Introdução a Computação Gráfica – Implementação do Pipeline Gráfico

JOÃO PAULO SILVA MARTINS - 20170107286

VANESSA GABRIELE LIMA PESSOA - 20170159710

OBJETIVO

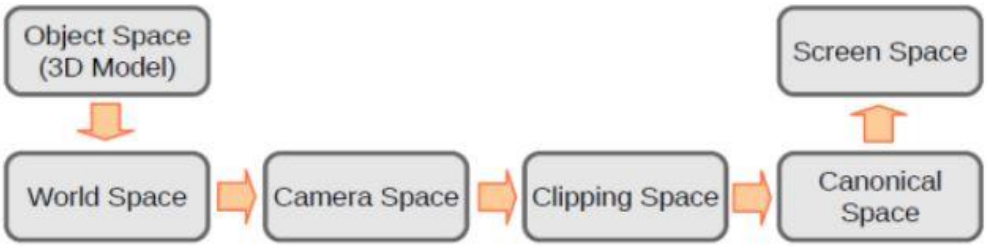
O trabalho consiste na implementação de um pipeline gráfico com as sequências de passos necessários para transformações de um objeto tridimensional, desde o espaço objeto ao espaço de tela

ATIVIDADE

O pipeline gráfico são os passos de transformações que devem ser seguidos para criar a representação de uma cena 3D em um cenário 2D. Cada passo do pipeline transforma descrições geométricas de um sistema de coordenadas para o outro.

Assim como no trabalho anterior, a implementação será feita utilizando a linguagem C/C++. Com o suporte de bibliotecas GLUT, GLM e o loader de arquivos OBJ disponibilizado pelo professor.

DESENVOLVIMENTO

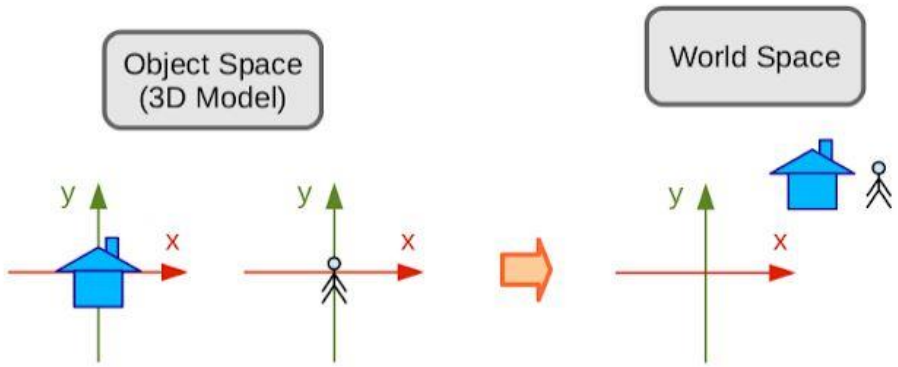


Existem seis etapas no pipeline, os quais cada um leva os vértices de um objeto de um espaço para o outro. As transformações ao longo do pipeline serão implementadas utilizando matrizes e coordenadas homogêneas.

1. **Transformação: Espaço do Objeto --> Espaço do Universo**

Modelos 3D são definidos no seu próprio sistema de coordenada. Esta etapa é responsável por transformar vértices do espaço objeto para o sistema de coordenada do espaço universo.

Pode-se aplicar a um objeto transformações de Rotação, Translação, Escala e Shear. Para realizar esse processo, deve-se realizar a multiplicação dos vértices do objeto pela matriz de modelagem, cada modelo pode ter sua própria matriz.

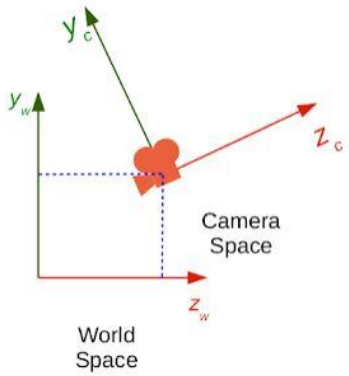


Foram feitas as transformações de translação e de escala:

```
mat4 gerarModelMatrix(float x, float y, float z){

    matrizModelagem = mat4(1.0f);
    mat4 translacao = translate(matrizModelagem, vec3(x,y,z));

    matrizModelagem = scale(translacao, vec3( 1.5f, 1.5f, 1.0f ));

    matrizTransformacaoFinal = matrizModelagem;
}
```

2. **Transformação: Espaço do Universo --> Espaço da Câmera**

Está é a etapa do pipeline responsável por transformar os vértices do espaço do universo para o espaço da câmera. Para isso é necessário definir a posição da câmera e para onde estará apontando, no sistema de coordenados do objeto.



