DESAGE Hippolyte
MEYER Johan
Date du TP (14/12/2021)

Rapport de TP3 – Représentation visuelle d'objets.

I. Introduction

Le but de ce TP2 est de lire automatiquement des images de chiffre grâce à l'analyse d'image et grâce à des modèles qui nous ont été donnés.

II. Préparation

Utilisation de Pygame

(1) Grâce à ces lignes de code, on va pouvoir importer la bibliothèque de pygame et son package et d'initialiser pygame. Ensuite, on crée une nouvelle fenêtre de dimensions 200 par 300.
Lorsque l'on exécute ce code, une fenêtre apparaît pendant un petit laps de temps et disparaît automatiquement.

(2) Avec ce nouveau code, la fenêtre s'affiche et ne se ferme pas automatiquement, il faut appuyer sur une touche pour que celle-ci disparaisse. Cela est possible grâce à la boucle « while » qui attend qu'on appuie sur une touche pour pouvoir fermer la fenêtre. Cela est rendu possible grâce à la fonction « KEYDOWN » qui détecte' » la pression d'une touche.

Utilisation de Pyopengl pour représenter des objets 3D

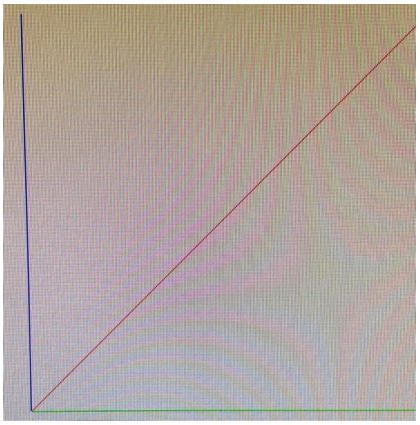
(1) Afin de pouvoir gérer la perspective en initialisant la matrice des perspectives nous allons devoir utiliser la commande : « perspective = glu.gluPerspective (45, 1, 0.1, 50) »
Afin d'utiliser la fonction gluPerspective (fovy, aspect, zNear, zFar) comme indiqué dans l'énoncé.

(2) Nous utilisons donc la fonction gl.glColor3fv([0, 0, 0]), dans les paramètres nous savons que le 1er correspond au rouge, le 2eme au vert et le 3eme au bleu.

Le code que nous utilisons est le suivant :

```
perspective = glu.gluPerspective (45, 1, 0.1, 50)

gl.glBegin(gl.GL_LINES) # Indique que l'on va commencer un trace en mode lignes (segments).

gl.glColor3fv([255, 0, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 1, -2)) # Deuxième vertice : fin de la ligne


gl.glColor3fv([0, 255, 0]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((1, 0, -2)) # Deuxième vertice : fin de la ligne


gl.glColor3fv([0, 0, 255]) # Indique la couleur du prochian segment en RGB
gl.glVertex3fv((0,0, -2)) # Premier vertice : départ de la ligne
gl.glVertex3fv((0, 1, -2)) # Deuxième vertice : fin de la ligne

gl.glEnd() # Find du tracé
pygame.display.flip() # Met à jour l'affichage de la fenêtre graphique
```

Résultat sur la fenêtre graphique :

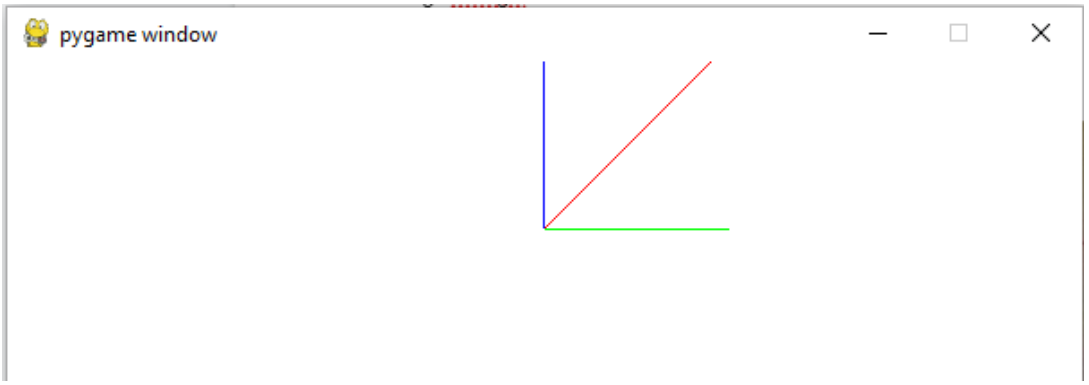


(3) Nous effectuons maintenant la translation, en décalant y de 2 et z de -5.

