Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Programação Gráfica Interativa

BCC327 - Computação Gráfica

Matheus Peixoto Ribeiro Vieira - 22.1.4104
Professor: Rafael Alves Bonfim de Queiroz

Ouro Preto
22 de novembro de 2024

Sumário

Lista de Figuras

Lista de Códigos Fonte

1 Introdução

Para este trabalho, é solicitado uma aplicação para explorar conceitos de programação gráfica interativa como eventos, manipulação em tempo real e interatividade.

1.1 Especificações da máquina

A máquina onde o desenvolvimento foi realizado possui a seguinte configuração:

- Processador: Intel Core i5-9300H
- Memória RAM: 16Gb.
- Sistema Operacional: WSL 2.0 com Ubuntu 22.04.5 LTS

2 Objetivo

Para explorar os conceitos de eventos, manipulação em tempo real e interatividade dois projetos são propostos, o primeiro deles é a mudança de cores do fundo da janela onde os dados estão sendo desenhados.

Dessa forma, na primeira atividade, o fundo da janela possuía a cor branca. Agora, isso será incrementado, onde, ao apertar o botão esquerdo do mouse, a cor irá mudar, variando entre branco, vermelho, verde e azul, respectivamente.

Ademais, seguindo os triângulos que foram gerados na primeira atividade e que podem ser vistos na figura 1, o triângulo azul será movido a partir das teclas WASD enquanto que o triângulo laranja será movimentado com as setas para cima, esquerda, baixo e direita.

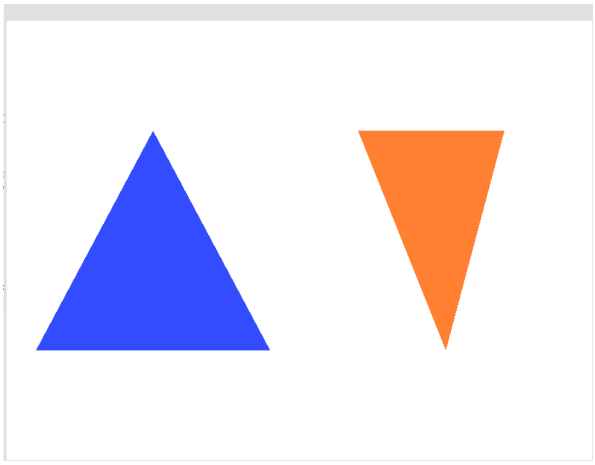

Figura 1: Triângulos da atividade 1

3 Modificando o fundo

Para armazenar a cor atual do fundo da janela, foi criada uma struct chamada de `BackgroundColor` que possui valores do tipo float para as variáveis r,g & b, representando os valores dos canais de cores. A struct é inicializada com todos os valores com valor de 1.0, indicando, assim, que temos a cor branca sendo exibida.

Um evento para o clique do botão esquerdo do mouse é feito com o uso de callbacks, como é explicado em [2]. Dessa forma, sempre que tal evento ocorrer a função do trecho de código 1 responsável por ajustar as cores será chamada.

A função inicia uma variável de contador que é estática e que, durante toda a vida do programa, irá manter o seu valor até que seja modificada, o que ocorre de acordo com o clique atual do mouse.

```
void mouseClickEvent(GLFWwindow* window, int button, int action, int mods){
    static int background = 0;
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
        background = (background + 1) % 4;

    cout << "Background value: " << background << endl;

    if      (background == 0){colors.r = 1.0, colors.g = 0.0; colors.b = 0.0;}
         // Vermelho
    else if (background == 1){colors.r = 0.0; colors.g = 1.0; colors.b = 0.0;}
         // Verde
    else if (background == 2){colors.r = 0.0; colors.g = 0.0; colors.b = 1.0;}
         // Azul
    else                     {colors.r = 1.0; colors.g = 1.0; colors.b = 1.0;}
         // Branco
    std::this_thread::sleep_for( std::chrono::microseconds( 100000 ) );
}
```

