CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Atividade 3

Ana Maria Pinto da Silva Neta - 20160143190

João Pessoa 2020

Introdução

Este trabalho consiste em implementar as técnicas básicas da Rasterização apresentadas na disciplina de Introdução à Computação Gráfica. Será implementado, a partir dos códigos fornecido pelo docente as modificações de matrizes: Model, Projection e View, para exibir a cena com parâmetros fornecidos pelo docente.

O trabalho foi desenvolvido utilizando uma framework, disponibilizada pelo docente responsável pela disciplina, Christian Azambuja Pagot. A linguagem adotada foi C++.

Desenvolvimento

Primeiramente, foi necessário a instalação de bibliotecas adicionais, incluídas no pdf da descrição da atividade

glm

Assim como foi disponibilizado o repositório do Github da disciplina cujo o subtópico é o *03_transformations*, assim a instalação foi realizada a partir da função clone para a clonagem do programa, com o comando abaixo:

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

GLEW

Esta biblioteca é capaz de gerar referências para as funções do OpenGL para programas em C++, assim foi realizada a instalação a partir do comando abaixo:

```
$ git clone https:github.com/nigels-com/glew.git glew
```

Compilação

Por fim, para que possamos executar o programa devemos utilizar os comandos abaixo

```
$ make
$./transform_gl
```

ou para computadores mais antigos,

```
$ make
$ MESA_GL_VERSION_OVERRIDE=3.3
MESA_GLSL_VERSION_OVERRIDE=330 ./transform_gl
```

Assim, em ambos os caso, mas em diferentes computadores, a imagem que foi mostrada foi essa abaixo:

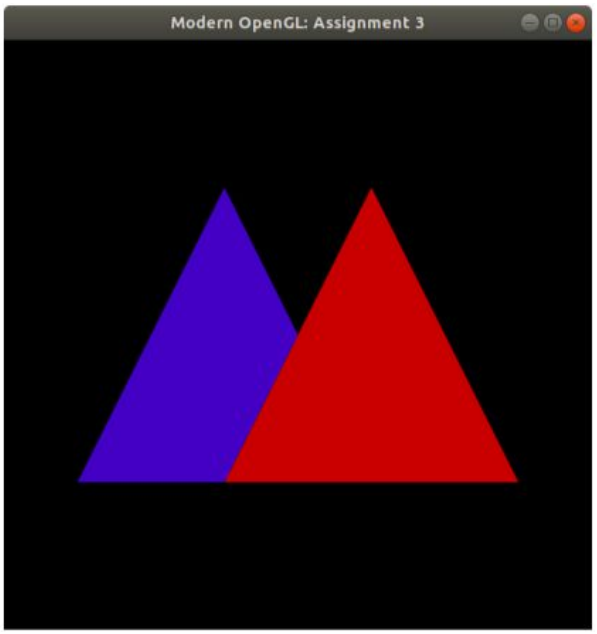

Imagem 1 - Cena do programa padrão.

Alterações

Como nesta atividade foi solicitado que modificarmos as matrizes para que a visualização da cena fosse alteradas conforme o que o professor solicitou, assim com os parâmetros dados foi possível a implementação das transformações geométricas.

Exercício 1 - Escala

O objetivo era modificar a matriz *Model* com os seguintes parâmetros de fatores de escala (x, y, z) = (1/3 , 3/2 , 1).

O que obtive através da matriz model,

```
float model_array[16] = {0.33f, 0.0f, 0.0f, 0.0f,
                         0.0f, 1.5f, 0.0f, 0.0f,
                         0.0f, 0.0f, 1.0f, 0.0f,
                         0.0f, 0.0f, 0.0f, 1.0f};
```

Imagem 2 - matriz *model_array.*

foi o seguinte resultado:

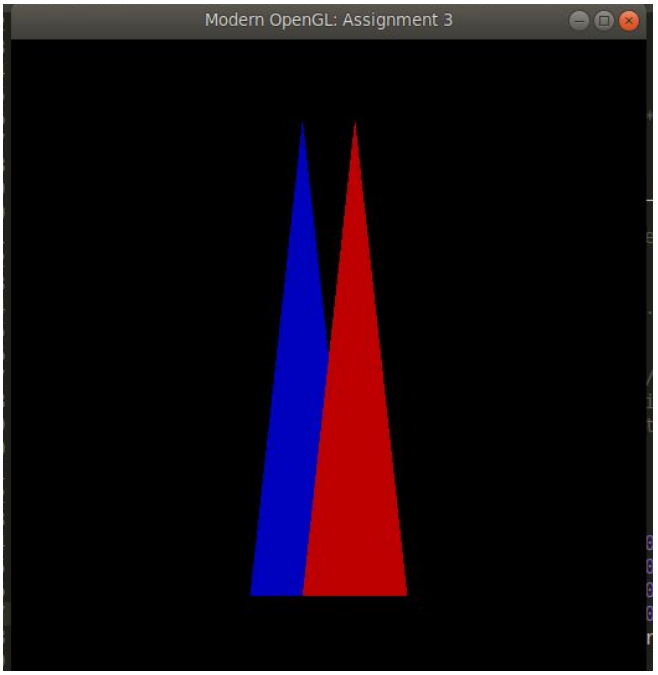

Imagem 3 - Escala.

