SANGOUARD Marine

MONNIER Marine

JENNY Camille

14/12/2021

# Rapport de TP3 – Représentation visuelle d’objets

## Introduction

Lors de ce TP, nous allons réaliser des représentations d’objets 3D à l’écran dans une fenêtre graphique qui permet des opérations de zoom, de rotations et de translations. Par exemple, nous allons représenter des maisons à partir d’objets simples que l’on va construire progressivement comme les murs, les portes et les fenêtres. Pour cela, nous allons nous utiliser 2 modules : Pygame et PyOpenGL

## Préparation à faire avant le TP

### Utilisation de Pygame

### À l’aide de la documentation de Pygame, expliquer ces 4 lignes.

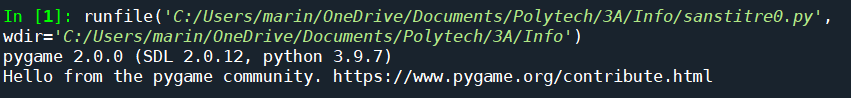
import pygame => importe le package pygame avec tous ses modules disponibles

pygame.init() => permet d’initialiser chacun de ces modules

pygame.display.set\_mode() => créé une fenêtre graphique, ici “ecran” est en fait un objet “Surface” qui est visible par l’utilisateur, avec comme propriétés : largeur 300 et hauteur 200.

pygame.quit() => quitte/supprime/ferme la fenêtre graphique

Rien ne se passe, il y a seulement le message ci-dessous qui apparaît :



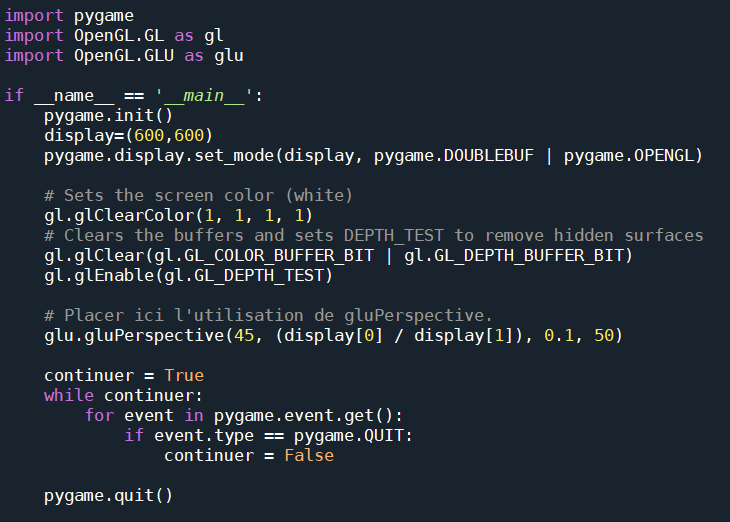
### Appuyer sur un bouton quelconque du clavier et observez ce qu’il se passe. Expliquer à l’aide du code et de la documentation de Pygame, comment on est parvenu à ce résultat.

Une fenêtre s’est ouverte. Si l’on appuie sur une touche du clavier, cette dernière se referme. En effet, la boucle “while” avec la variable “continuer = True” permet d’avoir la fenêtre qui reste ouverte. Elle se fermera que lorsque l’on ressortira de cette boucle, c’est à dire si le type d’un événement “pygame.KEYDOWN” a lieu (if event.type (in pygame.event.get()) == pygame.KEYDOWN). D’après la documentation, les événements pygame.KEYDOWN interviennent lorsque les boutons du clavier sont enfoncés et relâchés.

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

### Initialisez la matrice de perspective puis exécuter le fichier et voir s’il n’y a pas d’erreurs.

Le programme ne marchait pas, il y avait un problème avec le “exit()” donc nous avons repris la boucle “while” du programme précédent pour fermer la fenêtre mais cette fois-ci en utilisant “pygame.QUIT” au lieu de “pygame.KEYDOWN”. Nous avons ainsi le programme suivant :



### Utiliser cet exemple pour tracer un segment pour chaque axe x, y et z en couleurs rouge, vert et bleu respectivement.