Le code que nous utilisons est le suivant :

```
# Placer ici l'utilisation de gluPerspective.

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

Résultat sur la fenêtre graphique :

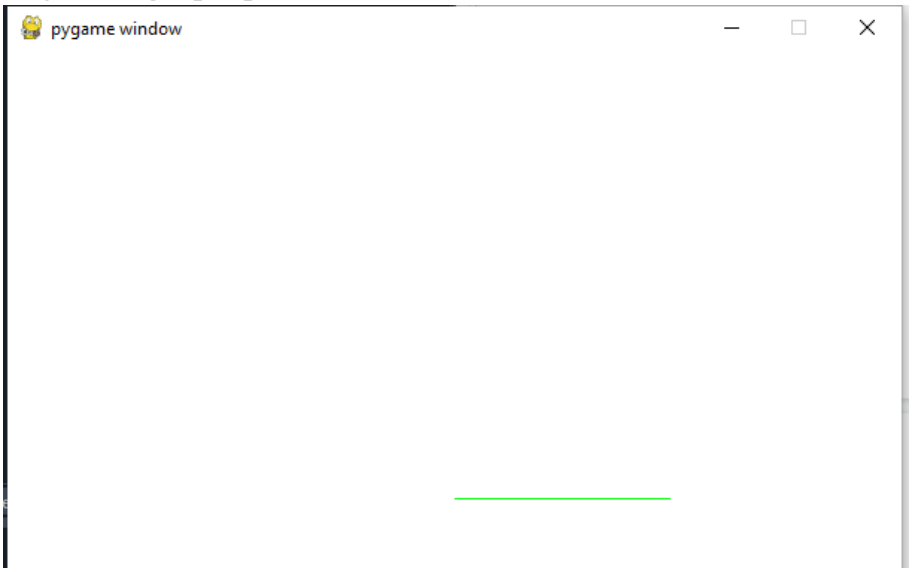


Nous avons fait la rotation, nous effectuons donc une rotation de 90° autour de l'axe x.

Le code que nous utilisons est le suivant :

```
# Placer ici l'utilisation de gluPerspective.

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

Résultat sur la fenêtre graphique :



Découverte de l'environnement de travail

1a) Nous avons écrit le code suivant :

```
def Q1a( ) :
    return Configuration( )
```

Nous remarquons que lorsque nous appuyons sur la touche a, le graphique disparaît, lorsque l'on réappuie sur la touche, le graphique apparaît.
Lorsque l'on appuie sur la touche z, le graphique tourne dans le sens trigonométrique et sur la touche Z, le graphique tourne dans le sens horaire.

1b) Le code va nous permettre de modifier la couleur d'un axe.

Le code que nous utilisons est le suivant :

```
def Q1b_f():

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

Résultat sur la fenêtre graphique :

Nous avons également une seconde possibilité pour changer la couleur de notre axe.
Le code que nous utilisons est le suivant :

```
def Q1b_f():

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

Résultat sur la fenêtre graphique :

Le chaînage de l'appel des méthodes *setParameter()* et *display()* est possible car ce sont deux méthodes de la même classe.
Un traitement particulier est effectué dans le « setter » pour le paramètre *screenPosition* est nécessaire car si non le programme n'a pas de référentiel de distance pour l'initialisation.

1c) Pour que l'axe *z* soit représenté verticalement sur l'écran et que l'axe *x* soit représenté horizontalement il faut effectuer une rotation de -90°, nous allons donc insérer l'instruction suivante dans la méthode *initializeTransformationMatrix()*.
Le code que nous utilisons est le suivant :

```
gl.glRotatef(-90,1,0,0)
```

Résultat sur la fenêtre graphique :