Exercício 2 - Translação

O objetivo foi modificar a matriz *mode*l, para transladar a imagem com os seguintes parâmetros em (x, y, z) = (1, 0, 0), e foi obtido os seguintes resultados:

```
float model_array[16] = {1.0f, 0.0f, 0.0f, 0.0f,
                         0.0f, 1.0f, 0.0f, 0.0f,
                         0.0f, 0.0f, 1.0f, 0.0f,
                         1.0f, 0.0f, 0.0f, 1.0f};
```

Imagem 4 - Matriz model Translação.

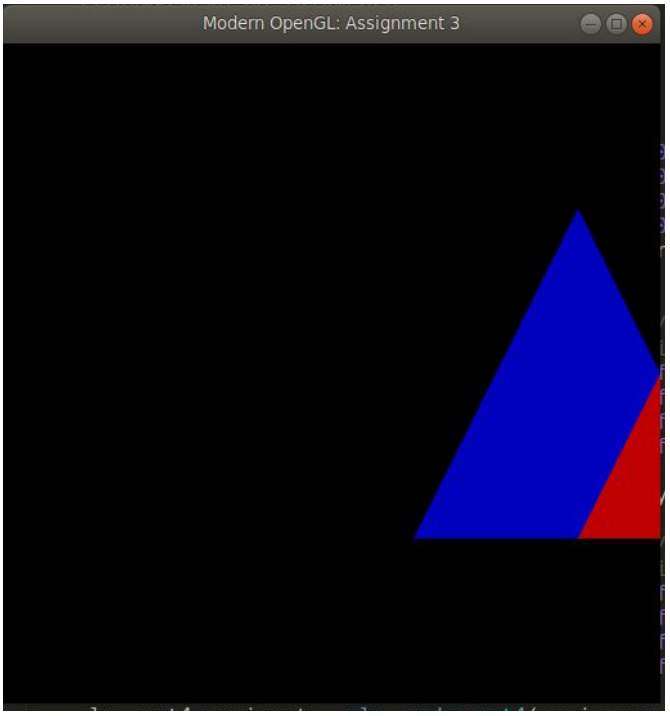

Imagem 5 - Translação.

Exercício 3 - Projeção

O objetivo deste é exercício é ao modificarmos apenas a matriz de projeção utilizando o seguinte parâmetro d = 1/8. Devemos salientar que precisamos retornar os parâmetros de translação ao seu valor padrão, e alteramos apenas o parâmetro da matriz projection para o valor definido, e com isso obteremos os seguintes resultados abaixo:

```
float proj_array[16] = {1.0f, 0.0f, 0.0f, 0.0f,
                        0.0f, 1.0f, 0.0f, 0.0f,
                        0.0f, 0.0f, 1.0f, -8.0f,
                        0.0f, 0.0f, 0.125f, 0.0f};
```

Imagem 6 - Matriz de Projeção.

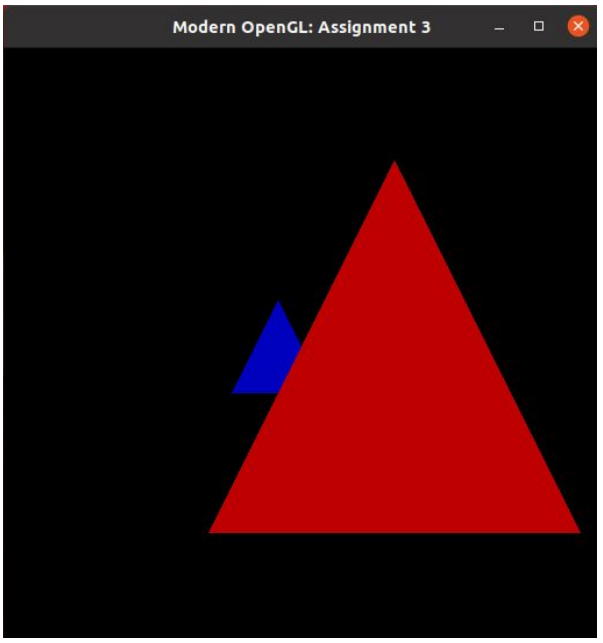

Imagem 7 - Projeção.