### Découverte de l’environnement

### 1a) Analyser le fichiers main.py et Configuration.py de manière précise et détaillée, les points principaux du code et du résultat obtenu à l’écran lorsque l’on appuie sur les touches « a », « z » et « Z ».

Après avoir ajouté return Configuration(), on peut utiliser certaines touches.

La touche ‘a’ permet d’afficher ou de ne plus afficher le repère.

La touche ‘z’ permet de faire tourner le repère autour de l’axe z dans le sens antihoraire, ‘Z’ dans le sens horaire.

En effet, la def « Q1a » utilise la classe Configuration programmée dans le fichier configuration et la méthode utilisée lorsque l’on appuie sur ces touches est celle appelée “processKeyDownEvent”

Cette méthode appelle la fonction « rotate » avec un déplacement vers les x positifs (donc sens antihoraire) lorsqu’elle détecte l’appui de la touche ‘z’, ou avec un déplacement vers les x négatifs (donc sens horaire) lorsqu’elle détecte l’appui des touches Shift et ‘z’. On utilise l’attribut dict pour gérer indifféremment l'utilisation d’un clavier Qwerty ou Azerty.

L’appuie sur la touche ‘a’ désactive les paramètres des axes (pour désactiver l’affichage) après un léger temps de latence (utilisation de wait(300))

### 1b) Analyser la modification des paramètres par défaut, puis Expliquer pourquoi le chaînage de l’appel des méthodes setParameter() et display() est possible. Expliquer pourquoi un traitement particulier doit être effectué dans le « setter » pour le paramètre screenPosition.

La ligne « Configuration({'screenPosition': -5, 'xAxisColor': [1, 1, 0]}).display() » permet de configurer les paramètres par défauts des axes. Ici on attribue la couleur jaune à l’axe x; y et z gardent leur valeur RVB initiales.

L’ajout de « setParameter('xAxisColor', [1, 1, 0]).setParameter('yAxisColor', [0,1,1]) » permet de modifier ces paramètres. Ici on met x en jaune (ce qui ne change rien) et y en cyan.

On peut chaîner la méthode display après la méthode setParameter car la méthode setParameter renvoie son paramètre self, et c’est ce qui est nécessaire d’avoir en entrée pour utiliser la méthode display.

Si le paramètre à modifier est un code couleur d’axe, il suffit dans le setter de mettre à jour ce paramètre en changeant sa valeur dans son dictionnaire.

Or si le paramètre à modifier est ScreenPosition, il est nécessaire d’appeler la fonction initializeTransformationMatrix(). Cette méthode permet de placer l’écran à la position z demandée (donc le placer plus ou moins proche).

### 1c) Ajouter une seule instruction à la méthode initializeTransformationMatrix() pour que l’axe z soit représenté verticalement sur l’écran et que l’axe x soit représenté horizontalement.

Nous avons rajouter « gl.glRotatef(-90, 1, 0, 0) » afin de faire une rotation de 90° selon l’axe x (basculement vers l’extérieur).

## Mise en place des interactions avec l’utilisateur avec Pygame

## 1d) « Page up » et « Page down» :

Si la touche appuyée est Page Up (i.e. Page Down) , on modifie l'échelle à l’aide de la fonction glScale : on multiplie (i.e. divise) par 1,1 les 3 paramètres des axes.

## 1e) Changement d’échelle par la souris :

A présent, on réalise la même action mais pour des entrées différentes, à savoir le mouvement de la molette de la souris. Si l’entree pygame.MOUSEBUTTONDOWN vaut 4, on fait un zoom avant (screen Position x1.1), si elle vaut 5 on fait un zoom arrière (screen Position / 1.1). Lors du TP, la gestion de la molette de la souris ne fonctionnait pas.

**1f) Déplacement par la souris :**

Si on détecte le bouton gauche pressé, on fait une rotation d’angle 5°. On divise par 10 le déplacement en x, y et z pour que la figure ne parte pas trop loin dans l’espace.

