Atividade Prática 3 - Implementação do Pipeline Gráfico

GDSCO0051 - Introdução à Computação Gráfica - Turma 02 - 2019.4

Data de entrega: 29/07/2020, 23h59min.

1 Objetivo

O objetivo deste trabalho é familiarizar os alunos com a estrutura do *pipeline* gráfico através da implementação das transformações geométricas que o compõem. Esta implementação será feita com auxílio da biblioteca `glm` e sua execução ocorrerá nos *shaders* do OpenGL.

2 Desenvolvimento

2.1 Código Template

Esta atividade de implementação será feita a partir de um código template `C++17` disponibilizado pelo professor e que está disponível no repositório da disciplina:

```
https://github.com/capagot/icg/tree/master/03_transformations
```

2.2 Dependências

O código template disponibilizado depende de software de terceiros. As seções a seguir apresentam cada uma destas dependências.

2.2.1 `glm`

Este programa depende da biblioteca `glm` (`https://glm.g-truc.net/0.9.9/index.html`). Para facilitar a instalação desta dependência, ela foi inclusa no projeto do Github na forma de um submódulo. Assim, ao se fazer um clone do projeto, a `glm` pode ser instalada automaticamente.

Para clonar o projeto, e fazer *dowload* automático do submódulo da `glm`, o seguinte comando deve ser executado:

```
$ git clone --recurse-submodules https://github.com/capagot/icg.git
```

A opção `clone` fará uma cópia local do repositório enquanto a opção `--recurse-submodules` fará com que o conteúdo de todos os submódulos do projeto sejam inclusos na cópia local.

2.2.2 `GLEW`

Enquanto a linguagem `C` aceita declarações implícitas de funções, a linguagem `C++` não aceita. Assim, para que um programa `C++`, que utilize o OpenGL moderno, possa ser compilado com sucesso, todas as extensões OpenGL devem estar declaradas de forma explícita. A biblioteca `GLEW` tem a função de gerar as referências para todas as extesões OpenGL. O programa template disponibilizado faz uso da `GLEW` para a geração destas referências. Os procedimentos para a instalação da `GLEW` variam de acordo

com o sistema operacional. Maiores detalhes sobre como instalar a `GLEW` podem ser obtidos em seu site oficial: `http://glew.sourceforge.net`.

Existem outras bibliotecas para carga das extensões OpenGL, como por exemplo a `GLAD`, disponível em `https://glad.dav1d.de`. Os alunos estão livres para utilizarem outras bilbioteca para a carga das extensões OpenGL. Lembramos apenas que, neste caso, como o código disponibilizado pelo professor foi baseado na `GLEW`, ele deverá ser adaptado de acordo.

2.3 Compilação e Execução do Código Template

Uma vez que todas as dependências estejam satisfeitas, pode-se passar para a compilação do código template. O comando de compilação em sistemas Linux deverá ser:

```
$ g++ -Wall -Wextra -Wpedantic -std=c++17 -O0 -g3 -DDEBUG main.cpp -lglut -lGLEW -lGLU
-lGL -o transform_gl
```

Como este comando já se encontra descrito no arquivo `Makefile` disponibilizado no projeto, basta apenas emitir o seguinte comando a partir do diretório raíz deste programa:

```
$ make
```

Durante a compilação alguns *warnings* poderão aparecer na tela. Uma vez compilado, ao ser executado, o programa deverá apresentar uma janela contendo dois triângulos coloridos, como ilustra a Figura 1:

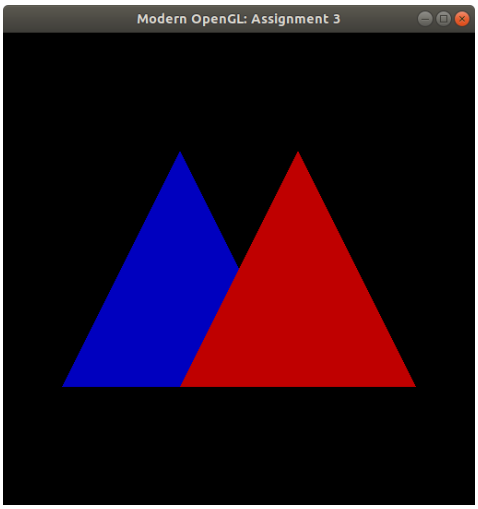

Figure 1: Janela OpenGL criada durante a execução do programa.