Exercício 4 - Posição da Câmera

O objetivo deste exercício é,primeiramente devemos manter alteração feita na matriz projection do exercício 3,e assim alterar a matriz view, com os seguintes parâmetros: P = (-1/10, 1/10,1/10), u = (0)î + (1)ĵ + (0)k̂ e D = (0, 0, -1),mas antes se faz necessário efetuar algumas contas para modificar a matriz:

A matriz responsável por determinar a posição da câmera é calculada a partir da multiplicação de duas outras matrizes, BT e T. Para gerarmos a matriz será necessário de três parâmetros: a posição da câmera(vetor P), o ponto para o qual ela aponta(vetor D),e seu vetor up(vetor u). E com isso podemos calcular o vetor direção da câmera, utilizando as fórmulas a seguir:

$$z_{cam} = -\frac{d_{cam}}{\|d_{cam}\|},$$

$$x_{cam} = \frac{u \times z_{cam}}{|u \times z_{cam}|},$$

$$y_{cam} = z_{cam} \times x_{cam}$$

Assim a matriz Bt é dada por:

$$B^T = \begin{bmatrix} x_{cam(i)} & x_{cam(j)} & x_{cam(k)} & 0 \\ y_{cam(i)} & y_{cam(j)} & y_{cam(k)} & 0 \\ z_{cam(i)} & z_{cam(j)} & z_{cam(k)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

E a matriz T por:

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_i \\ 0 & 1 & 0 & -p_j \\ 0 & 0 & 1 & -p_k \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Logo, assim foi possível obter os seguintes resultados:

```
float view_array[16] = {1.0f, 0.0f, -0.09f, 0.0f,
                        0.0f, 1.0f, -0.09f, 0.0f,
                        0.09f, -0.09f, 1.0f, 0.f,
                        0.09f, -0.09f, -0.12f, 1.0f};
```

Imagem 8 - Matriz view_array.

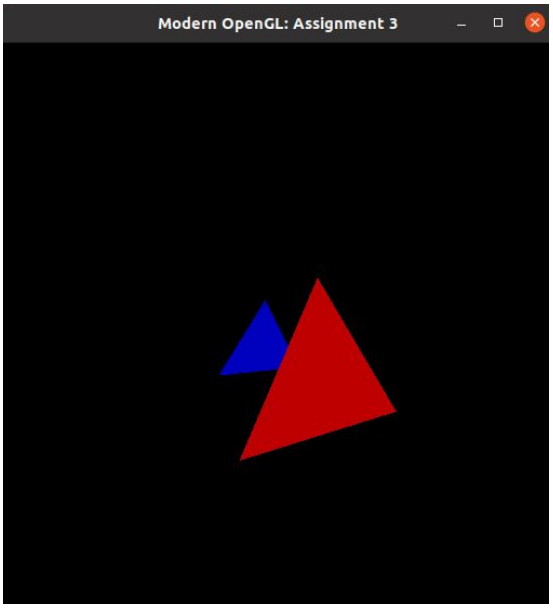

Imagem 9 - Posição da câmera.

Exercício 5 - Renderização livre

E por fim, a renderização livre, primeiramente obtive a partir de pesquisas os seguintes vértices para me basear nessa atividade, que tem o seguinte formato:

```
float vertices[] = {-0.5f,  0.5f, -0.4f, 1.0f, 0.0f, 0.0f,
                     0.5f,  0.5f, -0.4f, 0.0f, 1.0f, 0.0f,
                     0.5f, -0.5f, -0.4f, 0.0f, 0.0f, 1.0f,
                    -0.5f, -0.5f, -0.4f, 1.0f, 1.0f, 1.0f,
                    -0.5f,  0.5f, -0.4f, 1.0f, 0.0f, 0.0f,
                     0.5f, -0.5f, -0.4f, 0.0f, 0.0f, 1.0f};
```

Imagem 10- vértices.

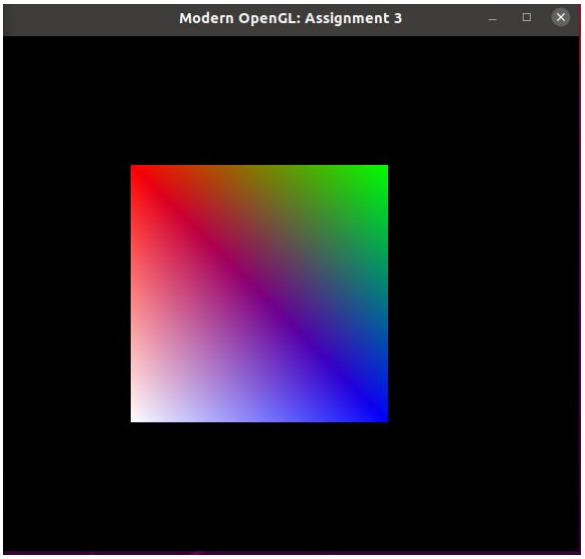

Imagem 10- Resultado dos vértices.