Si on détecte le bouton droit pressé, on fait une translation, en divisant encore une fois par 10 la valeur du déplacement en x, y et en z.

Résultats des tests de bases :

https://lh3.googleusercontent.com/lVlPV7RD0tNVNRgKjnBlgLZRdkgof1uwinYgDSQJVbjB8UM8QRmV2NFVwkXEyEJq3aHEj8mUccuESZNieYh42OqmAI0lxE10vAspnwG92FsRjGwEmU9bbAE3hYff8UupsMRFeCk3

1. **Création de section**

**2a) écrire la méthode generate(self) de la classe Section :**

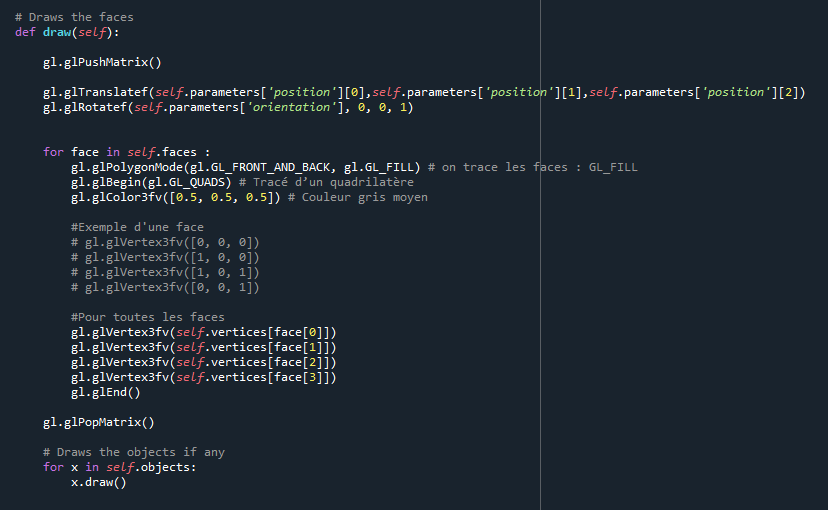
Pour définir les sommets, on initialise simplement leurs coordonnées en prenant les valeurs souhaitées de largeur, épaisseur, hauteur pour x,y,z.

Pour définir les faces, on crée une liste 2D dans laquelle on indique les 4 sommets délimitant chaque face.

**2b) Analyser la fonction Q2b()** **dans le fichier main.py et expliquer l’instruction** « **Configuration().add(section).display() » et écrire la méthode draw() pour la classe Section afin de tracer les faces de la section en gris :**

L’instruction Configuration().add(section).display()du fichier main.py utilise la méthode add de configuration pour ajouter une section dont les différents paramètres sont fixés, pour ensuite l’afficher avec display. Pour faire cela, cette instruction utilise le chaînage de plusieurs méthodes.

Dans la fonction draw, on commence par définir une matrice de projection propre à la section. On effectue ensuite une translation et une rotation sur la section qui est par défaut en position (0, 0, 0) pour la placer en (1, 1, 0). Une fois placée, on trace la section à l’aide d’une boucle qui va créer une par une chaque face. Pour les faces la couleur choisie est le gris moyen (0.5, 0.5, 0.5).



**2c) Ecrire la méthode drawEdges() dans la classe Section et modifier la méthode draw() de la question (2). b. :**

Nous commençons par indiquer la nouvelle couleur qui sera appliquée aux arêtes. On choisit une couleur légèrement plus foncée que la couleur utilisée précédemment.

On utilise glBegin(gl.GL\_LINES)pour tracer des segments.

Nous appliquons cette couleur a toutes les arêtes: pour cela il faut tracer des traits avec glVertex3fv entre tous les sommets deux à deux, 12 fois.

On utilise glEnd pour indiquer la fin du tracé.

Nous avons ensuite rajouter :

if self.parameters['edges'] == True:

self.drawEdges()

dans la méthode draw de Section afin d'effectuer le traçage des arêtes avant le remplissage des faces.

