Aluno: Calebe Pereira RGA: 2018.1904.020-8
Aluno: Rafael Germinari RGA: 2018.1904.026-7

Relatório

O trabalho proposto consiste em desenvolver um aplicação que atenda aos requisitos, adicionar e remover formas, habilitar e desabilitar reflexão, adicionar e remover fontes de luz, definir posição da câmera e vetor de visão, aplicar transformações nas formas, mostrar e esconder eixos, salvar a imagem corrente em arquivo png, mostrar as luzes posicionadas na cena e ler comandos de entrada por linha de comando.

Linguagem de Programação: Escolhemos a linguagem Python. Python3 versão 3.6.9.
Versão OpenGL: 3.0
Versão GLSL: 130

Parâmetros fixos: Não utilizamos constantes, mas atribuímos uma matriz model default, carregada com a matriz identidade e a shear default, que tem a mesma característica da modelDefault.

```
modelDefault = pyrr.matrix44.create_identity()
```

Arquivo zip: O arquivo zip possui o código fonte trabalho.py, o arquivo entrada.txt, usado para passar os comandos, a pasta shader, os arquivos .obj e .mtl das formas e o pdf do relatório, relatório_cg.pdf.

Optamos por utilizar uma classe objetos, sendo ela para armazenar informações de cada objeto que será desenhado.

```
class objeto(object):
```

Cada objeto armazena as seguintes informações: O shape, que é o tipo, o seu nome, a cor, sua model e wireModel, que é usado como uma flag para quando receber o comando wire_on, o objeto ser desenhado com GL_LINES.

Foram criadas algumas variáveis globais, pois elas são utilizadas em várias funções.

Criamos outra classe obj_light, usada para armazenar as luzes, cada luz armazena um nome, as posições (x, y, z), model e um lightMode, flag usada para saber se está ativo ou não.

Funções

def display:

- Essa função é executada a todo momento, nela o arquivo de entrada é aberto e nós percorremos ele para pegar os comandos e atribuímos esse comandos a uma lista (vet_obj). Depois disso, percorremos a lista e fizemos a verificação de cada elemento armazenado nela, podendo ser o comando, nomes, posições e a cor.

```
vet_obj = [] #vetor que pega os comandos do arquivo
lista_obj = [] #lista dos objetos
lista_luz = [] #lista das luzes
modelDefault = pyrr.matrix44.create_identity()#carrega a varivavel com a matriz identidae
shearDefault = pyrr.matrix44.create_identity()
arq = sys.argv[1]

arquivo = open(arq,'r')# le arquivo dos comandos
for linha in arquivo:
    linha = linha.replace('\n','')# adiciona oscomando para variavel linha

    for i in linha.split():# quebra o comando em partes e adiciona para a lista de obj
        vet_obj.append(i)

        print(vet_obj)

for i in range(len(vet_obj)):
    if(vet_obj[i] == 'add_shape'): #verifica se existe o comando add_shape
        # seta as variaveis no objeto começando com Nome,Shape,cor,model,wireMode,escala,rotação,translação,cam,reflects e shear
        aux = objeto(vet_obj[i+1],vet_obj[i+2],1,1,1,modelDefault,wireMode,1,1,1, 0,0,0,0, 0,0,0, 0,0,0, 0,0,0, 0.1,1,0.5, shearDefault)
        lista_obj.append(aux) #cria o objeto a ser desenhado


    elif(vet_obj[i] == 'color'): #verifica se existe o comando color
        for x in lista_obj:
            if(x.nome == vet_obj[i+1]):
                x.r = float(vet_obj[i+2])
                x.g = float(vet_obj[i+3])
                x.b = float(vet_obj[i+4])

    #verifica se existe o comando wire_on para ativar
    elif(vet_obj[i] == 'wire_on'):
        for x in lista_obj: #percorre a lista dos objetos e modifica o atributo wireMode para 1
            wireMode = 1
            x.wireMode = wireMode
```

def draw:

