WALLERAND Alex
SIMONET Théo
Date du TP (27/06/2001)

Rapport de TP3 – Représentation visuelle d'objets

I. Introduction

Dans ce TP, nous allons étudier la représentation d'objets en 3D sur une fenêtre graphique. Pour cela nous utiliserons la librairie pygame qui permettra de gérer l'interface graphique, et la libraire pyOpenGL pour afficher des objets dans la fenêtre générée.

II. Préparation

Utilisation de pyGame

1. Question (1).

Lorsque l'on rentre ce code dans Spider, on obtient un message qui indique que l'on utilise bien la librairie pygame et rien de plus. C'est normal, on importe d'abord le module pygame, puis la seconde ligne permet d'initialiser l'ensemble des modules pygame. Ensuite on donne pour valeur à la variable ecran l'initialisation d'une fenêtre pygame de taille 300x200, que l'on n'a pas le temps de voir s'afficher à l'exécution car la dernière ligne permet de stopper l'utilisation des modules de pygame.

2. Question (2).

L'exécution de ce code permet de générer une fenêtre pygame qui, lorsqu'on appuie sur une touche du clavier se ferme. En effet, on génére à l'aide de pygame une fenêtre de taDille 300*200 comme précédemment, puis à l'aide de la fonction de pygame qui permet d'obtenir les événements d'interactions avec l'ordinateur, on vérifie si l'utilisateur appuie sur une touche du clavier, ce qui permet alors de quitter la boucle tant que et de fermer la fenêtre.

Utilisation de pyOpenGL pour représenter des objets 3D

3. Question (1).

Il n'y a pas d'erreurs, la fenêtre pygame s'affiche avec deux axes. Dans le paramètre aspect de la fonction gluPerspective, on met le rapport des 2 valeurs de display, et pour les autres on met les valeurs indiqués dans l'énoncé à savoir : fovy=45, zNear=0.1 et zfar=50.

4. Question (2).

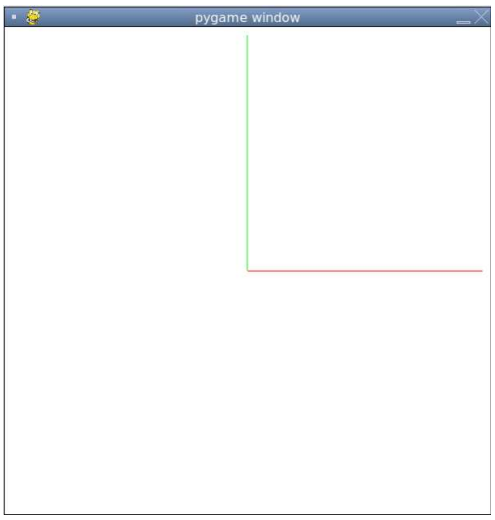
A l'aide de l'exemple, on trace de la même façon les 3 axes x,y et z en veillant bien aux valeurs de fin de tracé et de couleur.

```
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((2, 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, 2, -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((0, 0, 0)) # Deuxième vertice : fin de la ligne

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



5. Question (3).

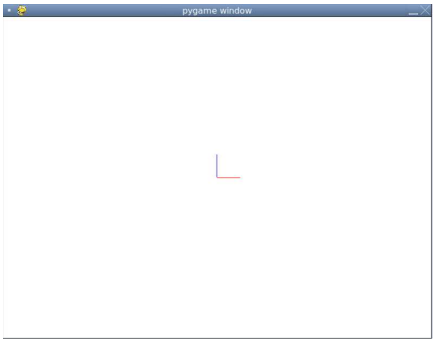
```
glu.gluPerspective(45,(display[0]/display[1]),0.1,50)

gl.glTranslatef(0,2,-5)
gl.glRotatef(-90,1,0,0)
```

Découverte de l'environnement de travail du TP

6. Question (1a).

Lorsque l'on rajoute la commande dans la fonction Q1a et que l'on exécute le fichier main, la fenêtre graphique s'affiche avec deux axes. On peut les faire tourner grâce à la touche z ou ne pas les afficher grâce à la touche a.



7. Question (1b).

Cette modification permet de déplacer de 5 pixels la fenêtre et de définir la couleur des axes.
Le chainage est possible car on met bien un point à la fin d'une méthode quand il y en a une après.

8. Question (1c).

Pour que l'axe z soit représenté verticalement et l'axe x horizontalement, on fait une rotation de 90 degrés à l'aide de la fonction glRotatef.