2.4 A Cena

A cena é composta por dois triângulos (vermelho e azul), sendo que, em relação à câmera, o triângulo vermelho está ligeiramente à frente do triângulo azul. A configuração geométrica da cena renderizada na Figura 1 pode ser visualizada nas Figuras 2, 3 e 4.

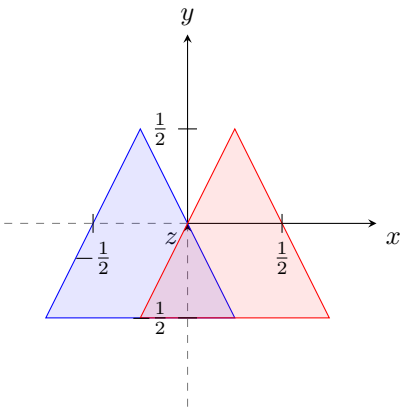

Figure 2: Plano XY (eixo Z negativo para dentro do plano da imagem.)

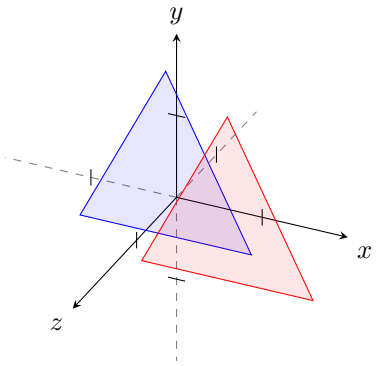

Figure 3: Visualização dos eixos XYZ.

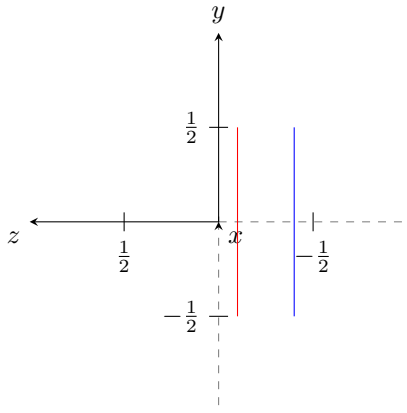

Figure 4: Plano ZY (eixo X negativo para dentro do plano da imagem.)

2.5 A Atividade

A atividade consiste em se alterar, no programa template, os conteúdos das matrizes M_{Model}, M_{View} e $M_{Projection}$ de forma que o programa gere as imagens abaixo. Observe que antes de realizar cada exercício o aluno deve se certificar que as matrizes do programa (*i.e.* M_{Model}, M_{View} e $M_{Projection}$) contém a identidade! A única exceção a esta regra é o exercício 4 (mais detalhes no enunciado do Exercício 4)!

2.5.1 Exercício 1: Escala

Modificar a matriz *Model* de forma que a imagem gerada pelo programa fique como a da Figura 5.

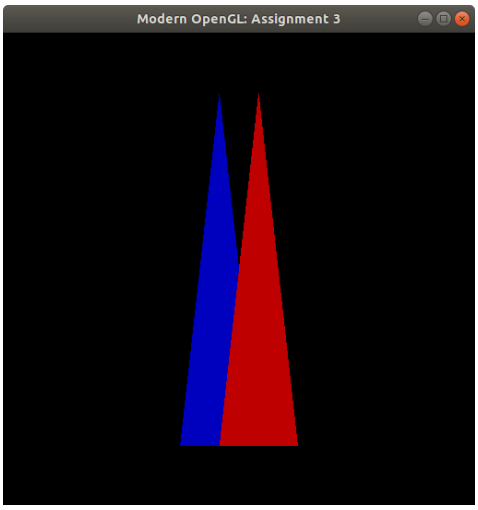

Figure 5: Fatores de escala em $(x, y, z) = (\frac{1}{3}, \frac{3}{2}, 1)$.

