Relatório Trabalho 2 - Introdução a Computação Gráfica
2017.2

Clístenes Onassis Chaves Araujo	Mat: **2019077969**
Lucas Correia Barbosa dos Santos	Mat: **2016000560**
Rebeca de Macêdo Ferreira	Mat: **2016000524**

A proposta da segunda atividade é a implementar todas as etapas de transformações de um pipeline gráfico em forma de matriz e utilizar o trabalho 1 para rasterizar as primitivas de um objeto que será carregado por um Loader disponibilizado pelo professor Christian Pagot. As transformações são as seguintes:

1. Espaço do Objeto -> Espaço do Universo
2. Espaço do Universo -> Espaço da Câmera
3. Espaço da Câmera -> Espaço Projetivo ou de Recorte
4. Espaço de Recorte -> Espaço Canônico
5. Espaço Canônico -> Espaço de Tela

As implementações foram feitas em c++ e utilizamos a biblioteca GLM sugerida pelo professor que faz operações de matrizes.

1. Espaço do Objeto p/ Universo

O espaço do objeto é o espaço onde cada objeto é modelado e cada objeto utiliza o seu próprio sistema de coordenadas. Para transformar os vértices dos objetos para o espaço do universo precisamos multiplicar cada vértice por uma matriz de modelagem que pode haver uma ou mais transformações. Vejamos a seguir os tipos de transformações:

Escala ***Shear*** ***Rotação (em x)***

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & m_x & 0 \\ 0 & 1 & m_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translação

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A Seguir a implementação correspondente em C++:

```
glm::mat4 Scale = glm::mat4(2.0);
Scale[3].w = 1.0;

glm::mat4 Translation = glm::mat4(1.0);
glm::vec4 v(1.0, 1.0, 1.0, 1.0);
Translation[3] = v;

glm::mat4 Rotate = glm::mat4(1.0);
Rotate[1].y = cos(ang +=0.5 * PI/180.0);
Rotate[2].y = - sin(ang +=0.5 * PI/180.0);
Rotate[1].z = sin(ang +=0.5 * PI/180.0);
Rotate[2].z = cos(ang +=0.5 * PI/180.0);
Rotate[3].w = 1.0;

glm::mat4 M_Model = Rotate * Scale;
```

Nossa Matriz model portando é composta de uma rotação e uma escala.

2. Espaço do Universo para espaço da Câmera

Para podermos transformar para o espaço da câmera, primeiramente precisamos posicionar a câmera em algum ponto no espaço do universo. Também precisamos de um vetor up que fixa a câmera no espaço e um vetor de direção que sinaliza para onde aponta a câmera.

Implementamos isso da seguinte maneira:

```
glm::vec3 posicao_cam(0.0,0.0,5.0);
glm::vec3 look_at(0.0,0.0,0.0);
glm::vec3 up_cam(0.0,1.0,0.0);
```

Com isso, devemos criar um novo sistema de coordenadas para a câmera, já que agora tudo será passado para este espaço. Para isto foi preciso fazer produto vetorial e normalizar os vetores.

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

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

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

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

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

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

imagem retirada dos slides do professor

Assim, implementamos esta mudança de sistema de coordenadas da seguinte maneira:

```
glm::vec3 zcam = -normalize(look_at - posicao_cam);
glm::vec3 xcam = normalize(cross(up_cam, zcam));
glm::vec3 ycam = normalize(cross(zcam, xcam));
```

O “cross” da biblioteca GLM faz o produto vetorial dos vetores e o normalize faz o comprimento de tais vetores serem iguais a 1, logo, temos um sistema de coordenadas ORTONORMAL.

Agora precisamos apenas de criar a matriz que multiplicará os vértices do espaço do universo para o espaço da câmera, a chamada Matriz view, e ela é composta de uma translação e rotação.

$$\mathbf{B}^{\mathrm{T}}=\begin{bmatrix} x_{cx} & x_{cy} & x_{cz} & 0 \\ y_{cx} & y_{cy} & y_{cz} & 0 \\ z_{cx} & z_{cy} & z_{cz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\mathbf{view}}=\mathbf{B}^{\mathrm{T}}\mathbf{T}$$

imagem retirada dos slides do professor

A implementação desta matriz foi feita da seguinte maneira:

```
B[0]= glm::vec4 (xcam, 0.0);
B[1]= glm::vec4 (ycam, 0.0);
B[2]= glm::vec4 (zcam, 0.0);
glm::vec4 homo(0.0, 0.0, 0.0,1.0);
B[3]=homo;

transposta[3] = glm::vec4 (-posicao_cam, 1.0);

glm::mat4 M_View = transpose(B)*transposta;
```

Poderíamos multiplicar a matriz view pela matriz model, assim teríamos:

M_ModelView = M_Model * M_View;

3. Do espaço da câmera p/ o espaço de Recorte ou Projetivo:

Nesta etapa do pipeline vamos transformar os vértice do espaço da câmera para o espaço de recorte usando a matriz de projeção.