Camera position: $\mathbf{p}=(p_x, p_y, p_z)$

View direction: $\mathbf{d}=(d_x, d_y, d_z)$

Up vector: $\mathbf{u}=(u_x, u_y, u_z)$

$$\mathbf{z}_c = -\frac{\mathbf{d}}{|\mathbf{d}|} = (z_{cx}, z_{cy}, z_{cz})$$

$$\mathbf{x}_c = \frac{\mathbf{u}_c \times \mathbf{z}_c}{|\mathbf{u}_c \times \mathbf{z}_c|} = (x_{cx}, x_{cy}, x_{cz})$$

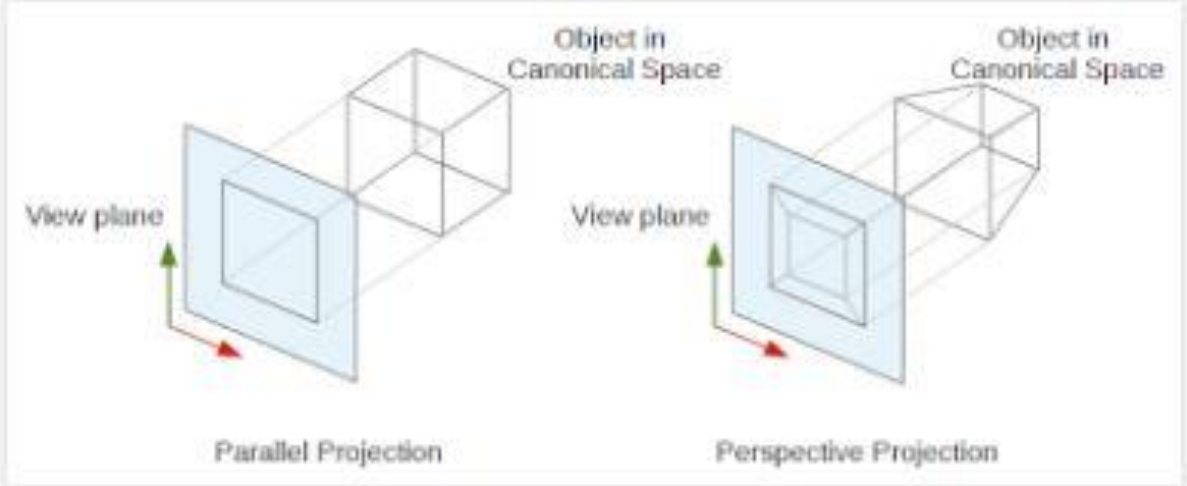
$$\mathbf{y}_c = \frac{\mathbf{z}_c \times \mathbf{x}_c}{|\mathbf{z}_c \times \mathbf{x}_c|} = (y_{cx}, y_{cy}, y_{cz})$$

A transformação do espaço do universo para o espaço da câmera é dado através da multiplicação dos vértices por uma matriz de visualização, que é composta por uma translação e uma rotação. A translação e rotação são definidos pelos vetores de posição, direção e “up”.

```
mat4 gerarMatrizViewPort(float x, float y, float z, float d){

    gerarModelMatrix(x, y, z);
    getViewMatrix();
    gerarMatrizProjecao(d);

    mat4 S1 = mat4(1.0f);
    S1[1].y = 1;

    mat4 S2 = mat4(vec4((IMAGE_WIDTH)/2,     0.0f,       0.0f, 0.0f),
                   vec4(    0.0f,      (IMAGE_HEIGHT)/2, 0.0f, 0.0f),
                   vec4(    0.0f,          0.0f,       1.0f, 0.0f),
                   vec4(    0.0f,          0.0f,       0.0f, 1.0f));

    mat4 T = mat4(1.0f);
    T[3] = vec4((IMAGE_WIDTH-1)/2, (IMAGE_HEIGHT-1)/2, 0.0f, 1.0f);


    matrizViewPort = T * S2 * S1;
    matrizTransformacaoFinal = matrizViewPort * matrizTransformacaoFinal;
}
```

3. Transformação: Espaço da Câmera --> Espaço de Recorte

Nesta etapa do pipeline, os vértices do espaço da câmera são transformados para o espaço de recorte. Nesse processo os vértices que formam a cena são projetados. A transformação é feita através da multiplicação dos vértices descritos no espaço da câmera pela matriz de projeção. São possíveis dois tipos de projeção: ortogonal e perspectiva.

