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

Introdução aos Sistemas Gráficos: Bibliotecas Gráficas

BCC327 - Computação Gráfica

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

Ouro Preto
16 de novembro de 2024

Sumário

Lista de Figuras

Lista de Códigos Fonte

1 Introdução

Para este trabalho, é solicitado uma aplicação para explorar conceitos iniciais da computação gráfica usando uma das seguintes bibliotecas: OpenGL, WebGL ou DirectX

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 Bibliotecas

Para a escolha de qual biblioteca será explorada no decorrer do trabalho, primeiramente é necessário ter um breve conhecimento sobre elas e suas características.

2.1 Open GL

Open GL é uma Interface de Programação de Aplicação (API) utilizada para a criação e renderização de objetos gráficos 2D e 3D, podendo ser utilizada em diferentes plataformas, tais como o Linux e Windows. [12]

O seu uso pode ser encontrado em aplicações de projeto e desenho assistido por computador (CAD), desenvolvimento de jogos, realidade virtual e áreas médicas. [2]

Dessa forma, devido à sua natureza de ser multi-plataforma e, pela natureza do curso de Ciência da Computação, esta biblioteca será escolhida, pois é compatível com sistemas Linux e permite uma programação direta em C/C++.

2.2 Web GL

O Web GL é uma API para renderização de de objetos gráficos 2D e 3D em navegadores web, dessa forma, utiliza o 'canva' do HTML para realizar a renderização, sendo que esta biblioteca é baseada no Open GL ES e possui integração com outros elementos da página. [4]

2.3 DirectX

O DirectX é um conjunto de APIs para tarefas de multimídia, sendo utilizada exclusivamente em plataformas da Microsoft, sendo estes o Windows e o Xbox, sendo esta uma das principais diferenças em relação ao OpenGL. [8, 11]

Como o DirectX não é um produto completo, mas sim um conjunto, ele possui diferentes bibliotecas para diferentes funcionalidades, como o DirectDraw para gráficos 2D, o Direct3D para gráficos em 3D, DirectSound para áudio, DirectPlay para jogos multi jogadores, etc. [8]

3 Configurações iniciais

Para iniciar com o OpenGL, primeiro ele foi instalado no sistema, dessa forma foi utilizado o seguinte pacote para baixar sua biblioteca para C/C++ em um sistema Ubuntu:

```
sudo apt-get install freeglut3-dev
```

Em seguida, foi feito um simples código para verificar a instalação do mesmo, onde foi gerado um círculo verde em um quadro preto, como pode ser observado na Figura 1. Sendo que foi utilizado o seguinte comando para a compilação do código:

```
gcc teste.c -lGL -lGLU -lglut -lm -o teste
```

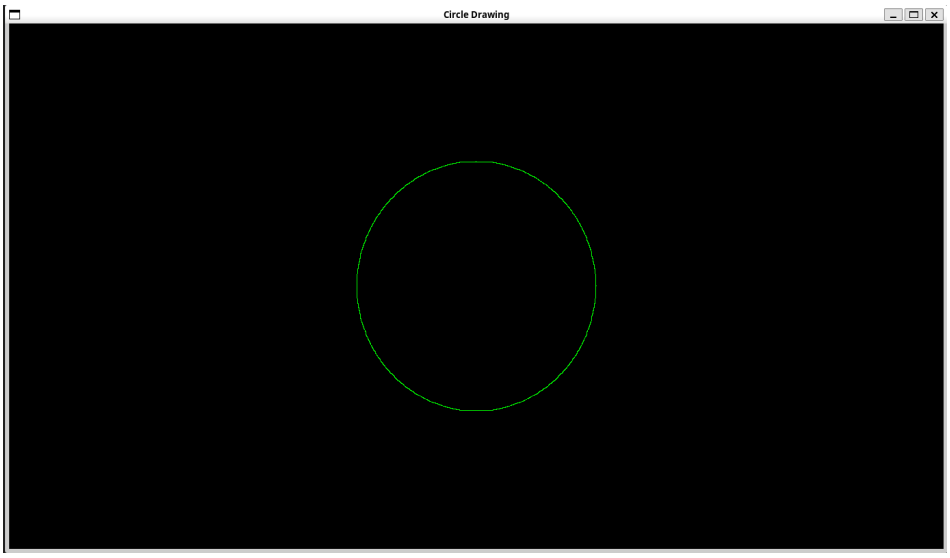


Figura 1: Saída do código de teste

É importante ressaltar que o passo a passo informado acima foi retirado integralmente de [3]