```
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)
```

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

1. Question (1d).

Pour changer le zoom avec les touches PageUp et PageDown on ajoute des conditions à l'aide de pyGame dans la fonction processKeyDownEvent aux touches correspondantes, puis à l'aide de la fonction glScalef on entre les valeurs de zoom voulues.

```
def processKeyDownEvent(self):
    # Rotates around the z-axis
    if self.event.dict['unicode'] == 'Z' or (self.event.mod & pygame.KMOD_SHIFT and self.event.key == pygame.K_z):
        gl.glRotate(-2.5, 0, 0, 1)
    elif self.event.dict['unicode'] == 'z' or self.event.key == pygame.K_z:
        gl.glRotate(2.5, 0, 0, 1)

    # Draws or suppresses the reference frame
    elif self.event.dict['unicode'] == 'a' or self.event.key == pygame.K_a:
        self.parameters['axes'] = not self.parameters['axes']
        pygame.time.wait(300)

    #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)
```

2. Question (1e).

On crée deux conditions, la première si l'événement button est wheelup, c'est-à-dire la molette vers l'avant, alors on zoom. Dans l'autre cas, si la molette est actionnée vers le bas alors on dézoom.

```
# Processes the MOUSEBUTTONDOWN event
def processMouseButtonDownEvent(self):
    if self.event.button == pygame.BUTTON_WHEELUP:
        gl.glScalef(1.1,1.1,1.1)
    elif self.event.button == pygame.BUTTON_WHEELDOWN:
        gl.glScalef(1/1.1,1/1.1,1/1.1)
```

3. Question (1f).

D'abord à l'aide de l'évenement pygame on vérifie si c'est le boutton gauche qui est appuyé, dans ce cas, on fait une rotation autour de x à l'aide de la fonction glRotatef, selon la valeur du déplacement obtenue dans l'attribut rel de l'événement MOUSEBUTTON.
Pour le clic droit, on fait de même en vérifiant que c'est bien le bouton droit de la souris qui est actionné, et on utilise glTranslatef pour déplacer les axes selon la valeur de déplacement dans l'attribut rel.

```
# 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)
    elif pygame.mouse.get_pressed()[2]==1:
        gl.glTranslatef(self.event.rel[0]/100,0,self.event.rel[1]/100)
```

IV. Création d'une section

1. Question (2a).

On s'inspire de l'ébauche afin de remplir la fonction generate : self.vertices contient l'ensemble des sommets de la section, et en fonction de la position des sommets dans l'attribut vertices, on remplit l'attribut faces avec des listes comprenant les sommets représentant une face.

```
# 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],
            [self.parameters['width'],self.parameters['thickness'],0],
            [0,self.parameters['thickness'],self.parameters['height']],
            [self.parameters['width'],self.parameters['thickness']self.parameters['height']]
            ]
    self.faces = [
            [0,3,2,1],
            [3,5,7,2],
            [0,3,5,4],
            [1,2,7,6],
            [0,4,6,1],
            [4,5,7,6]
            ]
```

V. Conclusion

Au travers de cette séance, nous avons pu créer et définir les principales propriétés de la fenêtre graphique que nous utiliserons dans la suite. Nous avons principalement appris l'utilisation des event de pygame afin d'opérer des changements sur la fenêtre à l'aide des fonctions du module pyOpenGl.
Maintenant que l'interface est correctement implémentée, nous devrons pour la prochaine séance créer les fonctions qui permettront de générer des objets dans la fenêtre tel que des murs.