Universidade Federal da Paraíba
Centro de Informática

Arthur Ricardo - 20170033039
Gabriel Marques Barbosa - 11406921

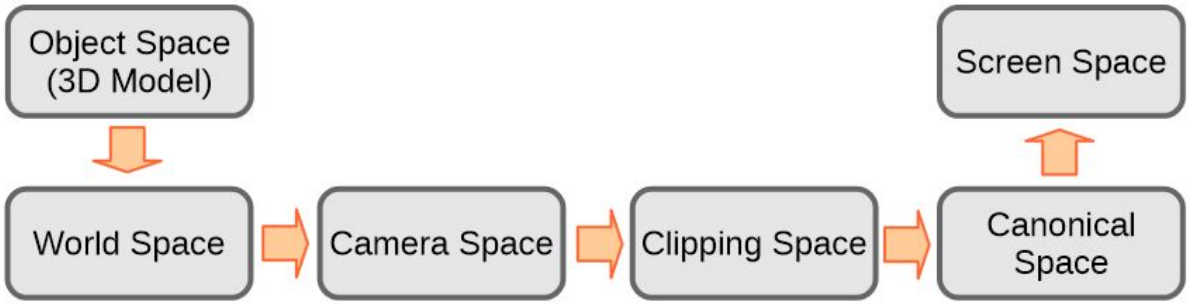
Introdução e implementação de um *Pipeline* Gráfico baseado em rasterização

Introdução a Computação Gráfica - 2018.2
Docente: Christian Azambuja Pagot

João Pessoa
22 de abril de 2019

Introdução

Para que uma representação de um objeto, no contexto de computação gráfica, seja representado na tela como se deseja, uma sequência de passos é realizada para transformar tal objeto para a cena que se quer. Tal sequência é chamada de *Pipeline* Gráfico. Basicamente, cada passo do *pipeline* consiste em transformações geométricas que são aplicadas sobre os vértices que constituem o objeto, transformando-o de um sistema de coordenadas (espaço) para outro, como ilustrado na figura abaixo:



Neste trabalho será discutida uma implementação de um *pipeline* baseado em rasterização. Serão também abordadas as transformações que serão aplicadas sobre os vértices, de modo a levar o objeto do espaço do objeto(object space) até o espaço de tela (screen space). Em seguida, o processo de rasterização das primitivas é realizado, utilizando um trabalho previamente desenvolvido nesta disciplina **[1, 2]** .

Transformações ao longo de pipeline

As transformações ao longo do *pipeline* gráfico podem ser (e normalmente são) implementadas utilizando matrizes e coordenadas homogêneas. O motivo para isso se dá pela possibilidade de escrever um conjunto de transformações como um produto de matrizes.

Espaço do Objeto → Espaço do Universo

O espaço do objeto é onde cada objeto é criado e modelado a partir de suas primitivas geométricas utilizando seu próprio sistema de coordenadas. Nesta passagem, entre o espaço do objeto e o espaço do universo, as coordenadas do objeto são transformadas do lugar onde ele foi modelado para a cena. Isso é feito com o uso de uma matriz chamada de **matriz *model*,** que consiste na combinação de uma série de transformações sobre os vértices do objeto para adequá-lo ao sistema de coordenadas da cena**.** As transformações podem ser:

Escala:

Esta transformação realiza um redimensionamento no objeto, multiplicando as coordenadas de seus vértices por um fator de escala, podendo o resultado desta aumentar, diminuir, deformar ou até mesmo espalhar tal objeto.

$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$
$$z' = z \cdot s_z$$

Rotação:

Para rotacionar precisamos estabelecer em qual eixo será feita a rotação. Para cada eixo será utilizado uma matriz de rotação diferente, sendo também necessário estabelecer de quanto será essa rotação, por meio de um ângulo θ . A seguir é apresentado como ficam as equações que compõem tais matrizes para cada um dos eixos.