Código 1: Mudar cor de fundo no clique do mouse

Para que de fato ocorra a mudança das cores, é necessário ir para o loop de exibição principal. Nele, enquanto a janela estiver aberta a seguinte função `glClearColor` sempre é chamada, sendo esta responsável por limpar os *bufferes* de cores que receberão os valores de vermelho, verde, azul e da camada alpha sendo passados [5]. Assim, mudando o valor dessas camadas, conseguimos modificar as cores de fundo da janela, como ocorre no código 2.

```
void loopDeExibicao(GLFWwindow* window, unsigned int* shaderPrograms, unsigned
     int* VAOs) {
    while (!glfwWindowShouldClose(window)) {
        ...
        glClearColor(colors.r, colors.g, colors.b, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        ...
    }
}
```

Código 2: Loop de Exibição principal mudando o fundo

Dessa forma, conseguimos os resultados que são exibidos nas figuras 2, 3, 4 e 5

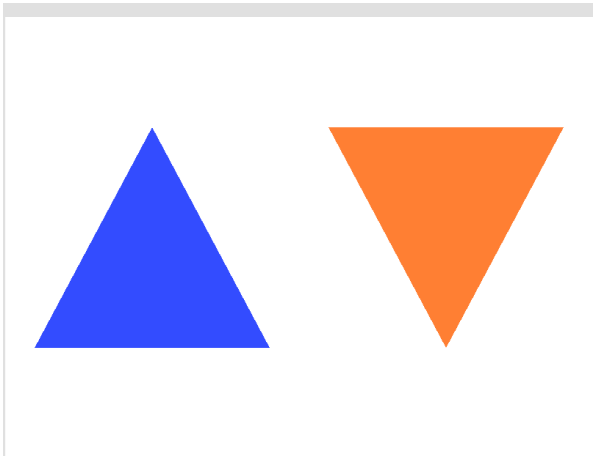

Figura 2: Fundo Branco

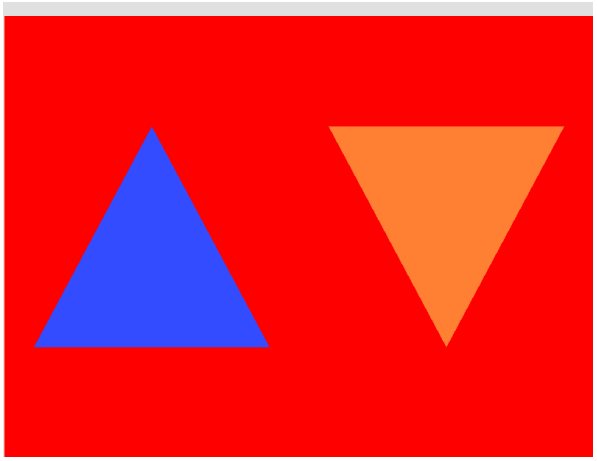

Figura 3: Fundo Vermelho

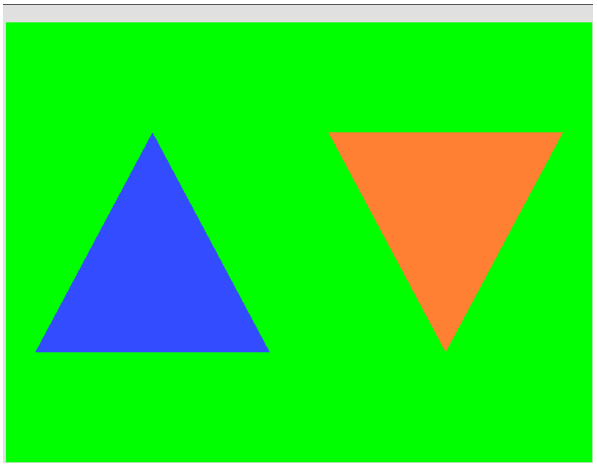

Figura 4: Fundo Verde

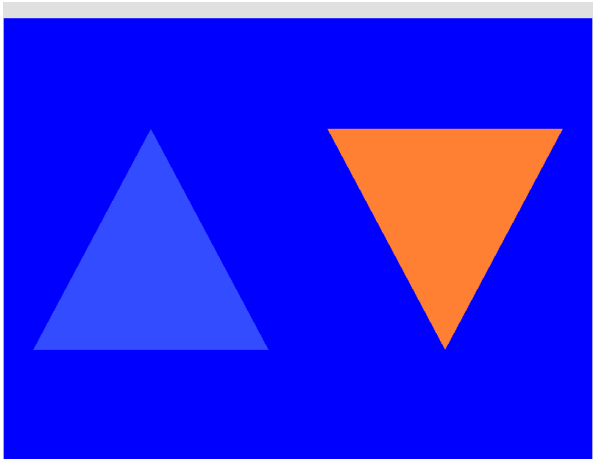

Figura 5: Fundo Azul

4 GLM

Os *shaders* do OpenGL são escritos na linguagem GLSL e, estes pequenos códigos, são compilados durante a execução do programa principal [6].

Dessa forma, a biblioteca GLM foi instalada, pois ela possibilita a escrita de códigos de GLSL em C++ [1]. Assim, tal biblioteca foi instalada a partir do comando `sudo apt install libglm-dev` [7].

5 Movimentando triângulos

Para que os triângulos possam ser movimentados, dois vetores tri-dimensionais foram criados e chamados de `offset`, sendo estes do tipo vec3 da biblioteca glm, sendo instanciados como se observa no código 3.

```
glm::vec3 offset = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 offset2 = glm::vec3(0.0f, 0.0f, 0.0f);
```

Código 3: Offsets

