*LAJUGIE Rodolphe*

*BARRIQUAND Valentin*

**Projet TSI**

Notre jeu est un jeu de simulation de musée dans lequel vous pourrez voir différentes œuvres ainsi que leurs noms. Attention on touche avec les yeux !

# Description de la librairie OpenGL

* Tout d’abord OpenGL est une librairie disponible sous différents langage de programmation. Nous utilisons ici la version OpenGL pour Python. Cette librairie permet de gérer les objet 3D dans l’espace ainsi que l’affichage de texture et déplacement de camera.
* Nous avons utilisé deux programmes GPU, l’un pour l’affichage des objets 3D tel que nos mur ou encore les œuvres et l’autre pour l’affichage du textes à l’approche des statues. Ce programme permet de générer et compiler les vertex shader et les fragment shader pour ensuite avoir un programme qui suit les lois qui ont été décrites dans les deux fichiers précédent.

# Structure

Nous avons d’abord commencé par agrandir le terrain sur lequel nous travaillons en lui donna tune taille de 100 par 100 et nous avons aussi changé sa texture comme on peut le voir sur le code suivant (tout ces codes se situe dans main.py) :

m5 = Mesh()

    p05, p15, p25, p35 = [-50, 0, -50], [50, 0, -50], [50, 0, 50], [-50, 0, 50]

    n5, c5 = [0, 1, 0], [1, 1, 1]

t05, t15, t25, t35 = [0, 0], [1, 0], [1, 1], [0, 1]

    m5.vertices = np.array([[p05 + n5 + c5 + t05], [p15 + n5 + c5 + t15], [p25 + n5 + c5 + t25], [p35 + n5 + c5 + t35]], np.float32)

    m5.faces = np.array([[0, 1, 2], [0, 2, 3]], np.uint32)

    texture = glutils.load\_texture('ressources/textures/sol.jpg')

    VAO = m5.load\_to\_gpu()

    o1 = Object3D(VAO, m.get\_nb\_triangles(), program3d\_id, texture, Transformation3D())

    viewer.add\_object(o1)

Ensuite à l’aide des deux boucles suivantes nous créons les murs de notre musée :

m2 = Mesh.load\_obj('ressources/objets/cube.obj')

    m2.normalize()

    m2.apply\_matrix(pyrr.matrix44.create\_from\_scale([1, 1, 1, 1]))

    texture2 = glutils.load\_texture('ressources/textures/noir.jpg')

    vao2 = m2.load\_to\_gpu()

    for val in range(30):

        tr2 = Transformation3D()

        tr2.translation.y = 0

        tr2.translation.z = -30

        tr2.translation.x = -30+2\*val

        o2 = Object3D(vao2, m2.get\_nb\_triangles(), program3d\_id, texture2, tr2)

        viewer.add\_object(o2)

*Code pour la base du mur*

m3 = Mesh.load\_obj('ressources/objets/cube.obj')

    m3.normalize()

    m3.apply\_matrix(pyrr.matrix44.create\_from\_scale([0.9, 15, 30, 1]))

    texture = glutils.load\_texture('ressources/textures/blanc.jpg')

    vao3 = m3.load\_to\_gpu()

    tr6 = Transformation3D()

    tr6.translation.x = -30

    tr6.translation.y = +1

    o6 = Object3D(vao3, m3.get\_nb\_triangles(), program3d\_id, texture, tr6)

    viewer.add\_object(o6)

*Code pour les murs hauts*

Enfin nous avons utilisé les mêmes lignes de code que pour le sol pour créer notre toit

Ensuite nous avons positionner des blocs que nous avons modifier avec la fonction scale afin de les rendre plus fin et ressemblant à des barrières :

mbar1 = Mesh.load\_obj('ressources/objets/cube.obj')

    mbar1.normalize()

    mbar1.apply\_matrix(pyrr.matrix44.create\_from\_scale([0.1, 1.7, 2, 1]))

    texturebar = glutils.load\_texture('ressources/textures/barriere.jpg')

    vaobar = mbar1.load\_to\_gpu()

    for loop in range(2):

        for val in range(7):

            trbar = Transformation3D()

            trbar.translation.y = 0

            trbar.translation.z = -27+val

            trbar.translation.x = 7.5 - 15\*loop

            obar = Object3D(vaobar, mbar1.get\_nb\_triangles(), program3d\_id, texturebar, trbar)

            viewer.add\_object(obar)