Seguindo os passos iniciais do `learnopengl.com` para primeiros passos com o OpenGL, é recomendado a instalação do GLFW para facilitar a criação de janelas, principalmente para quando o código se tornará multi plataforma, uma vez que o Linux e Windows, por exemplo, geram janelas de formas diferentes.

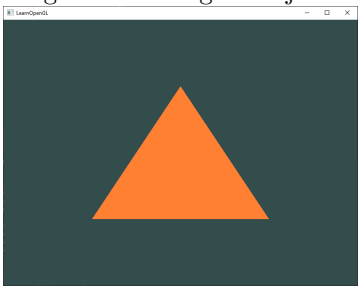
Todavia, após um certo tempo tentando, de forma frustrada, a instalação pelo tutorial oficial, foi possível instalar a biblioteca a partir do seguinte passo a passo postado no fórum do Stack Overflow [7].

Por fim, para finalizar a preparação, foi necessário instalar o GLAD, uma biblioteca open-source para facilitar o acesso, em tempo de execução, de funções mais específicas para o driver da placa de hardware. Assim, foi seguido o passo a passo mostrado em [9]

4 Janelas e triângulos

Como proposta de uma atividade inicial, serão seguidos os tutoriais do learnopengl.com, dessa forma, os introdutórios serão o *Hello Window* [5] e o *Hello Triangle* [6]. Dessa forma, então, o objetivo final será uma aplicação simples que exibe na tela uma imagem de um triângulo, semelhante ao que se tem na Figura 2.

Figura 2: Triângulo objetivo



Fonte: [6]

Dessa forma, começando o código, primeiramente é necessário inicializar o GLFW, definindo que será usada uma versão 3.X do OpenGL e depois define que a janela terá decorações, como o nome da mesma.

Em seguida, como mostrado no código 1, criamos uma janela que tem proporções de 800x600, sendo este valores definidos como constantes no código e, depois definimos o contexto dela como principal para a thread atual e é criado um callback para mudanças no tamanho da janela.

```
GLFWwindow* instanciarJanela(){
    GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Introducao ao OpenGL
        ", NULL, NULL);
    if (window == NULL){
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return NULL;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    return window;
}
```

Código 1: Instância da janela

A cada passo para realizar a renderização de um objeto, os dados são transferidos para pequenos programas chamados de shaders [6].

O OpenGL necessita que pelo menos um Vertex Shader seja definido, uma vez que ele não possui um padrão. Assim, seu papel é realizar o processamento de cada um dos vértices de um modelo 3D antes que eles sejam passados para as próximas camadas, sendo este o responsável para transformar as coordenadas locais em coordenadas na tela [10].

Assim, definimos um simples *shader*, em linguagem GLSL, que irá calcular a posição dos dados no espaço 2D, uma vez que todas as informações do OpenGL são definidas em 3D, mas a tela possui somente duas dimensões. Em seguida, o código deste *shader* é compilado, como pode ser observado no código 2

```
unsigned int compilarVertexShader(){
    const char *vertexShaderSource = "#version 420 core\n"
        "layout (location = 0) in vec3 aPos;\n"
        "void main()\n"
        "{\n"
        "   gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"
        "}\0";
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    return vertexShader;
}
```

Código 2: VertexShader

Para o projeto, é desejado a criação de um triângulo, então, para isso, é necessário calcular a cor dos pixeis que farão parte desse desenho, onde após a rasterização, eles serão agrupados em fragmentos e estes terão suas cores calculadas pelo shader de fragmento [1], que é definido e compilado na função do código 3, e é vinculado ao shader de vertex em sequência.

```
unsigned int compilarFragmentShader(unsigned int vertexShader ,string
    vec4_info){
    string fragmentShaderSource = "#version 420 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = "+ vec4_info +";\n"
    "}\n\0";
    const char* source = fragmentShaderSource.c_str();
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
```

```
    glShaderSource(fragmentShader, 1, &source, NULL);
    glCompileShader(fragmentShader);
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
    glDeleteShader(fragmentShader);
    return shader
}
```