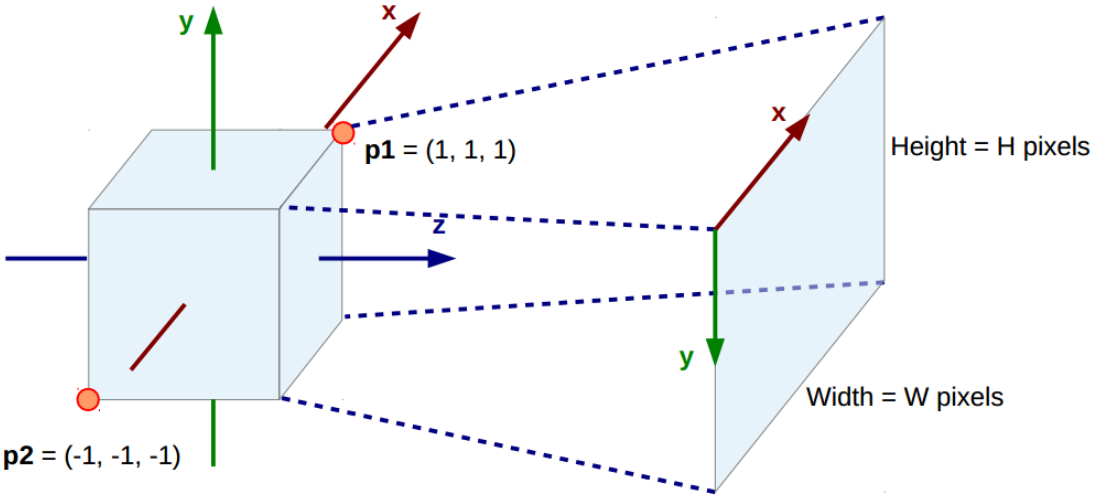


A matriz de projeção leva em consideração a distância do view plane.

```
mat4 gerarMatrizProjecao(float d){
    matrizProjecao = mat4(vec4(1.0f, 0.0f, 0.0f,     0.f),
                          vec4(0.0f, 1.0f, 0.0f,     0.f),
                          vec4(0.0f, 0.0f, 1.0f, -1.0f/d),
                          vec4(0.0f, 0.0f,  d,       1.f));

    matrizTransformacaoFinal = matrizProjecao * matrizTransformacaoFinal;
}
```

4. Transformação: Espaço de Recorte --> Espaço Canônico

Esta etapa é responsável por transformar o espaço de recorte no espaço canônico. Aqui é feito a homogeneização dos vértices, que é dividir todos pela coordenada homogênea, isto causa uma mudança na geometria, que é o que causa a distorção perspectiva na cena.



```
primeiroVertice = primeiroVertice / primeiroVertice.w;
segundoVertice = segundoVertice / segundoVertice.w;
terceiroVertice = terceiroVertice / terceiroVertice.w;
```

5. Transformação: Espaço Canônico --> Espaço de Tela

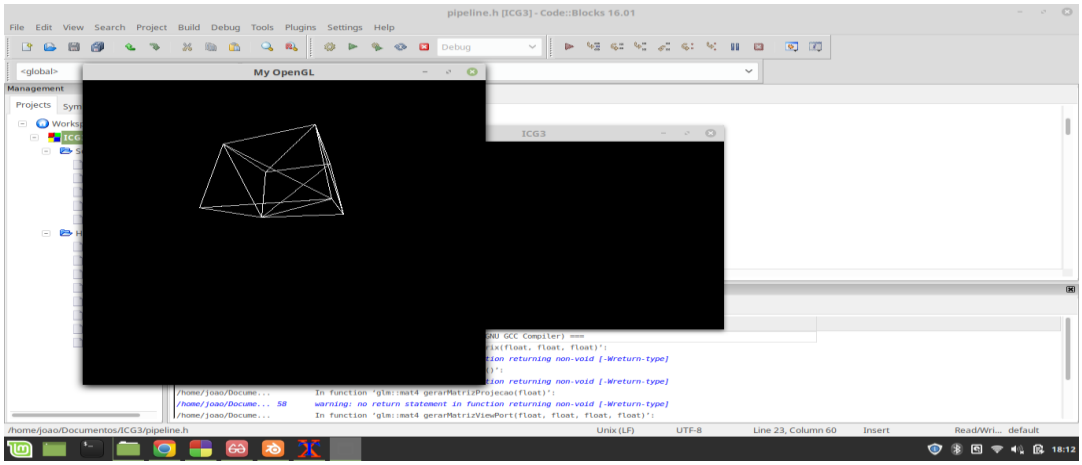
Esta etapa é responsável por transformar os vértices no espaço canônico para o espaço de tela. Isso se dá através da multiplicação dos vértices do espaço canônico pela matriz ViewPort.