No código GLSL do vertex shader modificações simples foram necessárias, sendo preciso somente acrescentar o uso do offset para determinar a posição de um item. Dessa forma, o local onde um objeto estará sendo localizado é a sua coordenada informada mais o seu novo valor, como segue no código 4.

```
version 420 core
layout (location = 0) in vec3 aPos;
uniform vec3 offset;
void main(){
    gl_Position = vec4(aPos.x + offset.x, aPos.y + offset.y, aPos.z + offset.z
        , 1.0);
}\0
```

Código 4: Vertex Shader com offset

No laço de repetição principal para exibir as figuras, primeiro temos a chamada da função `processInput`, responsável por capturar as teclas do teclado e modificar os valores dos offsets, como pode ser visto no código 5.

```
void processInput(GLFWwindow *window){
    if(glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);

    // Triangulo 1
    if (glfwGetKey(window, GLFW_KEY_W)     == GLFW_PRESS) offset.y += 0.01f;
    if (glfwGetKey(window, GLFW_KEY_S)     == GLFW_PRESS) offset.y -= 0.01f;
    if (glfwGetKey(window, GLFW_KEY_A)     == GLFW_PRESS) offset.x -= 0.01f;
    if (glfwGetKey(window, GLFW_KEY_D)     == GLFW_PRESS) offset.x += 0.01f;

    // Triangulo 2
    if (glfwGetKey(window, GLFW_KEY_UP)    == GLFW_PRESS) offset2.y += 0.01f;
    if (glfwGetKey(window, GLFW_KEY_DOWN)  == GLFW_PRESS) offset2.y -= 0.01f;
    if (glfwGetKey(window, GLFW_KEY_LEFT)  == GLFW_PRESS) offset2.x -= 0.01f;
    if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS) offset2.x += 0.01f;
}
```

Código 5: Capturar Inputs do usuário

Dessa forma, o OpenGL irá, no loop principal, utilizar o shader de cada triângulo para exibí-lo na tela a partir do `glUseProgram`. Em seguida, o `glGetUniformLocation` recupera a localização do offset em um shader determinado [3]. Com esse valor, então é possível modificar essa variável, que é do tipo 1 (float), usando os valores do offset correspondente para determinar a nova posição do item, isso tudo com a função `glUniform3fv` [4], finalizando com o bind do VAO para permitir a exibição e o desenho dos triângulos.