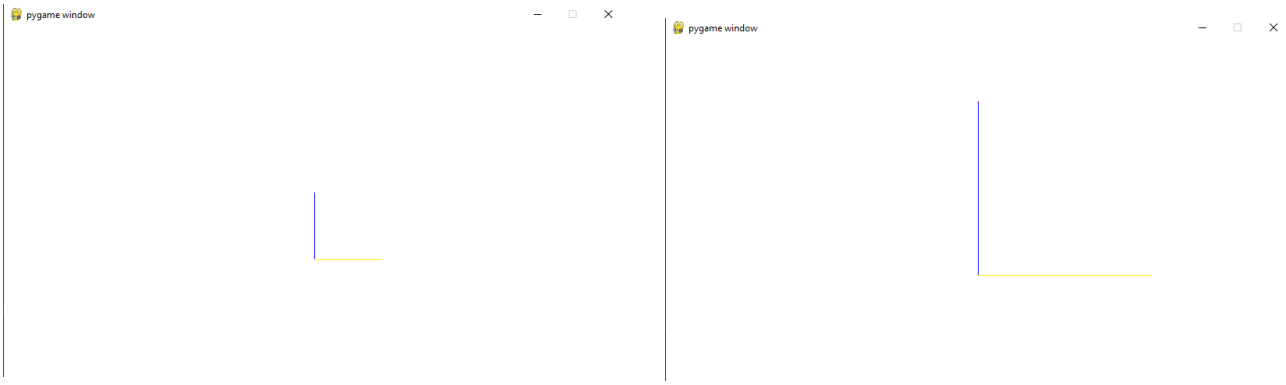
III. Mise en place des interactions avec l'utilisateur avec Pygame

1d) Nous avons ajouté la gestion des touches « Page Up », « Page Down ». Cela va nous permettre de zoomer de de dézoomer sur la fenêtre. Nous avons donc ajouté pour les trois axes un facteur d'échelle pour un zoom positif de 1.1 et pour un zoom négatif de 1/1.1.

Le code que nous utilisons est le suivant :

```
#zoom +/-
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)
```

Résultat sur la fenêtre graphique :



avant le zoom *après le zoom*

1e) Nous voulons maintenant pouvoir zoomer à l'aide de la roulette de la souris.

Comme pour la question précédente, nous avons ajouté pour les trois axes un facteur d'échelle pour un zoom positif de 1.1 et pour un zoom négatif de 1/1.1.

Le code que nous utilisons est le suivant :

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.buttom == 4:
        gl.glScalef(1.1,1.1,1.1)
    elif self.event.buttom == 5:
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```

Résultat sur la fenêtre graphique :

Nous obtenons les mêmes résultat qu'avec les zooms grâce aux touches

1f) Nous voulons maintenant pouvoir faire des rotations et des translations lorsque les boutons de la souris sont appuyés.

Si jamais nous appuyons sur le bouton de gauche nous allons récupérer le mouvement de la sourie sur l'axe x pour l'appliquer au mouvement de l'affichage sur la l'axe x et nous effectuons la même opération sur l'axe Z. Pour la translation l'opération sera sensiblement le même procédé mais nous allons diviser le mouvement de la sourie puisque si non c'est trop rapide et la précision est trop importante.

Le code que nous utilisons est le suivant :

```
# Processes the MOUSEMOTION event
def processMouseMotionEvent(self):

    if pygame.mouse.get_pressed()[0] == 1:
        gl.glRotate(-self.event.rel[0],1,0,0)
        gl.glRotate(-self.event.rel[1],0,0,1)

    if pygame.mouse.get_pressed()[2] == 1:
        gl.glTranslate(self.event.rel[0]/20,0,0)
        gl.glTranslate(0,0,self.event.rel[1]/20)
```

IV. Création d'une section

2a)

Nous allons désormais devoir gérer la section et l'afficher. Dans les vertices nous créons tous les points de notre parallélogramme grâce à la hauteur, a la largeur et la profondeur . Nous avons donc les six points nécessaires. Ensuite dans le faces nous allons créer les six faces une par une, utilisant les points numérotés que nous venons de créer :

Le code que nous utilisons est le suivant :

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

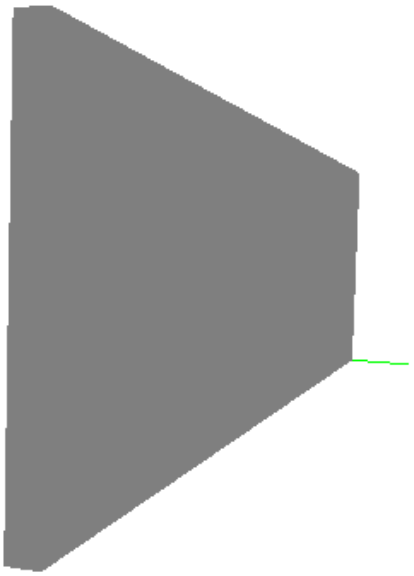
2b)

Il va maintenant nous falloir dessiner les faces que nous avons créer. Le glPushMatrix nous permettra de créer un nouveau calque. Ainsi pour toutes les faces nous utiliserons les vertices grâce à deux boucles les unes dans les autres.