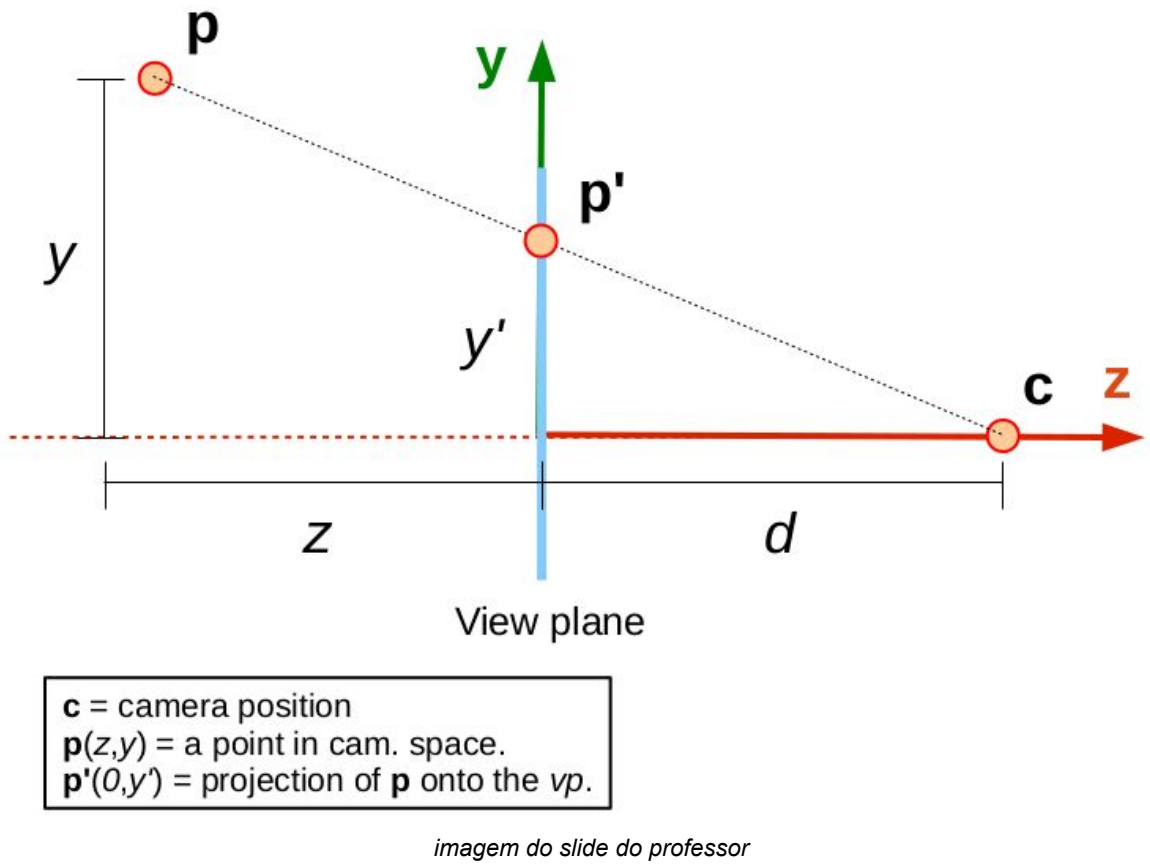


imagem do slide do professor

De acordo com a imagem, temos que ‘d’ é a distância entre a câmera e o view plane, ou near plane; ‘p’ seria um vértice qualquer no view plane. Precisamos de obter uma relação pra y’ e da mesma forma pra x’ e z’, conseguimos isso por semelhança de triangulos.

$$y' = \frac{y}{1 - \frac{z}{d}}$$
$$x' = \frac{x}{1 - \frac{z}{d}}$$
$$z' = \frac{z}{1 - \frac{z}{d}}$$

imagem do slide do professor

A matriz de projeção leva um ponto P (x,y,z,w,1) do espaço da câmera para um ponto P’ (x, y, z, 1-z/d) no espaço de recorte e é escrita da seguinte maneira:

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

Porém se a câmera tiver coincidindo com a origem, faz-se necessário uma matriz de translação:

$$M_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unindo essas duas matrizes, obtemos a matriz de projeção:

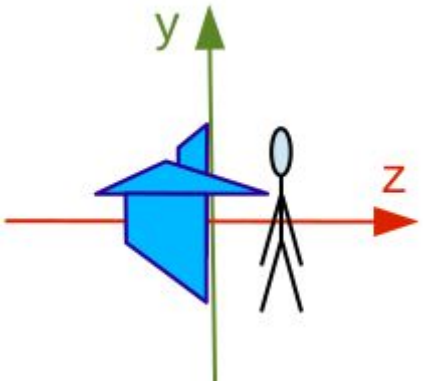
$$M_{pt} = M_p M_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & -\frac{1}{d} & 0 \end{bmatrix}$$