# Détails intérieurs

Etant donné que nous faisons un musée nous avons importé des stands en 3D et les avons positionner dans deux boucle pour la symétrie de part et d’autre de la pièce :

ms = Mesh.load\_obj('ressources/objets/pedestal.obj')

    ms.normalize()

    ms.apply\_matrix(pyrr.matrix44.create\_from\_scale([1,1,1,1]))

    vaostand = ms.load\_to\_gpu()

    texture = glutils.load\_texture('ressources/textures/BaseColor.png')

    for bcls in range(3):

        trs = Transformation3D()

        trs.translation.z = -22

        trs.translation.x = -15 + 15\*bcls

        trs.rotation\_center.z = 0.2

        os = Object3D(vaostand, m21.get\_nb\_triangles(), program3d\_id, texture ,trs)

        os.transformation.translation.y = 1

        viewer.add\_object(os)

Et puis nous avons poser sur ces socles les différentes œuvres à afficher (chaque création de notre chef est signalée par les lignes ‘=’ avec l’intitulé du code).

# Fonctionnalités

D’abord nous avons décidé de laisser la caméra suivre le joueur en permanence pour ce faire il faut appuyer une fois sur la barre espace au lancement afin de verrouiller la caméra à la 3ème personne.

Nous avons naturellement ajouté la fonctionnalité de collision avec tous les objets sauf le textes et les œuvres. En effet ce qui nous empêchera de traverser l’œuvre est la hitbox du piédestal qui nous empêche d’avancer davantage. Ces collisions sont gérées de la manière suivante : on regarde la distance entre notre personnage + le prochain déplacement et l’objet avec lequel nous voulons rentrer en collision, si cette distance est inférieur à une valeur choisi au préalable on n’effectue pas le déplacement. L’action de se déplacer est fait par la valeur de la variable booleen bol qui devient False si l’on est trop proche de l’objet et donc empêche l’action d'avancer

Cette fonction se trouve dans le fichier viewerGL.py :

d = pyrr.matrix33.apply\_to\_vector(pyrr.matrix33.create\_from\_eulers(self.objs[0].transformation.rotation\_euler), pyrr.Vector3([0, 0, 0.3]))

            bol = True

            for i in range(4,len(self.objs)-7):

                if abs(self.objs[0].transformation.translation.x + d[0] -self.objs[i].transformation.translation.x)<1.8 and abs(self.objs[0].transformation.translation.z + d[2] -self.objs[i].transformation.translation.z)<1.8 :

                    bol = False

            if bol :

                self.objs[0].transformation.translation += d

Nous avons décidé de gérer les collision que lorsque nous avançons afin de nous permettre de faire des easters eggs en passant à travers les murs uniquement en marche arrière (il y en a un essayez de le trouver, indice : du côté des voitures triangulaire).

Ensuite nous avons la gestion de l’affichage des noms des œuvres lorsqu’on s’approche d’elles. Tout d’abord nous créer autant d’objet texte que d’œuvres :

vao = Text.initalize\_geometry()

    texture = glutils.load\_texture('ressources/textures/fontB.jpg')

    for i in range(6):

        o = Text('', np.array([-0.8, 0.3], np.float32), np.array([0.8, 0.8], np.float32), vao, 2, programGUI\_id, texture)

        viewer.add\_object(o)

        o.visible = False

Puis après nous gérons l’affichage des textes dans des codes tous similaire à celui-ci qui se trouve dans viewerGL.py plus précisément dans l’action lors de l’appuie sur la touche HAUT :

if abs(self.objs[0].transformation.translation.x + d[0] - self.objs[1].transformation.translation.x)<3 and abs(self.objs[0].transformation.translation.z + d[2] -self.objs[1].transformation.translation.z)<3 :

                self.objs[-1].value = "Le singe suzanne"

                self.objs[-1].visible = True

            else :

                self.objs[-1].visible = False

Il regarde si la distance entre nous + notre future déplacement et l’objet que nous souhaitons est inférieur à une certaine valeur et si tel est le cas nous changeons la valeur du texte et mettons sa variable .visible de l’objet de class Text, à True afin de l’afficher. De plus ces mêmes lignes de codes sont présentes dans la touche pour reculer afin d’annuler l’affiche si on recule trop loin de l’œuvre.

Observation :

On a remarqué que, si le personnage tente de revenir à l’endroit où il spawn, celui est bloqué. Nous avons cherché d’où pouvait provenir l’erreur mais nous n’avons pas réussi à l’identifier.

Lien du GitHub si besoin de cloner le dépôt : <https://github.com/Valbrqd/Projet-TSI>

Quelques screen de la fenêtre :



*Vue lors de l’ouverture de la fenêtre (bien appuyer sur espace pour passer en vue 3e personne)*

*Une image contenant texte

Description générée automatiquement*

*En s’approchant assez d’une œuvre, le nom de celle-ci apparait*