



**UNIVERSIDADE FEDERAL DA PARAÍBA
DEPARTAMENTO DE INFORMÁTICA
CIÊNCIA DA COMPUTAÇÃO
INTRODUÇÃO À COMPUTAÇÃO GRÁFICA
PROFESSOR: CHRISTIAN AZAMBUJA PAGOT**

Guilherme Vale Sarmiento – 20160163479
Jó Sales de Medeiros Júnior – 20160132104
Samuel Gomes Bezerra Araújo - 20160163773

Relatório – Trabalho II

JOÃO PESSOA

2018

Sumário

1. Introdução	3
1.1 Pipeline Gráfico	3
2. Estratégias adotadas.....	8
3. Resultados e discussões	10
4. Dificuldades encontradas	12
5. Referências Bibliográficas	12

1. Introdução

Em computação gráfica, a sequência de etapas utilizadas para transformar um raster 2D em uma cena 3D é chamada de Pipeline Gráfico, ou seja, o pipeline irá transformar um modelo 3D, dada uma posição e orientação de uma câmera, em uma imagem 2D que representa o modelo 3D, para que assim possa ser exibido em uma tela de computador. Essa imagem 3D pode ser tanto de uma animação, quanto de um jogo, por exemplo.

1.1 Pipeline Gráfico

Cada etapa que o pipeline realiza envolve mudanças de sistemas de coordenadas que pode ser também chamado de mudanças de espaço.

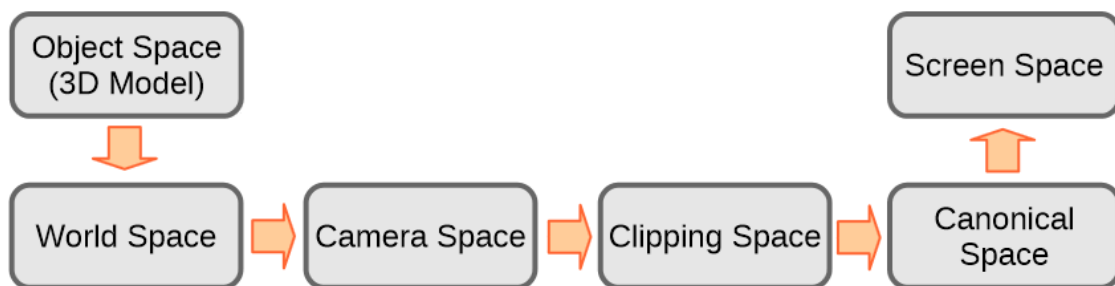


Figura 1 - Pipeline Gráfico

A primeira etapa do Pipeline Gráfico é a transformação do sistema de coordenadas do modelo para o sistema de coordenadas do universo. Essa etapa é necessária, pois um modelo em seu sistema de coordenadas possui o seu centro sempre na origem do seu sistema.

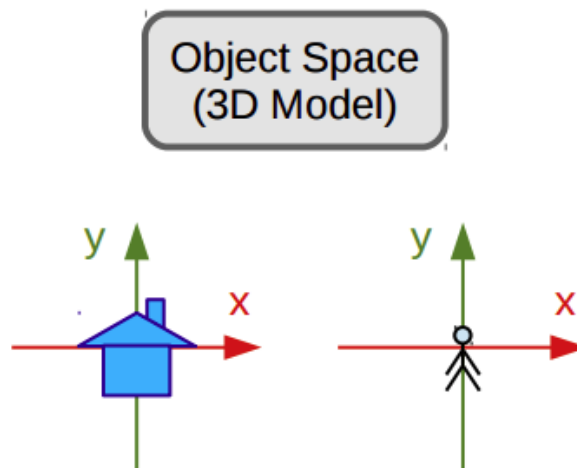
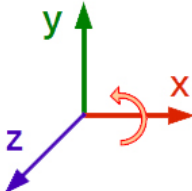


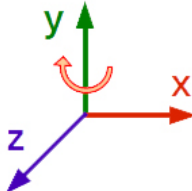
Figura 2 - Modelos e seus sistemas de coordenadas

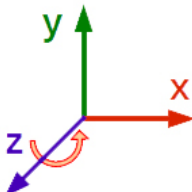
Para transformar do “espaço do objeto” para o “espaço universo” é necessário multiplicar os vértices do objeto por uma matriz chamada de matriz de modelagem (*model matrix*). A matriz de modelagem é composta pelas transformações que desejamos aplicar nos vértices do objeto, as transformações podem ser: Mudança de escala, translação, rotação e *shear*. Cada objeto pode ter a sua própria *model matrix*.