Implementamos esta matriz:

```
double distancia = 1.0;

glm::mat4 M_Projec = glm::mat4(1.0);

M_Projec[2] = glm::vec4(0.0, 0.0, 1.0, -1.0/distancia);
M_Projec[3] = glm::vec4(0.0, 0.0, distancia, 0.0);


glm::mat4 M_ModelViewProjection = M_Projec * M_View * M_Model;
```

4. Espaço de Recorte p/ espaço Canônico.

Nesta etapa do pipeline vamos transformar os vértice do espaço do recorte para o espaço de canônico. Neste ponto vamos dividir as coordenadas dos vértices do espaço de recorte pela sua coordenada de homogênea w, processo chamado de homogeneização. Dessa forma estamos aplicando a distorção não linear dos objetos, passando uma idéia de profundidade e assim fazendo os elementos mais distantes da câmera menores que os elementos mais próximos da câmera.



Implementação da transformação:

```
for (int i = 0; i < objContent->vertexCount; ++i){
    vertex_list[i] = M_ModelViewProjection*vertex_list[i];
    vertex_list[i].x = vertex_list[i].x / vertex_list[i].w;
    vertex_list[i].y = vertex_list[i].y / vertex_list[i].w;
    vertex_list[i].z = vertex_list[i].z / vertex_list[i].w;
    vertex_list[i].w = vertex_list[i].w / vertex_list[i].w;
}
```

5. Espaço Canônico p/ Espaço de Tela.

Nesta etapa vamo transformar o vértices do espaço canônico para o espaço da tela fazendo um mapeamento de cada vértice para a tela. Para tal, devemos multiplicar os vértices por uma matriz com translação e scalas.

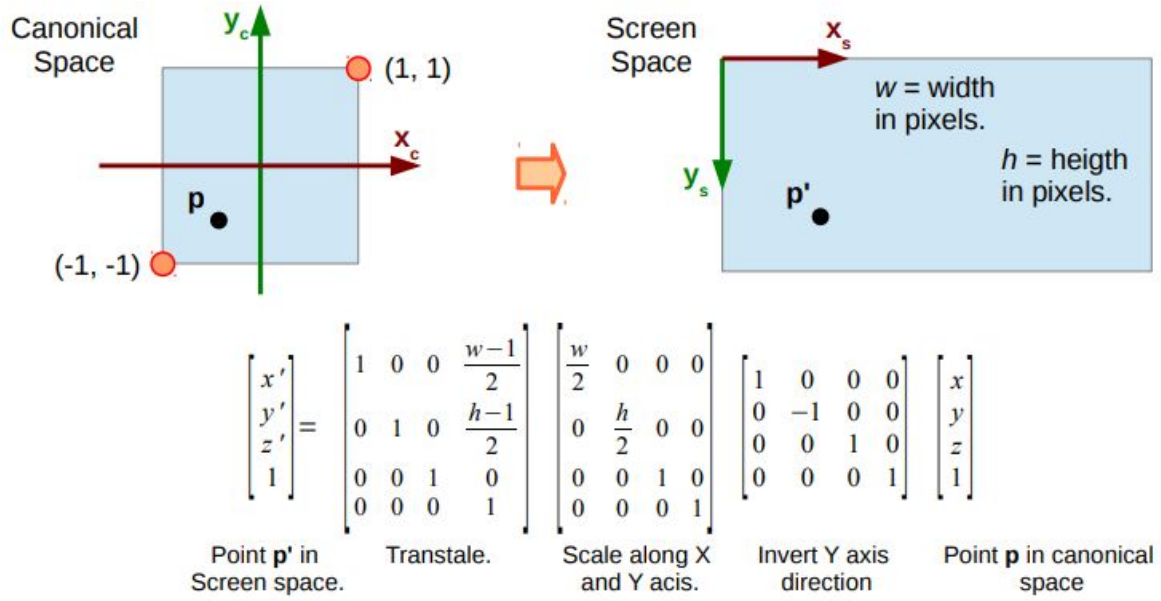


imagem do slide do professor

w = Width. Largura da tela.
h = Heigth. Altura da tela.