Le code que nous utilisons est le suivant :

```
# Draws the faces
def draw(self):
    gl.glPushMatrix()

    gl.glBegin(gl.GL_QUADS)
    gl.glColor3fv(self.parameters['color'])

    for face in self.faces:
        for i in range (0,4):
            gl.glVertex3fv(self.vertices[face[i]])

    gl.glEnd()

    gl.glPopMatrix()
```

Résultat sur la fenêtre graphique :



2c)

Afin de gérer les arêtes nous allons utiliser la méthode drawEdges. Dans celle-ci nous définissons toutes les arêtes une à une avec les deux points qui les composent. Pour chacune des arêtes nous définissons une nouvelle couleur pour que les arêtes se distinguent des faces, nous la nommons ‘color2’. A la suite de cela nous allons devoir lier le drawEdges avec le draw que nous utilisons pour les faces qu’on les traces (les changements sont représentés sur la deuxième photo ci-dessous).

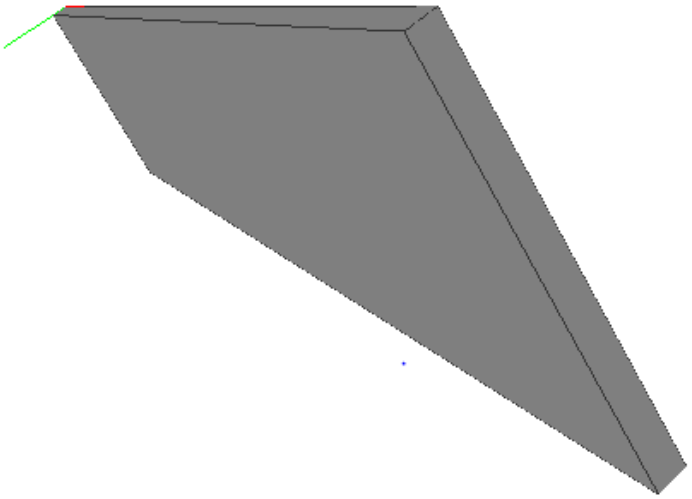
Le code que nous utilisons est le suivant :

```
# Draws the edges
def drawEdges(self):

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_LINE)

    gl.glBegin(gl.GL_LINES)

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[0])
    gl.glVertex3fv(self.vertices[1])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[1])
    gl.glVertex3fv(self.vertices[2])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[2])
    gl.glVertex3fv(self.vertices[3])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[3])
    gl.glVertex3fv(self.vertices[0])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[5])
    gl.glVertex3fv(self.vertices[6])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[6])
    gl.glVertex3fv(self.vertices[7])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[7])
    gl.glVertex3fv(self.vertices[4])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[4])
    gl.glVertex3fv(self.vertices[5])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[1])
    gl.glVertex3fv(self.vertices[5])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[2])
    gl.glVertex3fv(self.vertices[6])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[3])
    gl.glVertex3fv(self.vertices[7])

    gl.glColor3fv(self.parameters['color2'])
    gl.glVertex3fv(self.vertices[0])
    gl.glVertex3fv(self.vertices[4])

    gl.glEnd()
```

Les changements dans la méthode draw:

```
if self.parameters['edges']:
    self.drawEdges()

gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_FILL)
```

Résultat sur la fenêtre graphique :



v. Création des murs

3a)

Dans le constructeur du fichier Wall, il y a plusieurs tests qui s'assurent que les paramètres de paramètres sont bien présents. Dans le cas contraire, le constructeur va remplacer ces paramètres par des valeurs par défaut, des valeurs simples.

Nous avons donc dû utiliser le même schéma que les méthodes draw et add dans la méthode configuration, ces ajouts nous permettront par la suite d'ajouter plusieurs murs afin de construire la base de la maison.