2.5.2 Exercício 2: Translação

Modificar a matriz *Model* de forma que a imagem gerada pelo programa fique como a da Figura 6.

2.5.3 Exercício 3: Projeção Perspectiva

Modificar a matriz $M_{Projection}$ de forma que a imagem gerada pelo programa fique como a da Figura 7.

A matriz de projeção $M_{Projection}$ a ser utilizada é a que estudamos em aula e que considera a câmera na origem do seu sistema de coordenadas:

$$M_{Projection} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & -\frac{1}{d} & 0 \end{bmatrix}$$

onde d é a distância do centro de projeção até a origem do sistema de coordenadas da câmera.

2.5.4 Exercício 4: Posição da Câmera

Importante: Este exercício deve ser feito mantendo-se distorção perspectiva implementada na matriz $M_{Projection}$ durante a realização do exercício anterior!

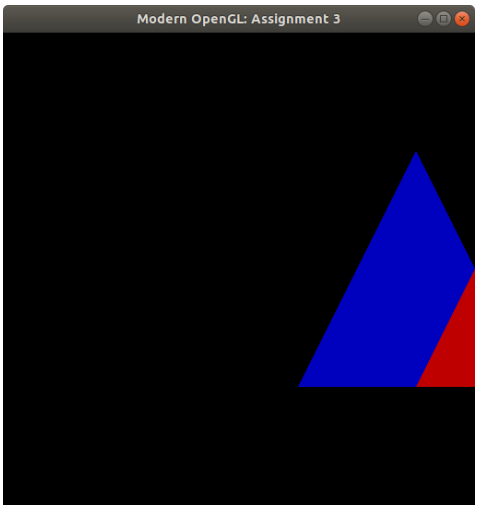

Figure 6: Translações em $(x, y, z) = (1, 0, 0)$.

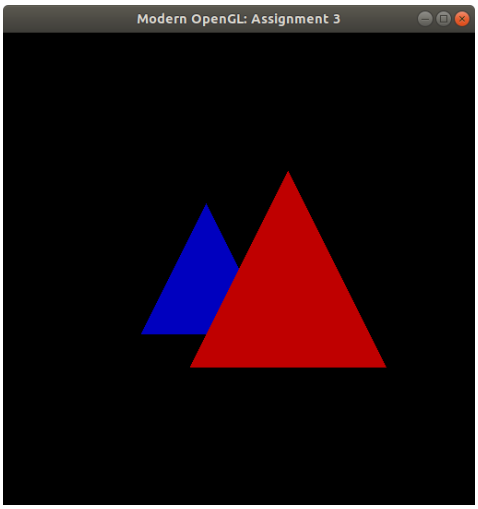

Figure 7: Parâmetro $d = \frac{1}{2}$.

Neste exercício os alunos deverão modificar a matriz M_{View} de forma que a imagem gerada pelo programa fique como a da Figura 8

Para a realização deste exercício, os alunos deverão determinar inicialmente os vetores da base do espaço câmera a partir das informações constantes na legenda da Figura 8. Após, deverão construir a matriz B da base da câmera e invertê-la, calculando sua trasposta B^T. Em seguida, os alunos deverão determinar a matriz T que translada a base da câmera, e faz a sua origem coincidir com a do espaço do universo. Em seguida, os alunos deverão construir a matrix M_{View} como o produto das matrizes B^T e T e então aplicar a transformação da M_{View} sobre os vértices dos triângulos. O resulado final, se estiver tudo correto, deve ser igual ao da Figura 8.

O vetor de direção da câmera pode ser obtido subtraindo-se a posição da câmera do ponto para o qual ela está apontando!

A solução do exercício deve conter todo código necessário para a construção dos vetores da base da câmera e das matrizes B^T, T e M_{View}.