Por fim, agrupando tudo, temos no código 6 o loop principal que possibilita a movimentação de ambos os triângulos de forma independente, como pode-se observar na Figura 6.

```
void loopDeExibicao(GLFWwindow* window, unsigned int* shaderPrograms, unsigned
     int* VAOs) {
    while (!glfwWindowShouldClose(window)) {
        processInput(window);
```

```
4
5           glClearColor(colors.r, colors.g, colors.b, 1.0f);
6           glClear(GL_COLOR_BUFFER_BIT);
7
8           //Triangulo 1
9           glUseProgram(shaderPrograms[0]);
10          glUniform3fv(glGetUniformLocation(shaderPrograms[0], "offset"), 1, &
                offset[0]);
11          glBindVertexArray(VAOs[0]);
12          glDrawArrays(GL_TRIANGLES, 0, 3);
13
14          // Triangulo 2
15          glUseProgram(shaderPrograms[1]);
16          glUniform3fv(glGetUniformLocation(shaderPrograms[1], "offset"), 1, &
                offset2[0]);
17          glBindVertexArray(VAOs[1]);
18          glDrawArrays(GL_TRIANGLES, 0, 3);
19
20          glfwSwapBuffers(window);
21          glfwPollEvents();
22      }
23  }
```

Código 6: Loop de exibição

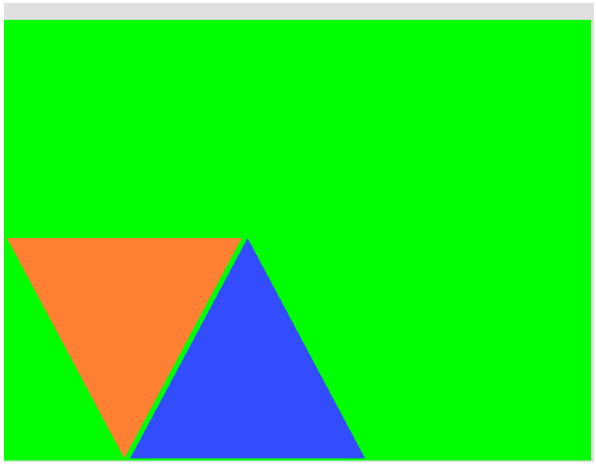

Figura 6: Triângulos movidos separadamente

Para a compilação e execução do programa a seguinte diretiva é executada:

```
g++ main.cpp glad.c -o exec -lGL -lGLU -lglfw3 -lX11 -lXxf86vm -lXrandr -lpthread -lXi
./exec
```

6 Conclusão

Com o fim da atividade, foi possível verificar os conhecimentos sobre eventos, como os que foram feitos para lidar com o clique do mouse e algumas teclas do teclado gerando um código interativo ao mesmo tempo em que tudo estava sendo modificado e executado em tempo real, como era esperado para tal trabalho.

Referências

[1] G-Truck Creation. Glm. https://github.com/g-truc/glm. [Online; acessado em 22-Novembro-2024].

[2] GLFW. Input guide. https://www.glfw.org/docs/3.3/input_guide.html#input_mouse_button. [Online; acessado em 22-Novembro-2024].

[3] Khronos. glgetuniformlocation. https://registry.khronos.org/OpenGL-Refpages/gl4/html/glGetUniformLocation.xhtml. [Online; acessado em 22-Novembro-2024].

[4] Khronos. gluniform3fv. https://registry.khronos.org/OpenGL-Refpages/gl4/html/glUniform.xhtml. [Online; acessado em 22-Novembro-2024].

[5] Microsoft. Função glclearcolor. https://learn.microsoft.com/pt-br/windows/win32/opengl/glclearcolor. [Online; acessado em 22-Novembro-2024].

[6] Learn OpenGL. Hello triangle. https://learnopengl.com/Getting-started/Hello-Triangle. [Online; acessado em 22-Novembro-2024].

[7] Stack Overflow. How to install glm-math on ubuntu 16.04? glm 0.9.9 a2-2. https://stackoverflow.com/questions/52156449/how-to-install-glm-math-on-ubuntu-16-04-glm-0-9-9a2-2. [Online; acessado em 22-Novembro-2024].