E a partir dos vértices iniciais e algumas alterações abaixo, obteve o seguinte resultado:

```
//RENDERIZAÇÃO LIVRE
float model_array[16] = {1.0f, 0.0f, 0.0f, 0.0f,
                         0.0f, 1.0f, 0.0f, 0.0f,
                         0.0f, 0.0f, 1.0f, 0.0f,
                         0.0f, 0.0f, 0.0f, 1.0f};
glm::mat4 model_mat = glm::make_mat4(model_array);
//POSIÇÃO DA CAMERA2
//para a visualização da posição da Câmera deve-se deixar a
//o model_Array original
/*float view_array[16] = {1.0f, 0.0f, 0.0f, 3.0f,
                        0.0f, 1.0f, 0.0f, 0.0f,
                        0.0f, 0.0f, 1.0f, -2.0f,
                        0.0f, 0.0f, 0.0f, 1.0f};*/
float view_array[16] = {1.0f,0.5f,-0.5f,0.5f,
                        0.0f,1.0f,0.5f,0.0f,
                        0.5f,-0.5f,1.0f,0.0f,
                        -0.1f,-0.1f,-0.5f,1.0f};
glm::mat4 view_mat = glm::make_mat4(view_array);

float proj_array[16] = {1.0f, 0.25f, 0.0f, 0.0f,
                        -4.0f, 1.0f, 0.0f, 0.0f,
                        0.0f, 0.0f, 1.0f, -4.0f,
                        0.0f, 0.0f, 0.25f, 1.0f};
 glm::mat4 proj_mat = glm::make_mat4(proj_array);
```

Imagem 11- códigos

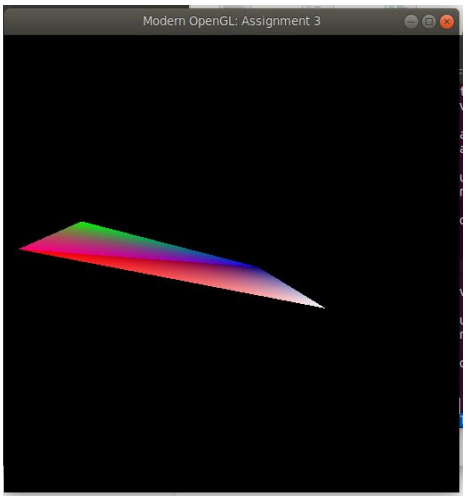

Imagem 12- Resultado quadrado deitado.

Também obtive alguns outros resultados, até chegar nesse, abaixo mostrarei apenas os resultados.

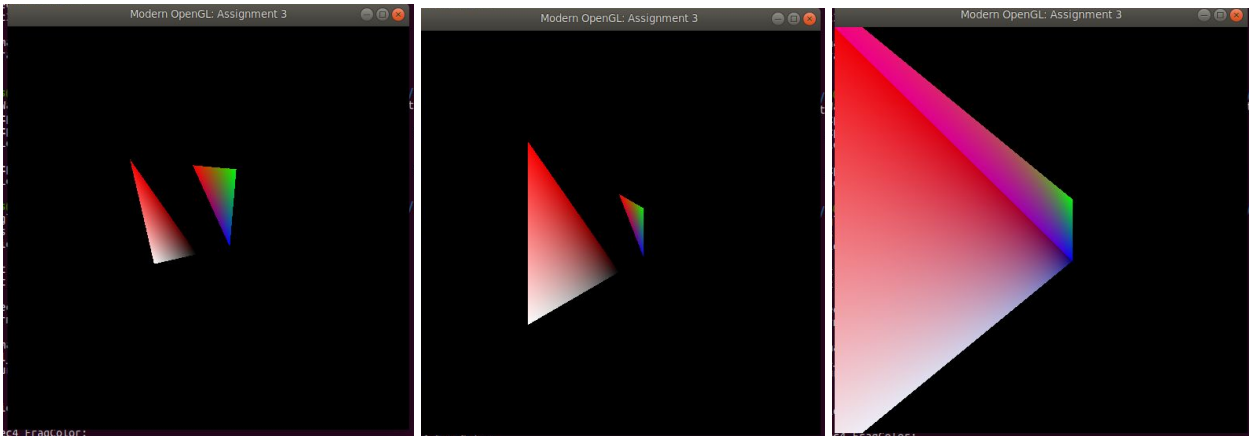

Imagem 13- Resultados livres.

Referências

- https://github.com/capagot/icg/tree/master/03_transformations
- https://glm.g-truc.net/0.9.9/index.html
- http://glew.sourceforge.net/
- https://glad.dav1d.de/
- http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices

.