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 3 - Matriz Mudança de Escala e Matriz Translação



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 4 - Matrizes Rotação em cada um dos eixos

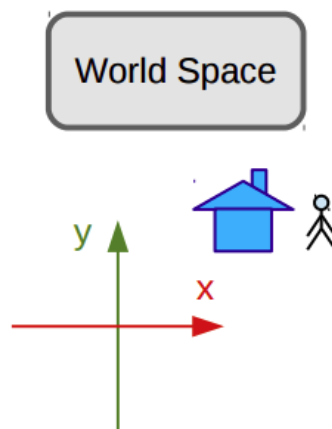


Figura 5 - Transformação Espaço do Objeto em Espaço do Universo

A próxima etapa é realizar a mudança do espaço do universo para o espaço da câmera. Essa etapa, em resumo, consiste em definir o que a câmera está enxergando, pois a câmera está sempre localizada na origem apontando para o eixo $-z$, então na verdade essa etapa realiza a mudança das coordenadas do objeto no espaço universo para o espaço da câmera.

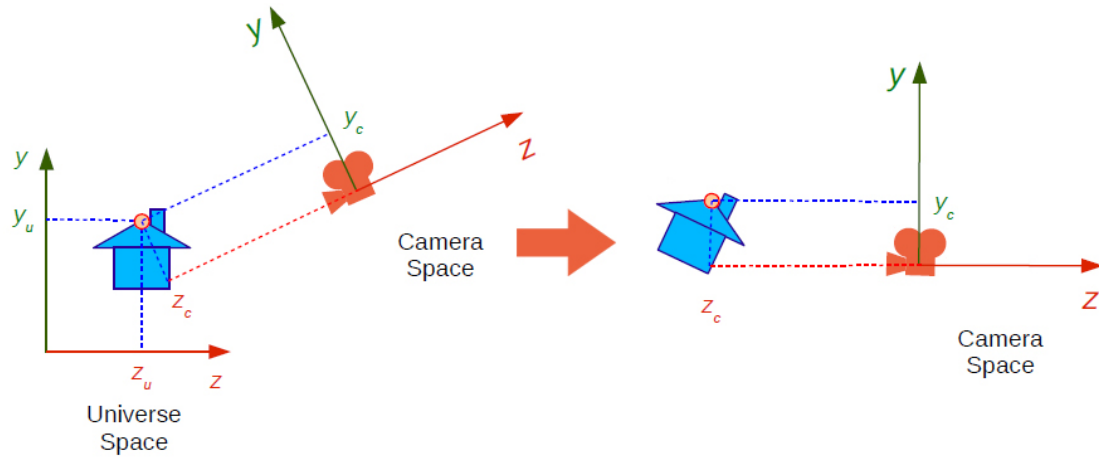


Figura 6 - Mudança para Espaço da Câmera

Essa mudança é realizada ao multiplicar as coordenadas por uma matriz chamada matriz *view*.

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

$$\mathbf{B}^T = \begin{bmatrix} x_{cx} & x_{cy} & x_{cz} & 0 \\ y_{cx} & y_{cy} & y_{cz} & 0 \\ z_{cx} & z_{cy} & z_{cz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 7 - Matriz View

A etapa seguinte é a multiplicação dos vértices no sistema de coordenadas da câmera por uma matriz chamada *projection*. Essa matriz pode ser de projeção perspectiva ou projeção ortogonal. A diferença está que a matriz de perspectiva faz com que os objetos distantes da câmera se tornem menor, dando a impressão de profundidade, e a matriz ortogonal preserva o paralelismo das retas.

