

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

1	Introdução	1
1.1	Especificações da máquina	1
2	Objetivo	1
3	Modificando o fundo	1
4	GLM	3
5	Movimentando triângulos	4

Lista de Figuras

1	Triângulos da atividade 1	1
2	Fundo Branco	2
3	Fundo Vermelho	3
4	Fundo Verde	3
5	Fundo Azul	3
6	Triângulos movidos separadamente	5

Lista de Códigos Fonte

1	Mudar cor de fundo no clique do mouse	2
2	Loop de Exibição principal mudando o fundo	2
3	Offsets	4
4	Vertex Shader com offset	4
5	Capturar Inputs do usuário	4
6	Loop de exibição	4

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.

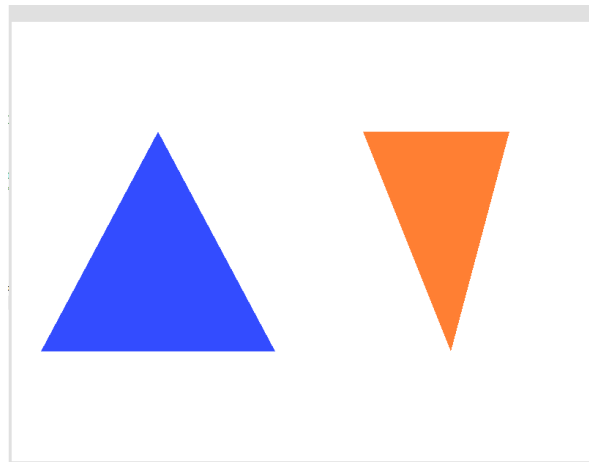


Figura 1: Triângulos da atividade 1

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.

```

1 void mouseClickedEvent(GLFWwindow* window, int button, int action, int mods){
2     static int background = 0;
3     if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
4         background = (background + 1) % 4;
5
6     cout << "Background value: " << background << endl;
7
8     if (background == 0){colors.r = 1.0, colors.g = 0.0; colors.b = 0.0;}
9         // Vermelho
10    else if (background == 1){colors.r = 0.0; colors.g = 1.0; colors.b = 0.0;}
11        // Verde
12    else if (background == 2){colors.r = 0.0; colors.g = 0.0; colors.b = 1.0;}
13        // Azul
14    else {colors.r = 1.0; colors.g = 1.0; colors.b = 1.0;}
15        // Branco
16    std::this_thread::sleep_for( std::chrono::microseconds( 100000 ) );
17 }

```

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.

```

1 void loopDeExibicao(GLFWwindow* window, unsigned int* shaderPrograms, unsigned
2     int* VAOs) {
3     while (!glfwWindowShouldClose(window)) {
4         ...
5         glClearColor(colors.r, colors.g, colors.b, 1.0f);
6         glClear(GL_COLOR_BUFFER_BIT);
7         ...
8     }
9 }