Código 3: FragmentShader

Foi decidido que, ao invés de um triângulo, dois seriam exibidos na tela, sendo um azul e o outro laranja. Dessa forma, os seus shaders de fragmentos são criados passando os valores que correspondem a essas cores pelo código 4. E, em sequência, suas posições na tela são definidas a partir das coordenadas de dispositivos normalizadas, que variam de [-1, 1] e, depois são convertidas em coordenadas de tela pelos shaders.

```
    unsigned int shaderProgram_1 = compilarFragmentShader(vertexShader, "vec4
        (0.2f, 0.3f, 1.0f, 1.0f)");
    unsigned int shaderProgram_2 = compilarFragmentShader(vertexShader, "vec4
        (1.0f, 0.5f, 0.2f, 1.0f)");
    unsigned int shaderPrograms[2] = {shaderProgram_1, shaderProgram_2};

    float triangles [2][9] = {
            // X     Y     Z
        {
            -0.9f, -0.5f, 0.0f, // Esquerdo inferior
            -0.1f, -0.5f, 0.0f, // Direito inferior
            -0.5f,  0.5f, 0.0f  // Topo
        },
        {
            0.2f, 0.5f, 0.0f, // Esquerdo inferior
            0.7f, 0.5f, 0.0f, // Direito inferior
            0.5f, -0.5f, 0.0f  // Topo
        }
    };
}
```

Código 4: Fragmentos

O VBO (Vertex Buffer Object) é um buffer de memória que o OpenGL utiliza para que a placa gráfica possa utilizar os dados dos vertexes. Por outro lado, o VBA (Vertex Array Object) é um contêiner de VBOs e armazena informações para que os objetos possam ser devidamente renderizados.

Dessa forma, eles são criados os triângulos são conectados a eles para que possam ser exibidos, sendo possível observar isto no código 5

```
int main(){
...
    unsigned int VAOs[2];
    glGenVertexArrays(2, VAOs);
    unsigned int VBOs[2];
    glGenBuffers(2, VBOs);
    consfigurarDesenhos(VAOs, VBOs, triangles);
...
}
void consfigurarDesenhos(unsigned int *VAOs, unsigned int *VBOs, float
    triangles[2][9]){
    for(int i = 0; i < 2; i++){
        glBindVertexArray(VAOs[i]);
        glBindBuffer(GL_ARRAY_BUFFER, VBOs[i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(triangles[i]), triangles[i],
            GL_STATIC_DRAW);
```

```
15          glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (
                void*)0);
16          glEnableVertexAttribArray(0);
17      }
18  }
```

Código 5: VBOs e VBAs

Por fim, é feito um loop de exibição onde todos os desenhos são feitos e exibidos na tela para o usuário, obtendo o resultado final que pode ser observado na figura 3.