Les codes que nous utilisons est le suivant :

```
# Draws the faces
def draw(self):
    for x in self.objects:
        x.draw()
```

```
# Adds an object
def add(self, x):
    self.objects.append(x)
    return self
```

Dans la partie 3a nous allons définir le premier mur, nous pouvons ajouter à cela les rotations que nous souhaitons par la suite. Nous pourrons donc choisir l'épaisseur, la largeur et la hauteur de ce dernier.

Le code que nous utilisons est le suivant :

```
def Q3a():

    wall1 = Wall({'position': [0, 0, 0], 'width':2, 'height':2, 'thickness':0.2, 'color': [0.5, 0.5, 0.5]})
    return Configuration().add(wall1)
```

VI. Création d'une maison

Nous allons dans un premier temps modifier la méthode draw dans la classe house. Pour cela nous nous inspirons de la méthode draw de la classe configuration. Nous aurons un schéma équivalent mais cette fois au lieu de créer qu'un seul mur notre boucle nous permettra de créer nos quatres mus.

Le code que nous utilisons est le suivant :

```
# Draws the house
def draw(self):

    gl.glPushMatrix()

    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
    gl.glRotate(self.parameters['orientation'],0,0,1)
    # for i in range (0, len(self.objects)):
    #     self.objects[i.draw()
    for obj in self.objects:
        obj.draw()
    gl.glPopMatrix()
```

Avant toutes choses il faut définir la rotation et la translation que nous utiliserons pour la définition de nos murs.

Le code que nous utilisons est le suivant :

```
gl.glPushMatrix()
gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])
gl.glRotate(self.parameters['orientation'],0,0,1)
```

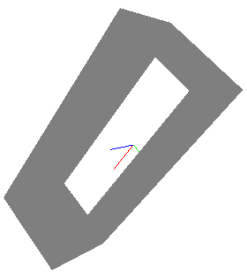
Nous allons maintenant devoir définir nos quatre murs. Nous les positionnons et le faisons bouger afin d'obtenir un carré qui définit notre maison.

Le code que nous utilisons est le suivant :

```
def Q4a():
    # Ecriture en utilisant des variables : A compléter
    wall1 = Wall({'position': [0, 0, 0], 'orientation':0, 'width':6, 'height':2, 'thickness':0.2, 'color': [0.5, 0.5, 0.5]})
    wall2 = Wall({'position': [0, 0, 0], 'orientation':-90, 'width':2, 'height':2, 'thickness':0.2, 'color': [0.5, 0.5, 0.5]})
    wall3 = Wall({'position': [6-0.2, 0, 0], 'orientation':-90, 'width':2, 'height':2, 'thickness':0.2, 'color': [0.5, 0.5, 0.5]})
    wall4 = Wall({'position': [0, -2-0.2, 0], 'orientation':0, 'width':6, 'height':2, 'thickness':0.2, 'color': [0.5, 0.5, 0.5]})

    house = House({'position': [-3, 1, 0], 'orientation':0})
    house.add(wall1).add(wall3).add(wall4).add(wall2)
    return Configuration().add(house)
```

Le résultat est bien celui escompté nous pourrons également faire bouger la maison par rapport à celle là si nous décidons d'en créer une deuxième.

Résultat sur la fenêtre graphique :



VII. Création d'ouvertures

5a)

Le principe pour cette question sera sensiblement le même que lorsque nous avons créé la maison. Seulement cette fois nous afficherons que certains bords de deux parallélépipèdes. Donc lorsque nous allons définir les deux formes nous ne rentrerons pas les faces supérieures et inférieures.

Le code que nous utilisons est le suivant :

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

Dans le draw nous aurons le même sans les éléments superflus pour les rotation et la rotation par exemple .

Le code que nous utilisons est le suivant :

```
# Draws the faces
def draw(self):

    gl.glPushMatrix()
    gl.glTranslate(self.parameters['position'][0],self.parameters['position'][1],self.parameters['position'][2])

    gl.glPolygonMode(gl.GL_FRONT_AND_BACK,gl.GL_FILL)
    gl.glBegin(gl.GL_QUADS)
    gl.glColor3fv(self.parameters['color'])

    for face in self.faces:
        for i in range (0,4):
            gl.glVertex3fv(self.vertices[face[i]])

    gl.glEnd()


    gl.glPopMatrix()
```

Nous obtenons bien les deux pseudo-ouvertures comme dans l'ennoncé.

Résultat sur la fenêtre graphique :