Em X:

$$x' = x$$
$$y' = y\cos(\theta) - z\sin(\theta)$$
$$z' = y\sin(\theta) + z\cos(\theta)$$

Em Y:

$$x' = x\cos(\theta) + z\sin(\theta)$$
$$y' = y$$
$$z' = -x\sin(\theta) + z\cos(\theta)$$

Em Z:

$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$
$$z' = z$$

Translação

Nesta transformação os vértices serão deslocados por fatores de translação d_x, d_y e d_z, correspondentes os deslocamentos em x , y e z, respectivamente. Para tal, basta somar esse fatores em cada vértice.

$$x' = x + d_x$$
$$y' = y + d_y$$
$$z' = z + d_z$$

Essa transformação porém, não pode ser representada por meio de uma única matriz, para isso, faz-se necessário o uso do sistema de coordenadas homogéneas.

***Shear* ou Cisalhamento**

Nessa transformação, enquanto uma ou mais coordenadas são fixadas, as outras são deslocadas de acordo com um coeficiente m_i, sendo i o eixo em que será aplicado o *shear.* Como ilustrado abaixo:

$$x' = x + m_x z$$
$$y' = y + m_y z$$
$$z' = z$$

Coordenadas Homogêneas

Coordenadas homogêneas tem como objetivo podermos combinar qualquer dessas transformações em uma matriz. Com tal sistema de coordenadas, ao invés de um vértice com coordenadas (X , Y , Z) agora teremos um vértice com coordenadas (X',Y',Z', W) onde X' = W * X , Y' = W * Y , Z' = W * Z , sendo W chamado de coordenada homogênea. Para facilitar, considere W = 1, sendo então a nova coordenada do vértice da forma (X , Y, Z , 1). Agora, somos capazes de representar a translação como um **Cisalhamento(*Shear*)** no espaço homogêneo . As outras transformações também deverão ser feitas no espaço homogêneo, com isso, suas matrizes ficam:

Escala:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Shear ou Cisalhamento:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & m_x & 0 \\ 0 & 1 & m_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotação:

Em X:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Em Y:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Em Z:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

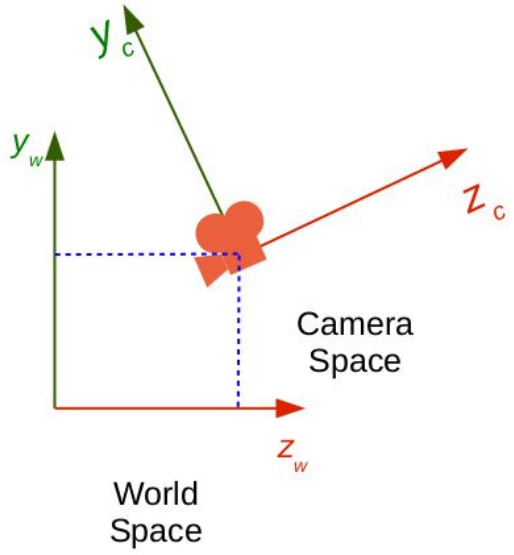
Translação:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Espaço do Universo → Espaço da Câmera

Essa etapa consiste na passagem da cena do espaço do universo (world space) para o espaço da câmera(Camera Space). Essa passagem é feita através da **matriz *view*** a qual apresenta uma mudança de base e uma translação .

Para a mudança de base devemos definir a posição da câmera , para onde a câmera está olhando e o vetor up (em termo popular, seria “o vetor para o qual a parte de cima da câmera aponta”) com isso podemos definir a base do seguinte modo:



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

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

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

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

$$\mathbf{x_c} = \frac{\mathbf{u_c} \times \mathbf{z_c}}{|\mathbf{u_c} \times \mathbf{z_c}|} = (x_{cx}, x_{cy}, x_{cz})$$

$$\mathbf{y_c} = \frac{\mathbf{z_c} \times \mathbf{x_c}}{|\mathbf{z_c} \times \mathbf{x_c}|} = (y_{cx}, y_{cy}, y_{cz})$$

agora é definida a matriz de mudança de base e a translação:

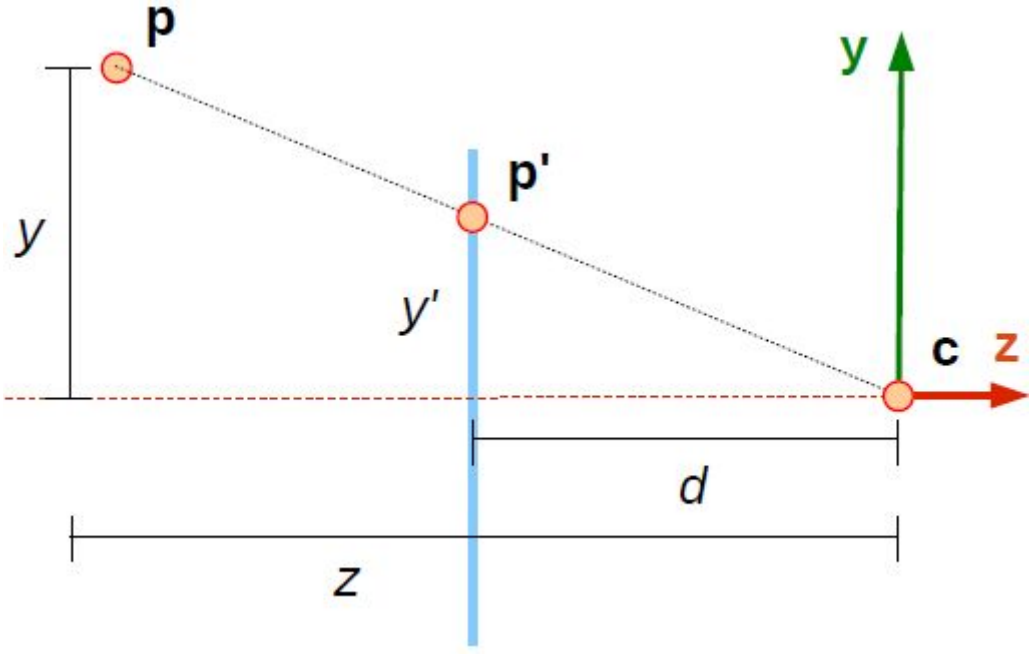
$$\mathbf{B}^{\mathbf{T}}=\begin{bmatrix} x_{cx} & x_{cy} & x_{cz} & 0 \\ y_{cx} & y_{cy} & y_{cz} & 0 \\ z_{cx} & z_{cy} & z_{cz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Então, para definir a Model View basta multiplicar as duas:

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

Espaço da Câmera → Espaço Projetivo ou *Clipping Space* (Espaço de Recorte)

Nesta fase, os vértices são transformados do espaço da câmera para um espaço chamado *Clipping Space* (no português, espaço de recorte). Esse nome se dá devido a que nesta etapa se tem os primeiros descartes de primitivas. Também é chamado de Espaço projetivo, pois as transformações que ocorreram nos vértices de um objeto para esse espaço dão a "noção de projeção do objeto", que era 3D, para uma visualização em 2D e a noção de distância através de distorção perspectiva. Para tal, as coordenadas dos vértices são multiplicadas por uma matriz conhecida como ***projection***. Na figura abaixo é ilustrada a ideia discutida:



Aqui, **p** representa um ponto no espaço da câmera, **p'** a projeção deste ponto, **c** a posição da câmera e **d** a distância da câmera. Assim, as coordenadas de **p'** são obtidas como:

$$\mathbf{p}' = (x', y', z', 1), \text{ onde } y' = \frac{y}{1 - \frac{z}{d}} \quad x' = \frac{x}{1 - \frac{z}{d}} \quad z' = \frac{z}{1 - \frac{z}{d}}$$

E a **matriz *projection*** tem a forma:

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

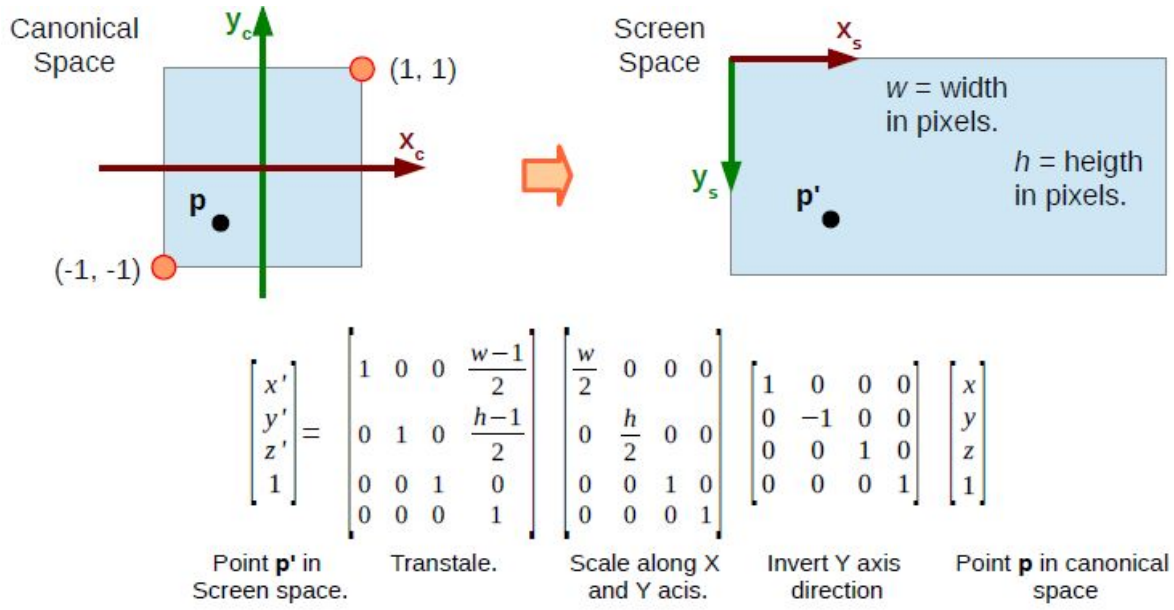
Aqui, a coordenada homogênea comumente assume valores diferentes de 1.

Espaço Projetivo ou de Recorte→ Espaço da Canônico

Nesta etapa, divide-se os valores das coordenadas que tínhamos no espaço projetivo pelos valores da coordenada homogênea, para que ela retorne a ser 1 e também transforma o *frustum* de visualização em um hexaedro. Em seguida, uma multiplicação por uma nova matriz, contendo escalas e translações, é realizada, de modo que este hexaedro agora terá coordenadas unitárias e os vértices estão agora no espaço canônico.

Espaço da Canônico → Espaço de Tela

Nesta parte, as coordenadas dos vértices, agora no espaço canônico, são multiplicadas por uma **matriz *ViewPort***. Esta matriz contém escalas e translações. Um exemplo é ilustrado a seguir:



Implementação

A implementação foi feita com auxílio da biblioteca **GLM** para a construção das matrizes ,da biblioteca **Assimp** para a importação dos vértices e para a parte da rasterização das retas foi utilizado o trabalho anterior da disciplina **[1,2].**

Foi criada uma struct chamada objeto que contém os vértices e os índices de ligação desses vértices.

```
struct objeto{

    std::vector<glm::vec4> vertices;
    std::vector<glm::vec2> indices;

};
```

Através da função um **objetoloader** a **struct objeto** é preenchida. Também foi criada a função **PipeLine** que recebe um ponteiro para essa **struct objeto**. A função **PipeLine** consiste das seguintes etapas:

Espaço do Objeto → Espaço do Universo

Como dito anteriormente a passagem do espaço do objeto para o espaço do universo pode ser feita com a combinação de varias matrizes para isso foi implementado as matrizes de todas as transformações possíveis como podemos observar abaixo:

```
9     //Model  Esp. Obj. --> Esp. Univ.
10    //Rotação
11    float Angle = 10.0f;
12
13    mat4 x_Rotetion = mat4(1,0,0,0,
14                           0,cos(Angle),-sin(Angle),0,
15                           0,sin(Angle),cos(Angle),0,
16                           0,0,0,1 );
17
18    mat4 y_Rotetion = mat4(cos(Angle),0,sin(Angle),0,
19                           0,1,0,0,
20                           -sin(Angle),0,cos(Angle),0,
21                           0,0,0,1 );
22
23    mat4 z_Rotetion = mat4(cos(Angle),-sin(Angle),0,0,
24                           sin(Angle),cos(Angle),0,0,
25                           0,0,1,0,
26                           0,0,0,1 );
27
```

```
30    //Shear
31    float mx = 0.0f;
32    float my = 0.0f;
33
34    mat4 shear = mat4(    1,0,mx,0,
35                          0,1,my,0,
36                          0,0,1,0,
37                          0,0,0,1 );
```

```
41    //Escala
42    float sx = 0.5f;
43    float sy = 0.5f;
44    float sz = 0.5f;
45    mat4 scale = mat4(    sx,0,0,0,
46                          0,sy,0,0,
47                          0,0,sz,0,
48                          0,0,0,1 );
```

```
53      //Translação
54      float dx = 1.0f;
55      float dy = 0.0f;
56      float dz = 0.0f;
57      mat4 translate = mat4(1,0,0,dx,
58                            0,1,0,dy,
59                            0,0,1,dz,
60                            0,0,0,1 );
```