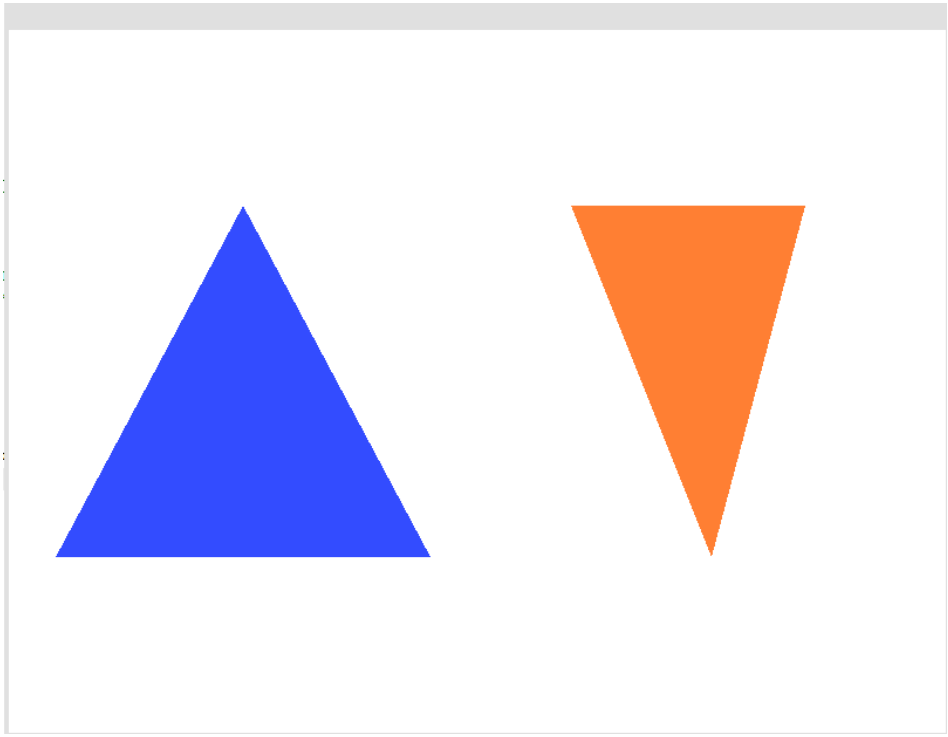


Figura 3: Resultado Final

Para a compilação do programa a seguinte diretiva é executada:

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

5 Conclusão e comentários finais

Com o fim da atividade, foi possível ter uma visão breve sobre diferentes bibliotecas gráficas e um aprendizado inicial com o OpenGL. Em geral, tive dificuldades para a instalação da biblioteca para ter o funcionamento esperado, mas ele foi obtido.

Em um primeiro contato, há muitos termos novos e que geram uma certa dificuldade para o entendimento do que está acontecendo no código, sendo necessárias pausas no tutorial para tentar entender mais a fundo o que estava de fato acontecendo, pois muitos dos assuntos são novidades e ainda serão abordados na disciplina.

Referências

[1] LBO Dev. O que é fragment shader. https://lbodev.com.br/glossario/o-que-e-fragment-shader/. [Online; acessado em 16-Novembro-2024].

[2] Khronos Group. The industry's foundation for high performance graphics. https://www.khronos.org/opengl/. [Online; acessado em 15-Novembro-2024].

[3] Aditya Kumar. Getting started with opengl. https://www.geeksforgeeks.org/getting-started-with-opengl/, 2023. [Online; acessado em 15-Novembro-2024].

[4] Mozilla. Começando com webgl. https://developer.mozilla.org/pt-BR/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL. [Online; acessado em 15-Novembro-2024].

[5] Learn OpenGL. Hello window. https://learnopengl.com/Getting-started/Hello-Window. [Online; acessado em 15-Novembro-2024].

[6] Learn OpenGL. Hello window. https://learnopengl.com/Getting-started/Hello-Triangle. [Online; acessado em 16-Novembro-2024].

[7] Stack Overflow. How to build and install glfw 3 and use it in a linux project. https://stackoverflow.com/questions/17768008/how-to-build-install-glfw-3-and-use-it-in-a-linux-project. [Online; acessado em 15-Novembro-2024].

[8] PCMag. Começando com webgl. https://www.pcmag.com/encyclopedia/term/directx. [Online; acessado em 15-Novembro-2024].

[9] Priyanshu. Opengl setup: Guide to install opengl in ubuntu. https://medium.com/geekculture/a-beginners-guide-to-setup-opengl-in-linux-debian-2bfe02ccd1e, Mar 2021. [Online; acessado em 15-Novembro-2024].

[10] Shader Tutorial. Shader basics - vertex shader. https://shader-tutorial.dev/basics/vertex-shader/. [Online; acessado em 16-Novembro-2024].

[11] Wikipedia. DirectX — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=DirectX&oldid=1251154793, 2024. [Online; acessado em 15-November-2024].

[12] Wikipedia. OpenGL — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=OpenGL&oldid=1254242372, 2024. [Online; acessado em 15-Novembro-2024].