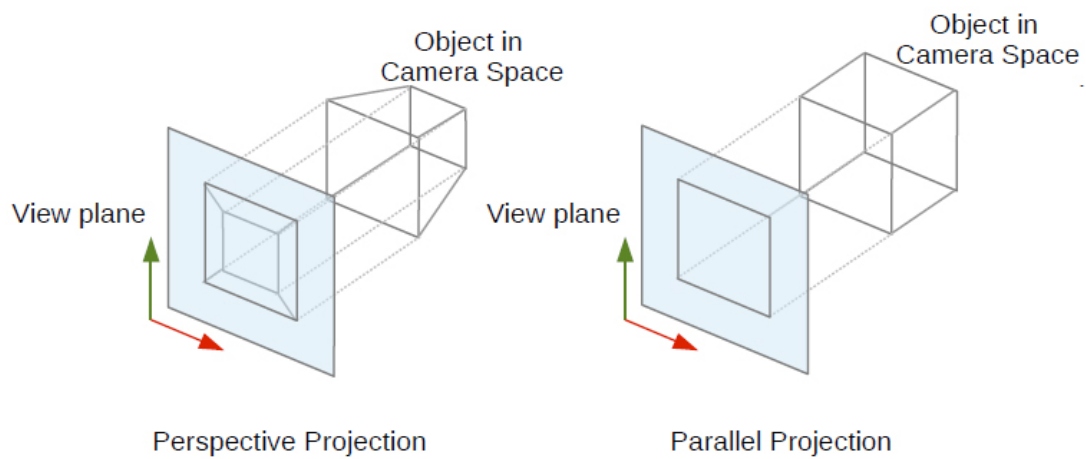


Figura 8 - Tipos de projeção

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

Figura 9 - Matriz de projeção perspectiva

A penúltima etapa, o espaço canônico, é preciso garantir que as coordenadas dos vértices estão entre -1 e 1, e para isso basta dividir as coordenadas do vértice pela sua coordenada homogênea.

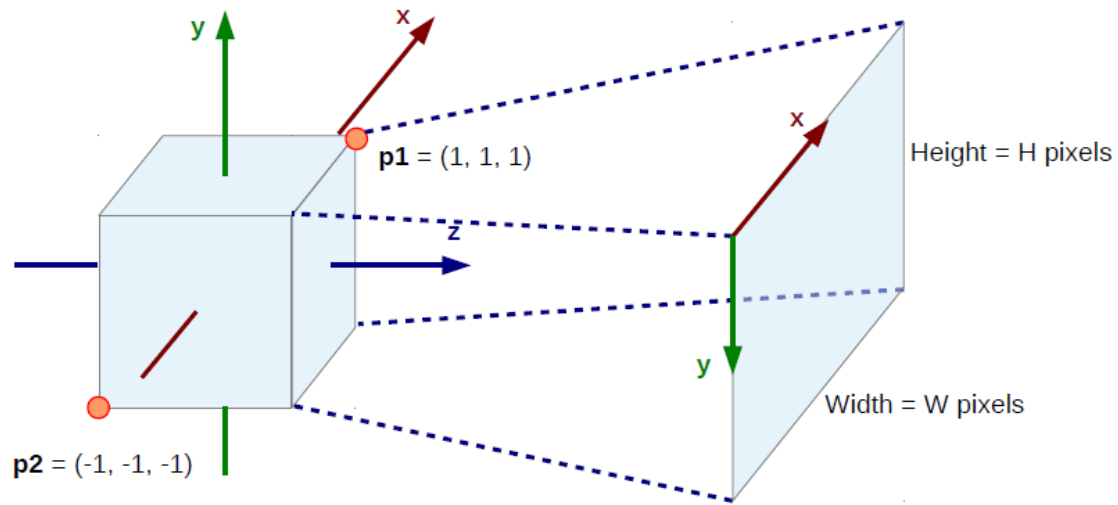


Figura 10 - Espaço canônico

Após isso multiplica os vértices por uma matriz chamada *viewport* que levará os vértices do espaço canônico para o espaço da tela.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{w-1}{2} \\ 0 & 1 & 0 & \frac{h-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transl. Scale along X Invert Y axis
 and Y axis. direction

Figura 11 - Matriz Viewport

E por último, basta aplicar a rasterização das coordenadas dos vértices resultantes. A rasterização foi definida no Trabalho I.