Implementação da transformação:

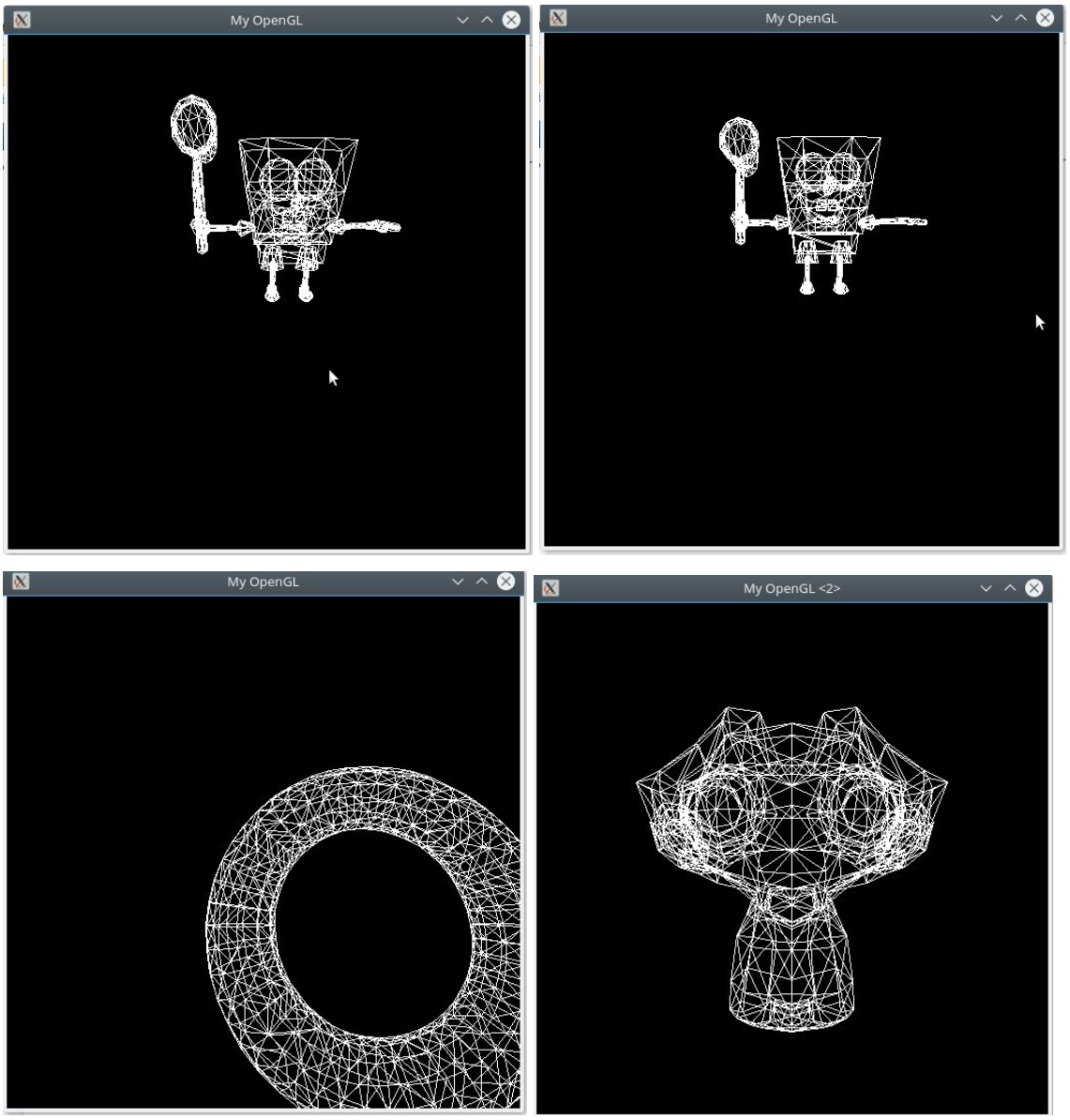
```
glm::mat4 translationTela = glm::mat4(1.0);

translationTela[3] = glm::vec4((IMAGE_WIDTH -1)/2, (IMAGE_HEIGHT -1)/2, 0.0, 1.0);

glm::mat4 scaleTela = glm::mat4(1.0);

scaleTela[0].x = IMAGE_WIDTH/2;
scaleTela[1].y = IMAGE_HEIGHT/2;

glm::mat4 inverteY = glm::mat4(1.0);
inverteY[1].y = -1.0;

glm::mat4 Matriz_final = glm::mat4(1.0);

Matriz_final = translationTela * scaleTela * inverteY;


for (int i = 0; i < objContent->vertexCount; ++i){

    vertex_list[i] = Matriz_final* vertex_list[i];

}
```

Dificuldades:

Nossa primeira dificuldade foi entender como iríamos unir nosso trabalho 1 (rasterização de primitivas) com o loader disponibilizado. Depois disso implementar o pipeline com a biblioteca GLM, aprender suas funções e como usá-las foi também um desafio.

Resultados Obtidos

Resultado em Vídeo: https://www.youtube.com/watch?v=yx6kPlWq85o



Referências

- Slides do professor Christian Pagot
- https://glm.g-truc.net/0.9.8/index.html
- Script do Octave