Espaço do Universo → Espaço da Câmera

Foram definidos os parâmetros da câmera e partir deles foram gerados todos os outros parâmetros necessários, como a base da câmera no espaço do universo e a translação necessária para fazer a origem do espaço da câmera e do espaço do universo coincidirem, para por fim ser criada a matriz view.

```
//Parametros da camera

vec3 camera_pos =vec3(0,0,5);   //Posicao da camera no universo.
vec3 camera_lookat =vec3(0,0,0);  // Ponto para onde a camera esta olhando.
vec3 camera_up = vec3(0,1,0);  // 'up' da camera no espaco do universo.
```

```
//Calculo do sistema ortonormal gerado a partir dos parametros da camera

vec3 camera_dir = vec3(camera_lookat - camera_pos);

vec3 z_camera = normalize(-(camera_dir));
vec3 x_camera = normalize(cross(camera_up,z_camera));
vec3 y_camera = cross(z_camera,x_camera);
```

```
//Construcao da matriz view: Esp. Univ. --> Esp. Cam.

mat4 Bt  =  mat4(x_camera.x,x_camera.y,x_camera.z,0,
                 y_camera.x,y_camera.y,y_camera.z,0,
                 z_camera.x,z_camera.y,z_camera.z,0,
                 0,0,0,1);

mat4 T = mat4(1,0,0,-camera_pos.x,
              0,1,0,-camera_pos.y,
              0,0,1,-camera_pos.z,
              0,0,0,1);

mat4 M_view =  T * Bt ;
```

Espaço da Câmera → Espaço Projetivo ou de Recorte

No caso desse trabalho essa parte foi mais simples pois a nossa matriz de projeção é mais rudimentar tendo apenas o near plane com isso para a construção dela determinamos a distância entre o centro de projeção e o near plane e criamos a matriz de projeção.

```
//Construcao da matriz de Projecao: Esp. Cam. --> Esp. Recorte

float d = 2 ; // distância do centro de projeção para o viewplane

mat4 M_projection = mat4(1,0,0,0,
                         0,1,0,0,
                         0,0,1,d,
                         0,0,-(1/d),0);
```

Espaço Projetivo ou de Recorte→ Espaço da Canônico

Como dito anteriormente para essa mudança de espaço basta apenas dividir todas as coordenadas pelo valor da coordenada homogênea, incluindo ela mesma, para que seu valor volte a ser 1:

```
 //Esp. Recorte --> Esp. Canonico

for(int i = 0; i < obj->vertices.size(); i++){

    obj->vertices[i] = vec4(obj->vertices[i] / obj->vertices[i].w);
}
```

Espaço da Canônico → Espaço de Tela

Foi criado uma escala com fator de escala -1 em Y para o espelhamento em Y do objeto, também uma translação com fator de translação 1 em X e 1 em Y para todas as coordenadas ficarem positivas e uma escala fator de escala de metade do tamanho da tela em X e o mesmo em Y para que todas as coordenadas X e Y fiquem com valores possíveis na tela depois disso foram multiplicadas essas três matrizes para a criação da viewport:

```
//contrução da M_viewport Esp. Canônico --> o Esp. de tela.

mat4 Scale_1 = mat4(1,0,0,0,
                    0,-1,0,0,
                    0,0,1,0,
                    0,0,0,1);



mat4 traslete_1 = mat4( 1,0,0,1,
                        0,1,0,1,
                        0,0,1,0,
                        0,0,0,1);



int w = 512;
int h = 512;

mat4 Scale_2= mat4(w/2,0,0,0,
                   0,h/2,0,0,
                   0,0,1,0,
                   0,0,0,1);



mat4 M_viewport =  Scale_1*traslete_1*Scale_2;
```