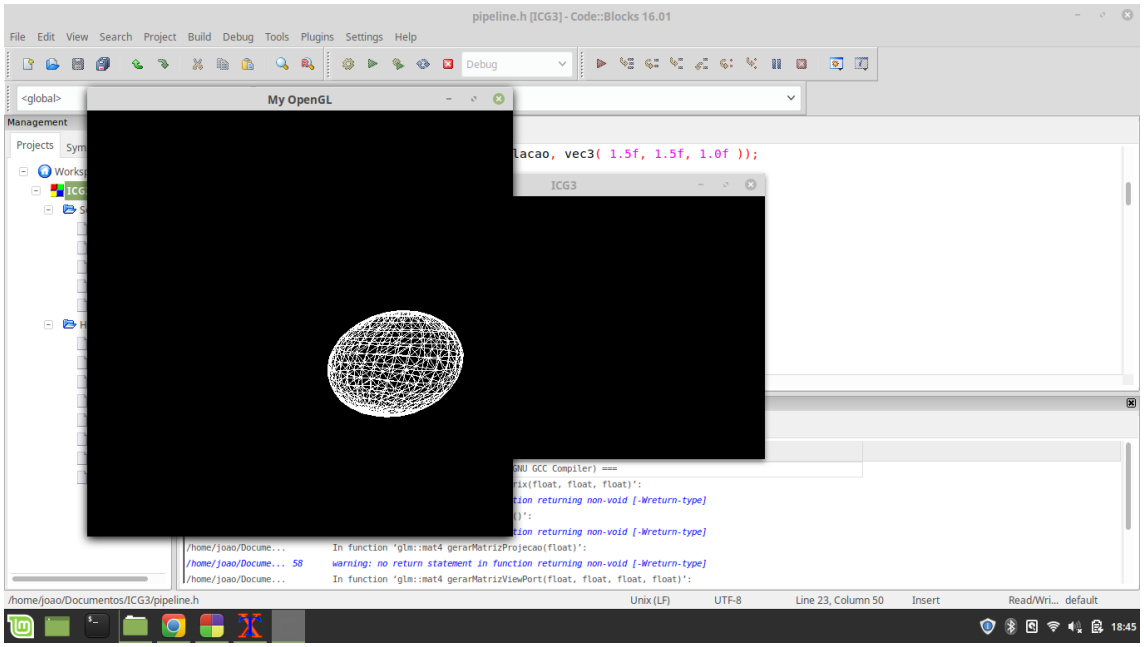
```
mat4 gerarMatrizViewPort(float x, float y, float z, float d){

    gerarModelMatrix(x, y, z);
    getViewMatrix();
    gerarMatrizProjecao(d);

    mat4 S1 = mat4(1.0f);
    S1[1].y = 1;

    mat4 S2 = mat4(vec4((IMAGE_WIDTH)/2,     0.0f,       0.0f, 0.0f),
                   vec4(    0.0f,      (IMAGE_HEIGHT)/2, 0.0f, 0.0f),
                   vec4(    0.0f,          0.0f,       1.0f, 0.0f),
                   vec4(    0.0f,          0.0f,       0.0f, 1.0f));

    mat4 T = mat4(1.0f);
    T[3] = vec4((IMAGE_WIDTH-1)/2, (IMAGE_HEIGHT-1)/2, 0.0f, 1.0f);


    matrizViewPort = T * S2 * S1;
    matrizTransformacaoFinal = matrizViewPort * matrizTransformacaoFinal;
}
```