```

Código 2: Loop de Exibição principal mudando o fundo

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

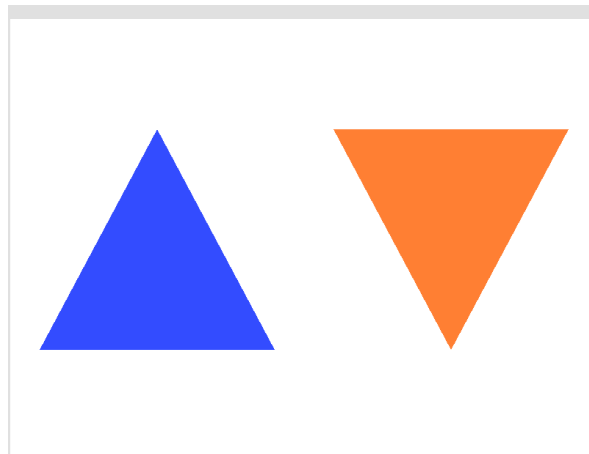


Figura 2: Fundo Branco

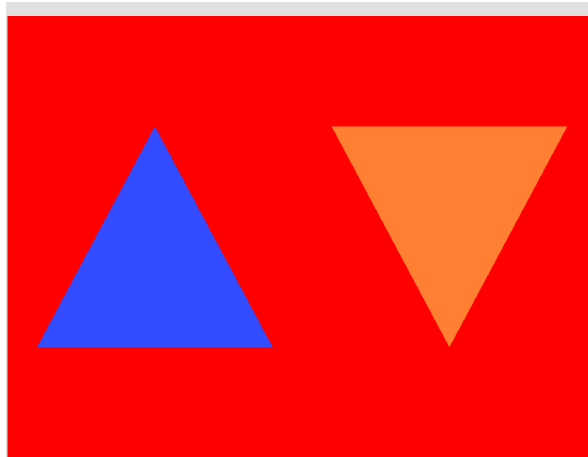


Figura 3: Fundo Vermelho

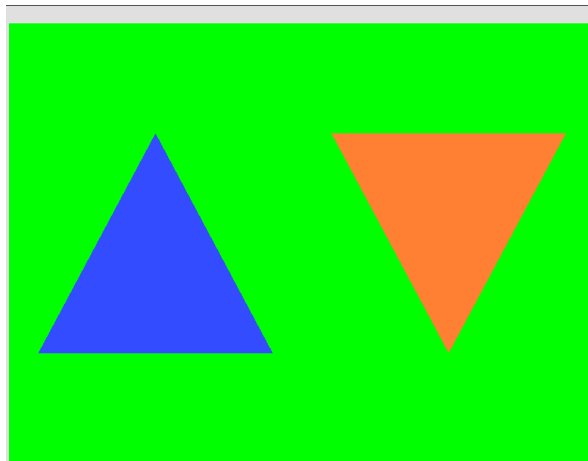


Figura 4: Fundo Verde

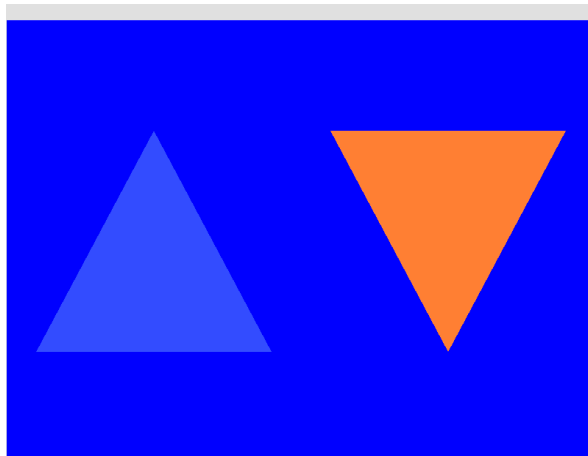


Figura 5: Fundo Azul

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.

```
1 glm::vec3 offset = glm::vec3(0.0f, 0.0f, 0.0f);
2 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.

```
1 version 420 core
2 layout (location = 0) in vec3 aPos;
3 uniform vec3 offset;
4 void main(){
5     gl_Position = vec4(aPos.x + offset.x, aPos.y + offset.y, aPos.z + offset.z
6     , 1.0);
7 }
```

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.

```
1 void processInput(GLFWwindow *window){
2     if(glwfGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
3         glfwSetWindowShouldClose(window, true);
4
5     // Triangulo 1
6     if (glwfGetKey(window, GLFW_KEY_W) == GLFW_PRESS) offset.y += 0.01f;
7     if (glwfGetKey(window, GLFW_KEY_S) == GLFW_PRESS) offset.y -= 0.01f;
8     if (glwfGetKey(window, GLFW_KEY_A) == GLFW_PRESS) offset.x -= 0.01f;
9     if (glwfGetKey(window, GLFW_KEY_D) == GLFW_PRESS) offset.x += 0.01f;
10
11    // Triangulo 2
12    if (glwfGetKey(window, GLFW_KEY_UP) == GLFW_PRESS) offset2.y += 0.01f;
13    if (glwfGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) offset2.y -= 0.01f;
14    if (glwfGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS) offset2.x -= 0.01f;
15    if (glwfGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS) offset2.x += 0.01f;
16 }
```

Código 5: Capturar Inputs do usuário

Dessa forma, o OpenGL irá, no loop principal, utilizar o shader de cada triângulo para exibi-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.

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

```

4
5     glClearColor(colors.r, colors.g, colors.b, 1.0f);
6     glClear(GL_COLOR_BUFFER_BIT);
7
8     //Triângulo 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    // Triângulo 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

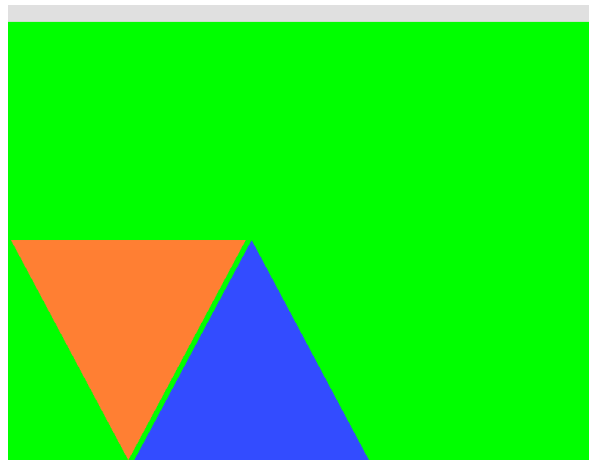


Figura 6: Triângulos movidos separadamente

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

```

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].