.

## VII. Création d'une maison

### 1. Question (1)

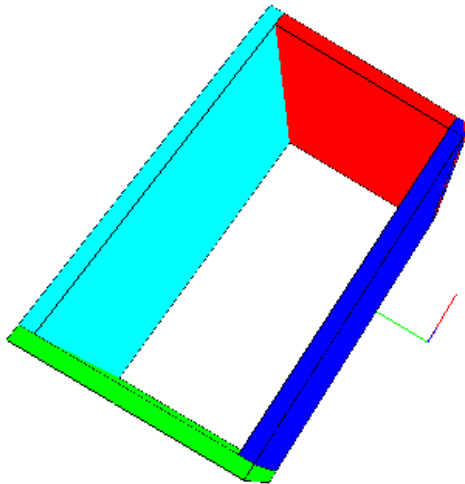
On s'inspire à nouveau de la méthode draw de la classe configuration afin d'écrire la méthode suivante pour la classe house :

```
# Draws the house
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glRotatef(self.parameters['orientation'], 0, 0, 1)
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1],
self.parameters['position'][2])
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL)
    for i in self.objects :
        i.draw()
    gl.glPopMatrix()
```

Cette méthode est la même que précédemment, seuls les objets pris en considération seront différents (la méthode permettra de tracer plusieurs murs pour former la maison).

On complète la question du fichier « main » avec les bonnes positions et tailles de murs afin d'avoir une maison avec 4 murs :

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'width' : 5, 'height' : 3,
'orientation' : 0, 'color' : [0, 0, 1]})
    wall2 = Wall({'position': [0, 0, 0], 'width' : 3, 'height' : 3,
'orientation' : 90, 'color' : [0, 1, 0]})
    wall3 = Wall({'position': [0, 2.8, 0], 'width' : 5, 'height' : 3,
'orientation' : 0, 'color' : [0, 1, 1]})
    wall4 = Wall({'position': [0, -5, 0], 'width' : 2.8, 'height' : 3,
'orientation' : 90, 'color' : [1, 0, 0]})
    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```



On peut remarquer que la maison ne commence pas à l'origine, de la même manière que les murs des questions précédentes.

Les couleurs sont voulues et nous on permis de différencier les différentes sections pour ajuster notre code.

## VIII. Création d'ouvertures

### 1. Question (5.a)

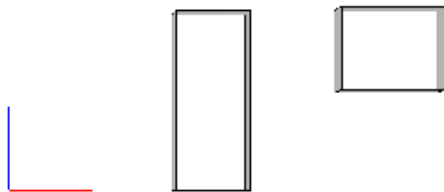
On définit les faces et les arêtes que l'on veut afficher dans la section Opening de la même manière que pour l'affichage des Sections :

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

Le fonctionnement de l'affichage est le même que précédemment. Seulement, cette fois-ci, on n'affiche que les 4 faces latérales pour chaque section afin de créer les ouvertures.

```
# Draws the faces
def draw(self):
    # A compléter en remplaçant pass par votre code
    gl.glPushMatrix()
    gl.glTranslatef(self.parameters['position'][0], self.parameters['position'][1],
self.parameters['position'][2])
    self.drawEdges()
    gl.glPolygonMode(gl.GL_FRONT_AND_BACK, gl.GL_FILL) # on trace les faces : GL_FILL
    for i in self.faces :
        gl.glBegin(gl.GL_QUADS) # Tracé d'un quadrilatère
        gl.glColor3fv(self.parameters['color']) # Couleur gris moyen
        for j in i :
            gl.glVertex3fv(self.vertices[j])
        gl.glEnd()
    gl.glPopMatrix()
```

On obtient le résultat suivant :



Ces deux sections ouvertes formeront les ouvertures dans un futur mur.

## 2. Question (2)

Nous avons implémenté la méthode canCreateOpening dans le classe Section :

```
# Checks if the opening can be created for the object x
def canCreateOpening(self, x):
    # A compléter en remplaçant pass par votre code
    if x.parameters['thickness'] == self.parameters['thickness'] :
        if x.parameters['position'][0] >= self.parameters['position'][0] :
            if x.parameters['position'][1] >= self.parameters['position'][1] :
                if x.parameters['position'][2] >= self.parameters['position'][2] :
                    if x.parameters['position'][2] + x.parameters['height'] <= self.parameters
['position'][2] + self.parameters['height']:
                        if x.parameters['position'][0] + x.parameters['width'] <= self.parameters
['position'][0] + self.parameters['width']:
                            return True
    return False
```

Nous l'avons testé avec la fonction Q5.b dans le fichier main.

Nous avons obtenu comme résultat (True, True, False). Il est normal d'obtenir False pour le troisième test car  $1 + 1.7 > 2.6$ . L'ouverture proposée dépasse la section par le « dessus » (le long de l'axe z)

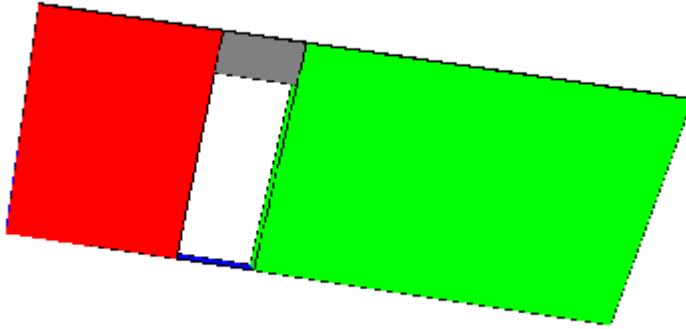
## 3. Question (5.c)

On ne présente pas la fonction createOpening dans ce compte rendu, la méthode étant plus volumineuse. Elle est cependant disponible en intégralité dans le code fourni sur GitHub

Nous testons cette fonction avec les fonctions fournies dans le fichier main, nous obtenons les résultats attendus :







Nota Bene : nous avons oublié de ne pas afficher les 'edges' avant de prendre la capture d'écran ! Pour faire un mur uniforme il faudrait mettre la même couleur pour toutes les sections et enlever les contours.

#### 4. Question (5.d)

La fonction enumerate permet de parcourir une liste en renvoyant pour chaque élément respectivement l'indice de l'élément dans la liste et l'élément. Elle renvoie donc un couple de valeurs.

Dans l'exemple donnée dans le sujet, section[0] renvoie « 0 » (la place de l'élément dans les listes des sections possibles) et section[1] renvoie l'objet opening1.

Nous avons complété la fonction add dans la classe Wall :

```
# Adds an object
def add(self, x):
    # A compléter en remplaçant pass par votre code
    good_section = self.findSection(x)
    if good_section == None :
        return self
    L = good_section[1].createOpening(x)
    self.objects.extend(L)
    self.objects.pop(good_section[0])
    return self
```

Nous ne pouvons malheureusement pas la tester car la fonction Q5.d dans le fichier main n'est pas fonctionnelle. Il nous manque du temps pour pouvoir aller plus loin.

## **IX. Conclusion du TP**

Nous n'avons pas trouvé ce TP facile, nous avons rencontré des difficultés mais l'objectif à atteindre nous ayant motivé nous avons, nous semble-t-il, bien avancé.

C'est probablement le TP durant lequel nous avons le plus appris.

Il est, par ailleurs, regrettable de devoir finir si près de la fin du TP, malgré la satisfaction d'être arrivé juste là.

<b>Fin du compte rendu</b>
----------------------------