**V - Création des murs**

**3a) :**

Dans le constructeur, on utilise les paramètres déjà donnés auparavant. Si certains paramètres sont manquants, alors on leur attribut des valeurs par défaut.

Ensuite on crée une liste d’objets dans laquelle on va mettre les parentSections. Ces dernières sont des sections avec tous les paramètres définis précédemment, qui proviennent de la classe Section dans le programme Section.

Dans la méthode Q3a dans le main, on ajoute un mur avec les paramètres que l’on souhaite en chaînant la méthode add de configuration à Wall.

Résultats des tests de Wall

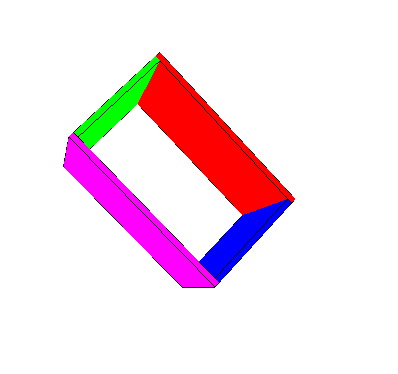
https://lh6.googleusercontent.com/Clt_7S75PkIghxqj6--ht81oeoDRUZ8smWorIUJJIpPfnm0tV6C2oVkkWx_HYKLTUJ7Eh0QCXMGK0J5BlKwjppMQu7aNEvRaxZvzeXzoYkT8ay4hdijoy92yg0kbev8PX68F0UES

**VI - Création d’une maison**

**4a) :**

Dans la méthode Q4a dans le main, on crée 4 murs avec les paramètres que l’on souhaite en chaînant la méthode add de configuration à wall. On crée un premier mur (W=10, T=0.3, H=5), un deuxième qui est identique au premier mais translaté sur y de la longueur souhaitée de la maison, un troisième et un quatrième sur lesquels on effectue une rotation de 90 par rapport à l’axe z. Ces derniers sont espacés de la largeur de la maison. Pour que les murs ne se chevauchent pas, nous avons réajusté quelques paramètres.

Pour mieux visualiser chaque mur, on a attribué une couleur différente à chaque mur.



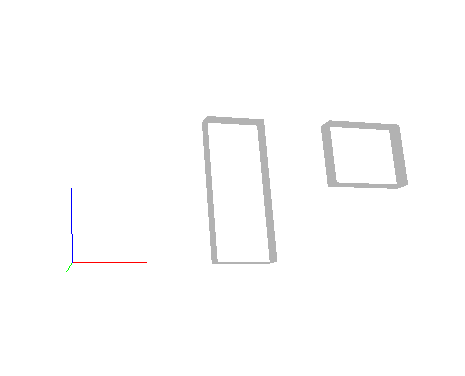
**VII - Création d’ouvertures**

**5a) :**

Dans la classe Opening, on complète generate()

On crée les vertices et les de la même façon que dans Section, mais avec 2 faces opposées en moins pour créer l’ouverture.

On complète également draw() de la même façon que dans Section, mais sans la rotation.



**5b) :**

Dans la méthode canCreateOpening(self, x), on retourne Vrai si :

le coin inférieur gauche de l’ouverture appartient au mur;

et la hauteur (height) de l’ouverture + sa position en z ne dépasse pas la hauteur du mur

et la largeur de l’ouverture (width ici) est inférieur à la distance entre le coin inférieur gauche de l’ouverture et le coin inférieur droit du mur;

et l'épaisseur (thickness) de la porte est égale à celle du mur.

On obtient false pour la vérification de l’opening 3 car la hauteur de la fenêtre par rapport au sol ajoutée à la hauteur de la fenêtre est supérieure à la hauteur de la section.

## Conclusion

Nous avons, lors de ce TP, réussi à modéliser une section puis à arranger ces sections pour former une maison dans laquelle il sera ensuite possible d’ajouter des objets. Nous avons ensuite commencé la réalisation d’ouvertures qui serviront de fenêtre et de portes. Nous avons aussi modélisé les rotations et translations de la vue sur la maison grâce à la souris ou certaines touches du clavier.