2. Estratégias adotadas

Transformando os conhecimentos em código, a construção da Matriz Model é feita a partir de transformações dos pontos do espaço do objeto para o espaço do universo, então, precisamos considerar que, o espaço do objeto é composto pelos pontos e sistema de coordenadas do objeto, pontos estes que contém valores em cada uma de suas coordenadas x, y e z. Desta forma, para transportar pontos do espaço do objeto para o espaço do universo, nós precisamos utilizar transformações geométricas sobre cada ponto desta matriz.

```
identidade.matriz[0][0]=1; identidade.matriz[0][1]=0; identidade.matriz[0][2]=0; identidade.matriz[0][3]=0;
identidade.matriz[1][0]=0; identidade.matriz[1][1]=1; identidade.matriz[1][2]=0; identidade.matriz[1][3]=0;
identidade.matriz[2][0]=0; identidade.matriz[2][1]=0; identidade.matriz[2][2]=1; identidade.matriz[2][3]=0;
identidade.matriz[3][0]=0; identidade.matriz[3][1]=0; identidade.matriz[3][2]=0; identidade.matriz[3][3]=1;

translacao.matriz[0][0]=1; translacao.matriz[0][1]=0; translacao.matriz[0][2]=0; translacao.matriz[0][3]=0;
translacao.matriz[1][0]=0; translacao.matriz[1][1]=1; translacao.matriz[1][2]=0; translacao.matriz[1][3]=0;
translacao.matriz[2][0]=0; translacao.matriz[2][1]=0; translacao.matriz[2][2]=1; translacao.matriz[2][3]=0;
translacao.matriz[3][0]=0; translacao.matriz[3][1]=0; translacao.matriz[3][2]=0; translacao.matriz[3][3]=1;

escala.matriz[0][0]=2; escala.matriz[0][1]=0; escala.matriz[0][2]=0; escala.matriz[0][3]=0;
escala.matriz[1][0]=0; escala.matriz[1][1]=2; escala.matriz[1][2]=0; escala.matriz[1][3]=0;
escala.matriz[2][0]=0; escala.matriz[2][1]=0; escala.matriz[2][2]=2; escala.matriz[2][3]=0;
escala.matriz[3][0]=0; escala.matriz[3][1]=0; escala.matriz[3][2]=0; escala.matriz[3][3]=2;

M_model = identidade * escala * translacao;
```

O processo de levar os vértices do espaço do universo para o espaço da câmera, é montando a Matriz View. Com os nossos objetos reunidos no espaço do universo e com o seus respectivos sistemas de coordenadas agora tridimensionais, nós precisamos definir o posicionamento da nossa câmera, para através do produto dos nossos objetos pela Matriz View possamos chegar ao espaço de câmera. Dessa forma, as três informações importantes que precisamos definir a respeito da nossa câmera são: Posição da câmera, Direção da câmera e o vetor Up.

```
//Espaco Universo -> Espaco Camera

Vertice posicaoCamera(0,0,5);
Vertice LookAt(0,0,0);
Vertice Up(0,1,0);

Vertice cameraZ=(posicaoCamera-LookAt)/ normalize(posicaoCamera-LookAt);
Vertice cameraX=cross(Up,cameraZ)/normalize(prodVetorial(Up,cameraZ));
Vertice cameraY=cross(cameraZ,cameraX)/normalize(prodVetorial(cameraZ,cameraX));

Matriz M_view, bT, T;

bT.matriz[0][0]=cameraX.V[0]; bT.matriz[0][1]=cameraX.V[1]; bT.matriz[0][2]=cameraX.V[2]; bT.matriz[0][3]=0;
bT.matriz[1][0]=cameraX.V[0]; bT.matriz[1][1]=cameraX.V[1]; bT.matriz[1][2]=cameraX.V[2]; bT.matriz[1][3]=0;
bT.matriz[2][0]=cameraX.V[0]; bT.matriz[2][1]=cameraX.V[1]; bT.matriz[2][2]=cameraX.V[2]; bT.matriz[2][3]=0;
bT.matriz[3][0]=0; bT.matriz[3][1]=0; bT.matriz[3][2]=0; bT.matriz[3][3]=1;

T.matriz[0][0]=1; T.matriz[0][1]=0; T.matriz[0][2]=0; T.matriz[0][3]=-posicaoCamera.V[0];
T.matriz[1][0]=0; T.matriz[1][1]=1; T.matriz[1][2]=0; T.matriz[1][3]=-posicaoCamera.V[1];
T.matriz[2][0]=0; T.matriz[2][1]=0; T.matriz[2][2]=1; T.matriz[2][3]=-posicaoCamera.V[2];
T.matriz[3][0]=0; T.matriz[3][1]=0; T.matriz[3][2]=0; T.matriz[3][3]=1;

//Matriz View
M_view= bT*T
```



```
// Matriz ModelView

Matriz ModelView = M_view * M_model;
```