Espaço do Objeto → Espaço de Tela

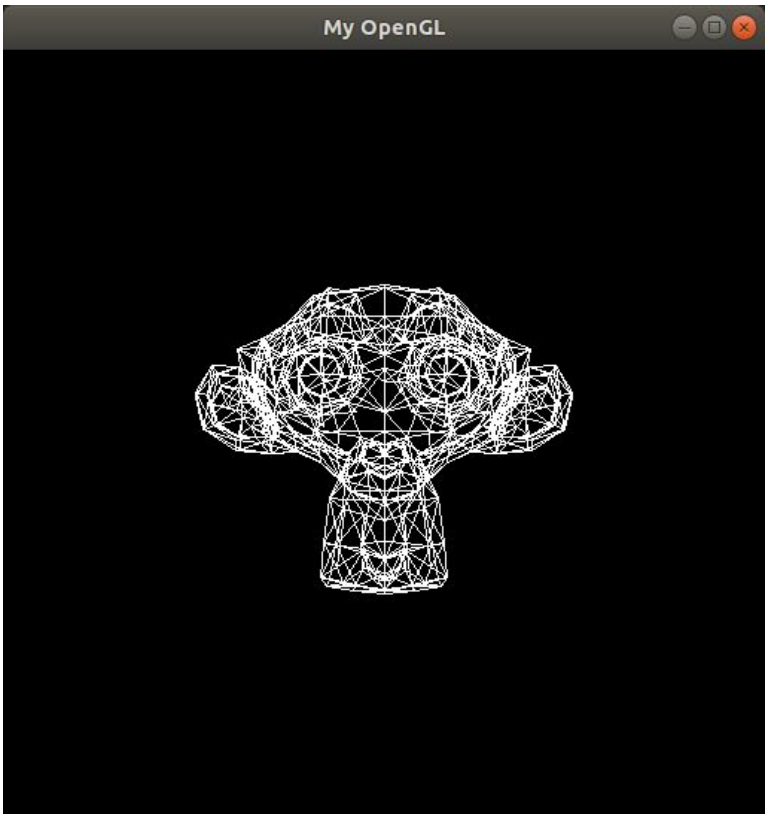
Com as matrizes ***model***, ***view***, ***projection*** e ***viewport*** criadas, foi necessário apenas fazer a multiplicação da ***model***, ***view*** e ***projection*** para criar a ***ModelView_Projection.*** Em seguida, basta multiplicar os vértices do objeto por essa matriz, depois dividir todas as coordenadas pela coordenada homogênea w e multiplicar pela ***Viewport*** para obter os vértices em coordenadas de tela.

```
//Construcao da matriz ModelViewProjection: Esp. Obj. --> Esp. Recorte

 mat4 M_model_view_proj =M_model_view * M_projection ;


 for(int i = 0; i < obj->vertices.size(); i++){

     obj->vertices[i] = vec4(obj->vertices[i] * M_model_view_proj );
 }

  //Esp. Recorte --> Esp. Canonico
 for(int i = 0; i < obj->vertices.size(); i++){

     obj->vertices[i] = vec4(obj->vertices[i] / obj->vertices[i].w);
 }


 //Esp. Canonico --> Esp. de Tela

 for(int i = 0; i < obj->vertices.size(); i++){

     obj->vertices[i] = vec4(floor(obj->vertices[i] * M_viewport) );
 }
```

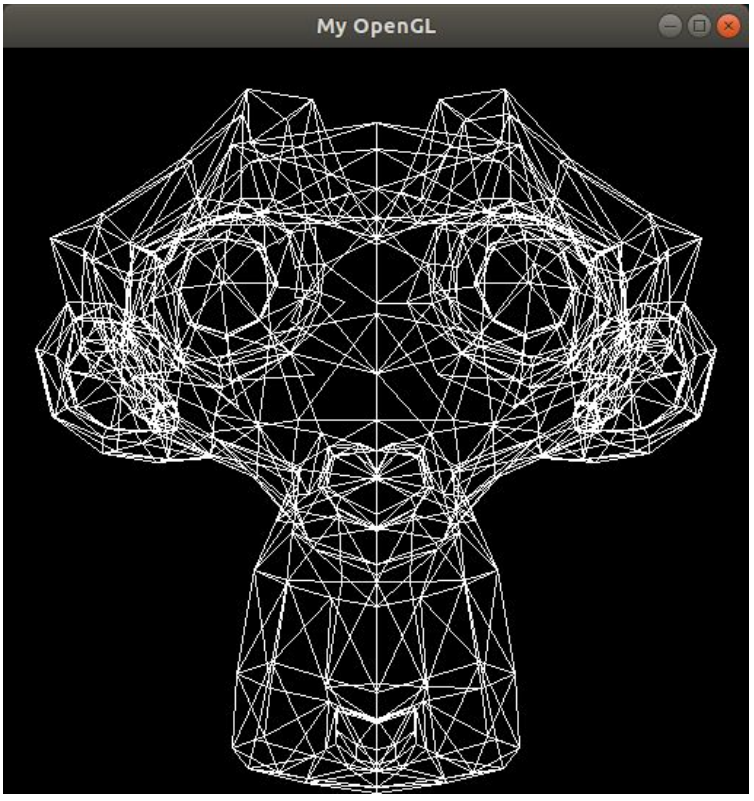
Com os vértices em coordenadas de tela agora só é necessário rasterizar as linhas para obter a imagem do nosso objeto.