- Essa função é responsável por executar os comandos para desenhar os objetos armazenados na lista de objetos (lista_obj). Então percorre a lista de objetos e para cada um deles, verifica qual o shape e pega as informações para desenhar o objeto em questão.

```
for i in lista_obj: #varre a lista de objetos
    #desenha o cubo
    if(i.shape == 'cube'):
        init(i.shape,lista_obj) #manda o .obj que vai ser carregado e a lista dos objetos
        glUseProgram(shaderProgram)
        glBindVertexArray(vao)
        glBindBuffer(GL_ARRAY_BUFFER, vbo)
        # passa as transformacoes para os shaders
        glUniformMatrix4fv(uMat, 1, GL_FALSE, i.model)
        glUniformMatrix4fv(idView, 1, GL_FALSE, view)
        glUniformMatrix4fv(idProj, 1, GL_FALSE, projection)
        if(reflectionFlag == 1):    # verifica se o reflections esta ativo
            for x in lista_luz: # percorre a lista das luzes
                # passa as informacoes para os shaders
                glUniform3fv(idLightPos, 1,x.x,x.y,x.z)
                glUniform3fv(idColor,1,[i.r,i.g,i.b])
                glUniform3fv(idLight,1,[1.0,1.0,1.0])
                glUniform3fv(reflec,1,[i.ambient, i.diffuse, i.specular])
                glUniform3fv(ambientForce,1,[i.ambientForce])
                glUniform3fv(diffuseForce,1,[i.diffuseForce])
                glUniform3fv(specularForce,1,[i.specularForce])
                glUniform3fv(idViewPos,1,poscam)

        # passa as transformacoes para os shaders
        glUniformMatrix4fv(uMat, 1, GL_FALSE, i.model)
        glUniformMatrix4fv(idView, 1, GL_FALSE, view)
        glUniformMatrix4fv(idProj, 1, GL_FALSE, projection)

        Color = glGetUniformLocation(shaderProgram,"uColor")
        glUniform3f(Color,i.r, i.g, i.b) # atribuindo a cor
        if(i.wireMode == 1): #verifica se esta habilitado o wiremode
            glDrawArrays(GL_LINE_LOOP, 0, 42)
        else:
            glDrawArrays(GL_TRIANGLES, 0, 42) #desenhando o cubo
```

def lendo_obj:

- Função usada para ler os arquivos .obj.

```
def lendo_obj(nome):
    texto = nome+'.obj'
    aux = readObjFile(texto)
    aux.parse()
    for name, material in aux.materials.items():
        return material.vertices
```

def readShaderFile:

- Função usada para ler os shaders (arquivos vp e fp).

```
def readObjFile(path):
    return obj.Wavefront(path)
```

def init:

- Essa função é responsável por carregar todas as informações que serão usadas para desenhar, por exemplo o shader, os vértices, nela pegamos as posições dos atributos, definimos a localização no shader e além disso, nós fizemos as verificações para quando temos que efetuar uma transformação no objeto.

```
glClearColor(0, 0, 0, 1)
# carrega o shading baseado no nome que foi passado
if(shadingName == 'phong'):
    for i in lista_obj:
        reflectionFlag = 1
        i.ambient = 1
        i.diffuse = 1
        i.specular = 1
        vertex_code = readShaderFile('reflect.vp')
        fragment_code = readShaderFile('reflect.fp')
elif(shadingName == 'smooth'):
    for i in lista_obj:
        if(i.ambient == 1 or i.diffuse == 1 or i.specular == 1):
            # caso tenha sido feito o comando reflection atribuimos 1 para a flag e chamando o shaders das reflections
            reflectionFlag = 1
            vertex_code = readShaderFile('smooth.vp')
            fragment_code = readShaderFile('smooth.fp')
elif(shadingName == 'flat'):
    for i in lista_obj:
        if(i.ambient == 1 or i.diffuse == 1 or i.specular == 1):
            # caso tenha sido feito o comando reflection atribuimos 1 para a flag e chamando o shaders das reflections
            reflectionFlag = 1
            vertex_code = readShaderFile('flat.vp')
            fragment_code = readShaderFile('flat.fp')

else:
    vertex_code = readShaderFile('none.vp')
    fragment_code = readShaderFile('none.fp')



for i in lista_obj: # percorre a lista dos objetos, verificando os atributos das reflectons
    if(i.ambient == 1 or i.diffuse == 1 or i.specular == 1):
        # caso tenha sido feito o comando reflection atribuimos 1 para a flag e chamando o shaders das reflections
```

def drawlight:

- Essa função é responsável por desenhar as luzes na cena, funciona como a init, carregando as informações.

```
vertex_code = readShaderFile('none.vp')
fragment_code = readShaderFile('none.fp')

# compile shaders and program
vertexShader = shaders.compileShader(vertex_code, GL_VERTEX_SHADER)
fragmentShader = shaders.compileShader(fragment_code, GL_FRAGMENT_SHADER)
shaderProgram = shaders.compileProgram(vertexShader, fragmentShader)

# Create and bind the Vertex Array Object
vao = GLuint(0)
glGenVertexArrays(1, vao)
glBindVertexArray(vao)

obj = 'cube'
# lendo os obj pegando vertices e normais
vertices = np.array(lendo_obj(obj), dtype='f')
print("vertices:", len(vertices)//6)


vbo = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBufferData(GL_ARRAY_BUFFER, vertices, GL_STATIC_DRAW)
glVertexAttribPointer(0, 3, GL_FLOAT, False, 6 * sizeof(GLfloat), ctypes.c_void_p(3*sizeof(GLfloat)))  # first 0 is the location in shader
glVertexAttribPointer(1, 3, GL_FLOAT, False, 6 * sizeof(GLfloat), ctypes.c_void_p(0))  # first 0 is the location in shader
glEnableVertexAttribArray(0)
glEnableVertexAttribArray(1)

#verifica em cada objeto a scala
for i in lista_luz:
    i.model = pyrr.matrix44.create_identity()
    scale = pyrr.matrix44.create_from_scale([0.03, 0.03, 0.03],dtype='f')
    i.model = pyrr.matrix44.multiply(i.model,scale)
```

def drawAxis:

- Essa função é responsável por desenhar os eixos, também funciona como a init.

```
vertex_code = readShaderFile('axis.vp')
fragment_code = readShaderFile('axis.fp')

# compila o shaders e program
vertexShader = shaders.compileShader(vertex_code, GL_VERTEX_SHADER)
fragmentShader = shaders.compileShader(fragment_code, GL_FRAGMENT_SHADER)
shaderProgram = shaders.compileProgram(vertexShader, fragmentShader)

# cria e faz o bind no vertex arrat object
vao = GLuint(0)
glGenVertexArrays(1, vao)
glBindVertexArray(vao)

#vertices
x = np.array([[255,0,0], [ 0, 0 ,0], [-255, 0, 0]],dtype='f')
y = np.array([[0,255, 0], [ 0, 0, 0], [0, -255, 0]],dtype='f')
z = np.array([[0 ,0, 255], [ 0, 0, 0], [0, 0, -255]],dtype='f')
vertices = np.concatenate((x,y,z))

vbo = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBufferData(GL_ARRAY_BUFFER, vertices, GL_STATIC_DRAW)
glVertexAttribPointer(0, 3, GL_FLOAT, False, 0, None)  # first 0 is the location in shader
glBindAttribLocation(shaderProgram, 0, 'vertexPosition')
glEnableVertexAttribArray(0);

view = matrix44.create_look_at(poscam, lookat, [0.0, 1.0, 0.0])
projection = matrix44.create_orthogonal_projection(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0)

idView = glGetUniformLocation(shaderProgram, "view")
idProj = glGetUniformLocation(shaderProgram, "projection")

glBindBuffer(GL_ARRAY_BUFFER, 0)
glBindVertexArray(0)
```

Informações adicionais

Implementamos todos os comando, mas com a observação para os comando a seguir:
Implementamos o comando shear, porém tivemos alguns problemas, que não identificamos o motivo, sendo assim não funciona como deveria.
No comando shading flat, tentamos implementar usando GL_FLAT e no vertex shader o flat out vec3 fcolor, para passar a cor para o fragment shader, mas aparentemente está funcionando como o smooth.