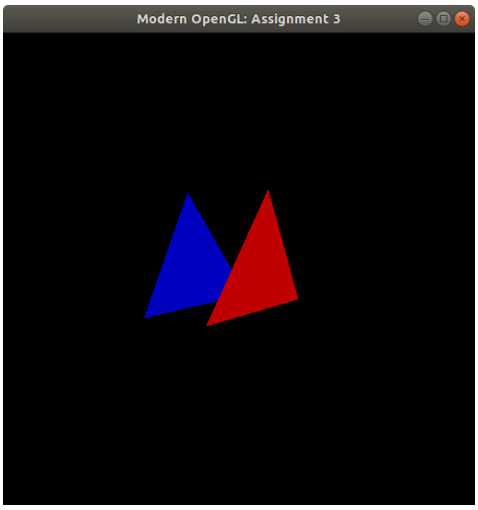

Figure 8: Posição da câmera = $(-\frac{1}{10}, \frac{1}{10}, \frac{1}{4})$, vetor *Up* da câmera = (0,1,0), ponto para o qual a câmera está apontando = (0,0,0).

2.5.5 Exercício 5: Transformações Livres

Neste exercício os alunos deverão fazer modificações nas matrizes M_{Model}, M_{View} e $M_{Projection}$ de forma a gerarem uma cena diferente das geradas anteriormente. Deverão haver transformações nas três matrizes!

Opcionalmente, os alunos poderão renderizar uma geometria diferente da fornecida (*e.g.* mais triângulos, um cubo, um objeto mais complexo carregado a partir de um arquivo externo, *wireframes*, etc.)

2.5.6 Informações Úteis

A `glm` implementa matrizes internamente no formato *column major*, ou seja, uma coluna da matriz após a outra. Desta forma, caso queiramos implementar a matriz abaixo

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

o código correspondente utilizando a `glm` deverá ser

```
float m_array[16] = {1.0f, 5.0f,  9.0f, 13.0f,
                     2.0f, 6.0f, 10.0f, 14.0f,
                     3.0f, 7.0f, 11.0f, 15.0f,
                     4.0f, 8.0f, 12.0f, 16.0f};
glm::mat4 m = glm::make_mat4(m_array);
```

Observe também que, para podermos implementar a matriz 4x4 na `glm`, devemos primeiro criar um *array* de 16 `floats` e então passá-lo como argumento para a função `glm::make_mat4`, que então retornará uma matriz da `glm` do tipo `glm::mat4`.

Maiores detalhes sobre como utilizar matrizes com a `glm` podem ser obtidos na página no site `http://www.opengl-tutorial.org`, mais especificamente no link `http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices`.

3 Entrega

Os trabalhos devem ser entregues até o dia 29/07/2020, impreterivelmente até as 23 horas e 55 minutos. O trabalho será considerado entregue assim que estiver acessível, de forma pública, no repositório Git informado, via SIGAA, pelo aluno. Este trabalho pode ser desenvolvido em duplas.

No repositório deverão constar os *printscreens* dos resultados acompanhados de pequenos textos (frases ou parágrafos) explicativos. O repositório deve conter também o código fonte. Abaixo segue um detalhamento de cada um dos componentes que devem aparecer no *post*.

- ***Printscreens***: Devem demonstrar a evolução do processo de implementação, incluindo eventuais problemas encontrados, e os resultados obtidos após suas correções.
- **Texto**:
  - Deve iniciar com um parágrafo que descreva a atividade desenvolvida.
  - Deve explicar brevemente as estratégias adotadas pelo aluno na resolução da atividade.
  - Breve discussão sobre os resultados gerados, dificuldades e possíveis melhoras.
  - Listar as referência bibliográficas consultadas durante o desenvolvimento do trabalho (livros, artigos, endereços web), se for o caso.
  - OBS: A inclusão de trechos de código no corpo do post deve se limitar ao mínimo indispensável para o correto entendimento do texto!