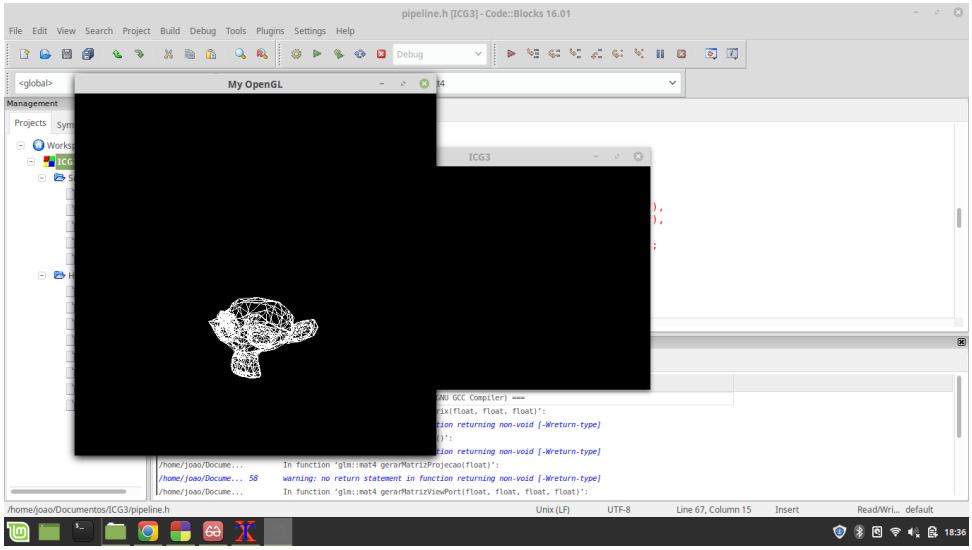
6. Rasterização

Nesta última etapa é gerada a rasterização do modelo transformado pelo pipeline. A implementação foi feita no trabalho anterior.

RESULTADOS

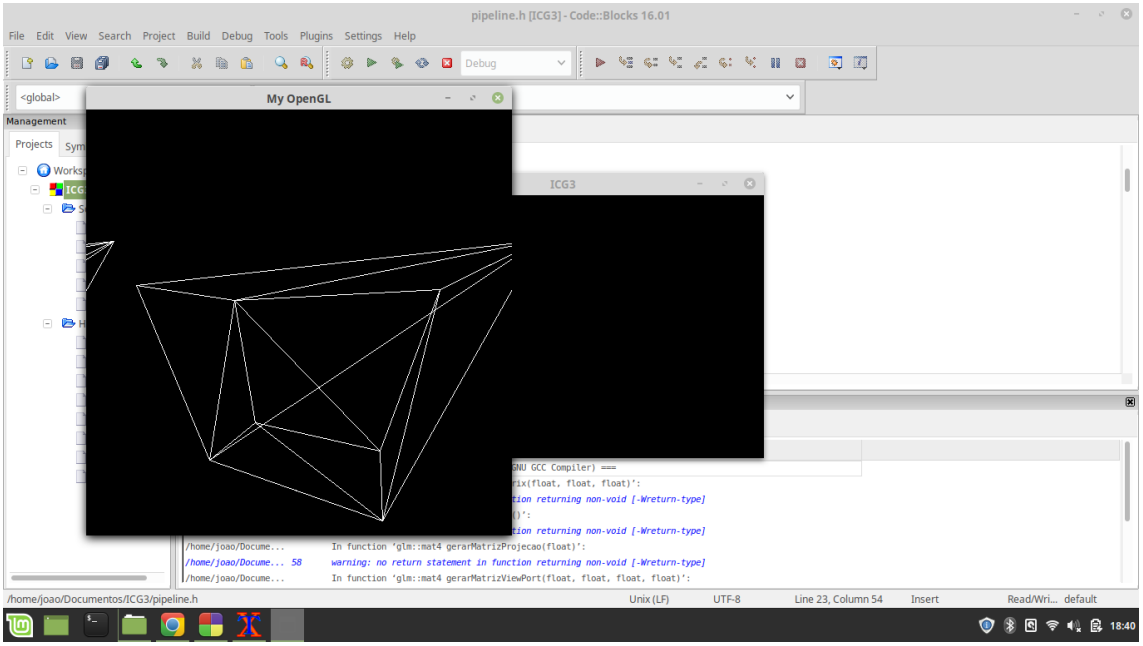

pipeline.h [ICG3] - Code::Blocks 16.01
My OpenGL
ICG3


pipeline.h [ICG3] - Code::Blocks 16.01
My OpenGL
ICG3


pipeline.h [ICG3] - Code::Blocks 16.01
My OpenGL
ICG3

DIFICULDADES

A maior dificuldade foi quando a alteração da escala ficava muito grande e parte do cubo ultrapassava o espaço da tela.



REPOSITORIO

https://github.com/joaopsilvam88/ICG-Projeto-2