Prática 1 – Computação Gráfica – OpenGL

Instalação

1. Instale uma versão do Python inferior à 3.10, infelizmente nossa biblioteca não funciona com as novas versões.
2. Instale as ferramentas de compilação C++ do Visual Studio 2022 (https://visualstudio.microsoft.com/pt-br/downloads/?q=build+tools)
3. Utilizando pip instale os seguintes pacotes:
    a. pip install numpy
    b. pip install cython
    c. pip install pyopengl
    d. pip install triangle
    e. pip install pyglet
    f. pip install glumpy
4. Para verificar que tudo foi instalado corretamente, tente executar o seguinte trecho de código:

```
from glumpy import app, gloo, gl

window = app.Window()
```

Teoria

Nós vamos usar o Glumpy que é um binding do OpenGL em Python. A função dele é facilitar nossa vida, mas ainda assim o OpenGL deve ser executado em C++, então vamos acabar misturando as linguagens. Nós vamos utilizar primitivas do OpenGL que devem ser escritas em C++, essas primitivas são pontos, linhas e triângulos (conforme vimos na teoria). Todos os trechos de código em C++ devem estar entre aspas triplas (""") e cada um deles terá uma main (void main).

Glossário

attribute vec2 position;	Struct que armazena um par ordenado
gl_Position = vec4(position, 0.0, 1.0);	Cria uma struct com 4 componentes (coordenadas homogêneas). Essa struct armazena a posição do fragmento (ponto).
gl_FragColor = vec4(vermelho, verde, azul, alfa);	Configura a cor de um fragmento (ponto).
gl_PointSize	Define o tamanho de um ponto (ponto flutuante).
varying vec4 vColor	Configura um valor de cor que pode mudar.
on_init	Função que configura a forma de renderização da tela.

gl.glEnable(gl.GL_BLEND)	Habilita renderização com blend de cores.
gl.glBlendFunc(gl.GL_SRC_ALPHA, gl.GL_ONE_MINUS_SRC_ALPHA)	Define os parâmetros do blend.

Nós precisamos criar algumas definições para dizer o que é um vértice e um fragmento.

```
vertex = """
    attribute vec2 position;
    void main()
    {
        gl_Position = vec4(position, 0.0, 1.0);
    }
"""

fragment = """
    void main()
    {
        gl_FragColor = vec4(1, 0, 0, 1); // Vermelho
    }
"""
```

Todos os nossos códigos terão uma função de setup. Nela nós vamos definir os valores numéricos dos vértices e passar eles para o programa em C++, isso é feito usando o dicionário “program”. Por fim nós precisamos configurar o programa para o modo de desenho usando a variável global draw_mode e dizendo o que queremos desenhar.

```
def setup():
    vertices = np.array([(-0.5, -0.5), (0.5, -0.5), (-0.5, 0.5),
                         (0.5, 0.5), (-0.75, 0.75), (0.75, 0.75)], dtype=np.float32)
    program['position'] = vertices
    global draw_mode
    draw_mode = gl.GL_POINTS
```

Não adianta desenhar sem criar uma janela, então precisamos usar a biblioteca Glumpy para gerar uma janela apropriada. Para usar Glumpy, devemos importa-la no começo do código, assim como Numpy.

```
import numpy as np
from glumpy import app, gloo, gl, glm
```

Para criar a janela, usamos o objeto app e o método Window. Devemos informar as coordenadas e cores da janela. Vamos criar uma janela completamente branca, para facilitar a visualização.

```
window = app.Window(width=720, height=480, color=(1, 1, 1, 1))
```

Agora precisamos criar uma função para realmente desenhar. Para adicionar o contexto da janela em qualquer função que irá interagir com ela devemos adicionar @window.event antes das funções criadas. A função abaixo limpa a janela e desenha a primitiva configurada em draw_mode.

```
@window.event
def on_draw(dt):
    window.clear()
    program.draw(draw_mode)
```

Para finalizar o código precisamos executá-lo. Isso é feito usando o trecho de código abaixo. A primeira função inicializa o programa, informando quais os vértices, fragmentos e quantos vértices esperamos no código.

```
program = gloo.Program(vertex, fragment, count=6)
setup()
app.run()
```

Exercícios:

1) Troque as primitivas renderizadas para Linhas e Triângulos. Quando fizer isso, reduza o tamanho dos pontos.
2) Mude a cor das primitivas.
3) Crie uma função on_init no contexto da janela. A função “on_init” deve habilitar a renderização em blend usando as duas funções do glossário. Faça com que as cores dos triângulos mudem suavemente entre os vértices. Para isso você vai precisar adicionar uma característica de cor variante no vertex e a mesma característica em fragment. Além disso em setup, precisamos de um vetor de array com 6 cores (uma para cada vértice) e precisamos passar essa característica através do dicionário program[‘color’].