Demonstrações dos resultados obtidos

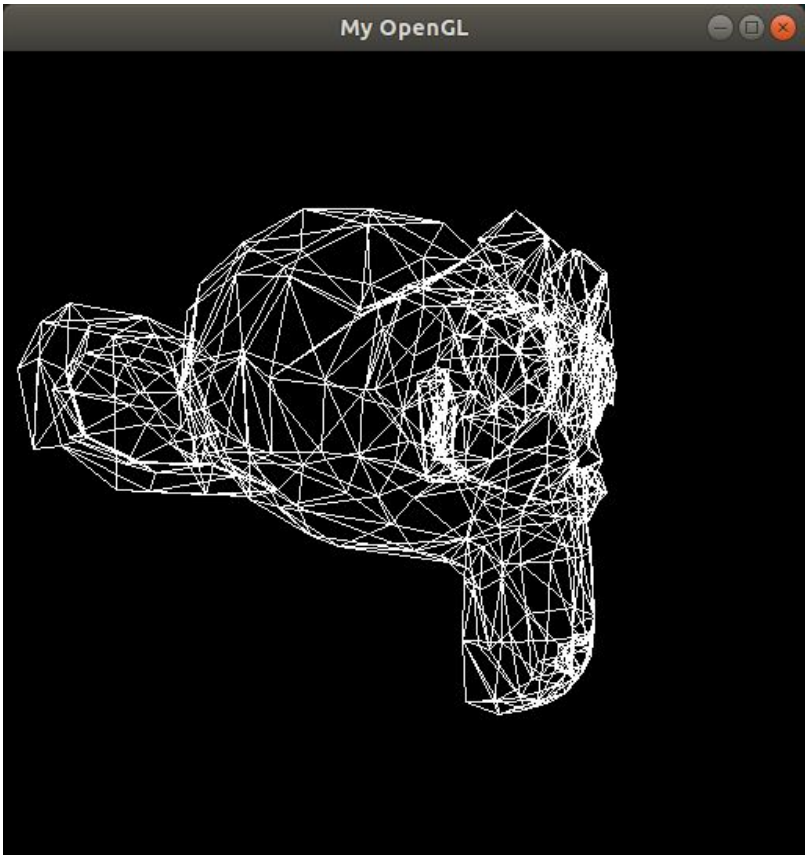
Sem nenhuma transformação (escala, rotação, translação ou *shear*) na Model:



Escala com fator 2 em X , Y e Z:



Escala com fator de escala 1,5 em X , Y e Z e rotação de 45º em Y:



Referências

[1] **LOPES, A. R.** *1° trabalho prático - Introdução a Computação Gráfica - 2018.2* .
UFPB.
Disponível em: https://github.com/Rekran/CGProjeto
Último acesso: 22/04/2019

[2] - **BARBOSA, G. M.** *1º Trabalho prático - Introdução a Computação Gráfica* - 2018.2.
UFPB.
Disponível em: https://github.com/GMarques1958/CG-Trabalho1-2018.2.
Último acesso: 22/04/2019.

[3] - **PAGOT, C. A.** *Slides das aulas de Introdução a Computação Gráfica. 2018.2.* UFPB.

[4] - **CARVALHO, M. A. G.** *Slides das aulas de Computação Gráfica: Projeções.* Faculdade de Tecnologia da Unicamp.
Disponível em:https://www.ft.unicamp.br/~magic/ST765/CG2009_Projecoes.pdf.
Último acesso: 22/04/2019.