Neste momento, os vértices deverão ser multiplicados por uma matriz especial chamada Projeção, fazendo com que os vértices do espaço da câmera sejam levados para o espaço de recorte. A multiplicação por essa matriz irá definir o tipo de projeção, podendo ser ortogonal ou perspectiva.

```
// Espaço Câmera -> Espaço de Recorte

// Matriz de Projeção
Matriz projecao;

double d=2.2;

projecao.matriz[0][0]=1; projecao.matriz[0][1]=0; projecao.matriz[0][2]=0;   projecao.matriz[0][3]=0;
projecao.matriz[1][0]=0; projecao.matriz[1][1]=1; projecao.matriz[1][2]=0;   projecao.matriz[1][3]=0;
projecao.matriz[2][0]=0; projecao.matriz[2][1]=0; projecao.matriz[2][2]=1;   projecao.matriz[2][3]=d;
projecao.matriz[3][0]=0; projecao.matriz[3][1]=0; projecao.matriz[3][2]=-1/d; projecao.matriz[3][3]=0;

//Matriz ModelViewProjection
Matriz ModelViewProjection = projecao * ModelView;
```

Os vértices do espaço projetivo são transformados para o espaço canônico, para isto divide-se as coordenadas dos vértices no espaço projetivo pela sua coordenada homogênea. Isto gera uma mudança na geometria da cena, os objetos próximos da câmera ficam maiores, e os mais afastados ficam menores.

```
//Aplicando transformações nos vértices do objeto
for(int i=0; i<objData->faceCount; i++){
    for(int j=0; j<3; j++){
        V3[i][j]= ModelViewProjection * V2[i][j];
    }
}

//Dividindo pela Coordenada Homogênea
for(int i=0; i<objData->faceCount; i++){
    for(int j=0; j<3; j++){
        V3[i][j]= V3[i][j]/V3[i][j].V[3];
    }
}
```

Após o espaço canônico é preciso preparar os vértices para serem rasterizados na tela. Este processo é feito multiplicando os vértices por uma matriz chamada viewport. Essa matriz leva os vértices do espaço canônico para o “espaço da tela”.

```
//Espaco Canônico -> Espaco de TELA

matriz1.matriz[0][3]=((IMAGE_WIDTH-1)/4);
matriz1.matriz[1][3]=((IMAGE_HEIGHT-1)/4);

matriz2.matriz[0][0]=((IMAGE_WIDTH-1)/4);
matriz2.matriz[1][1]=((IMAGE_HEIGHT-1)/4);

matriz3.matriz[1][1]=-1;

matrizFinal = matriz1 * matriz2 * matriz3;

for(int i=0; i<objData->faceCount; i++){
    for(int j=0; j<3; j++){
        V3[i][j]= matrizFinal*V3[i][j];
    }
}
```

3. Resultados e discussões

Depois de rasterizado, o resultado foi o seguinte:

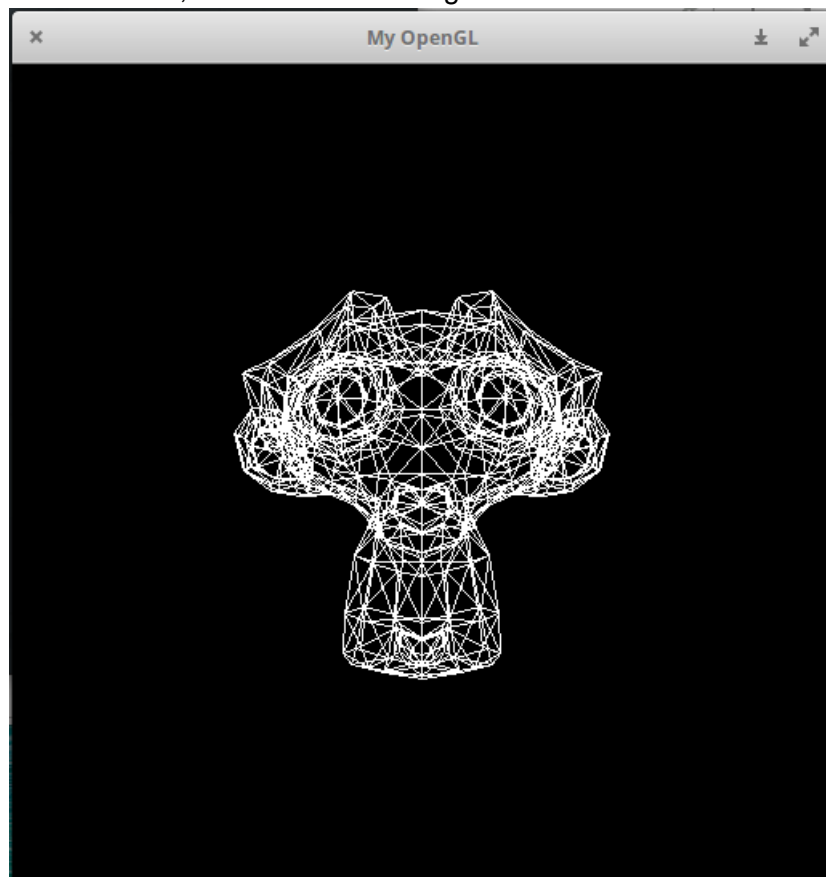


Figura 12 - Imagem do monkey_head.obj renderizado usando triângulos

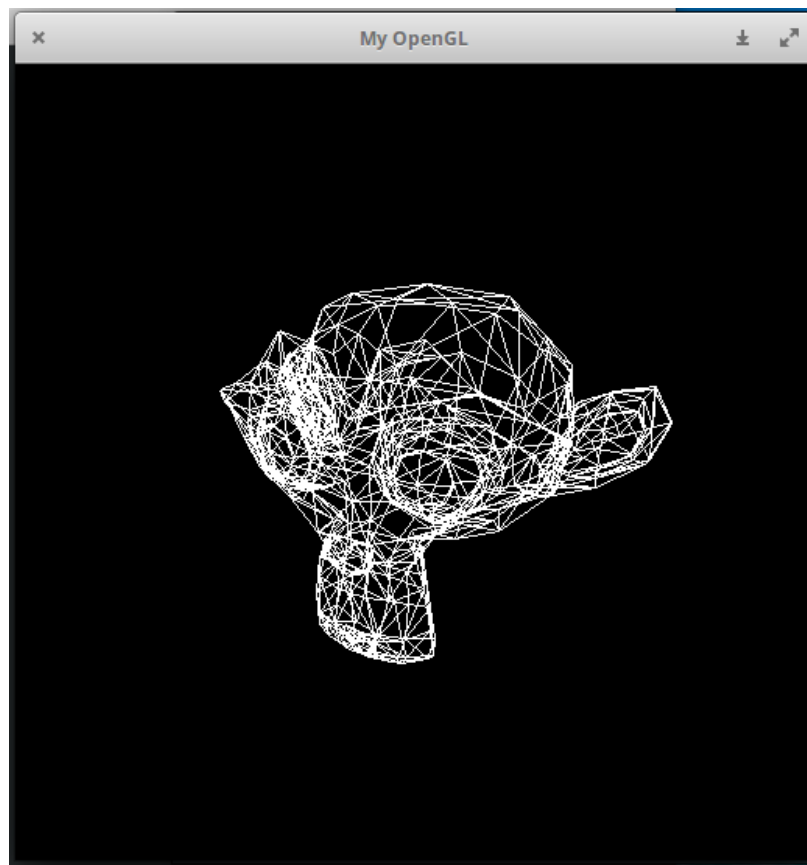


Figura 13 - Imagem do monkey_head.obj com outra vista

Em seguida segue a comparação entre a rasterização descrita neste relatório, e uma em OpenGL

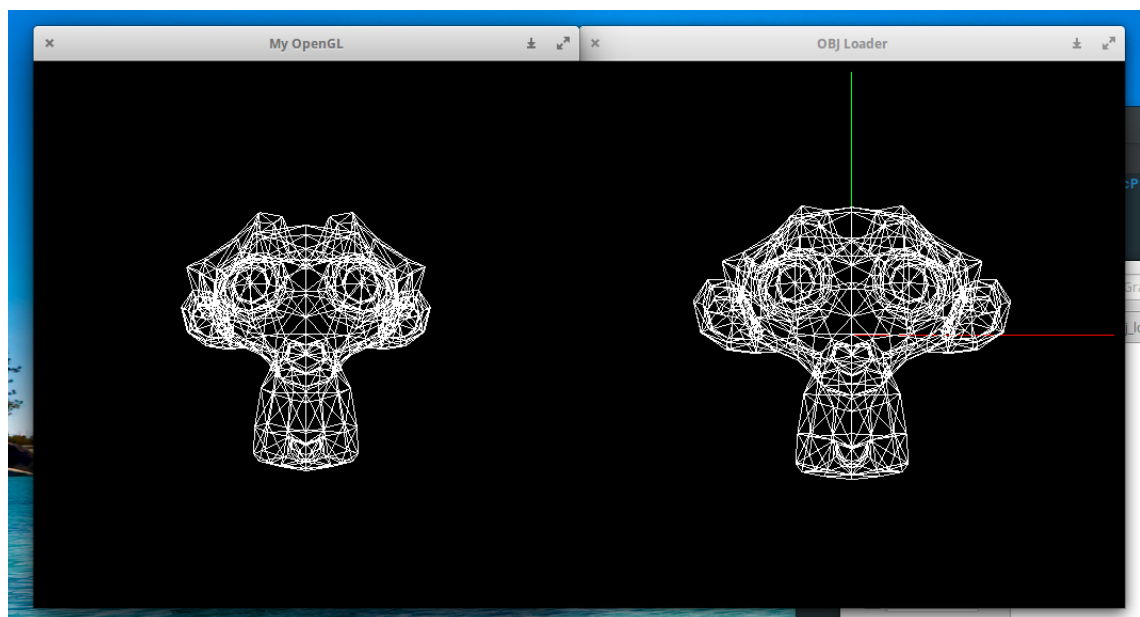


Figura 14 – Imagem esquerda: pipeline gráfico. Imagem direita: OpenGL

Segue o link do vídeo das transformações:

<https://www.youtube.com/watch?v=oRi4fwQscu4>

4. Dificuldades encontradas

A maior dificuldade neste trabalho esteve nos detalhes implementação das transformações de um espaço para outro, além de garantir que cada etapa do pipeline gráfico estivesse funcionando corretamente. Outra dificuldade neste trabalho foi descrever cada passo do pipeline de uma forma inteligível e não cansativa.

5. Referências Bibliográficas

Material do Lehrer

https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Graphics_pipeline.html

Slides sobre pipeline gráfico encontrados no sistema integrado de gestão de atividades acadêmicas da UFPB, disponibilizados pelo professor Azambuja Pagot.