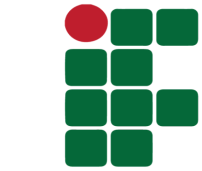

INSTITUTO
FEDERAL
Ceará

Representação de sólidos

Coordenadas do Objetos
Coordenadas do Mundo
Coordenadas da câmera
Projeção em Perspectiva

Computação Gráfica

Ajalmar Rego da Rocha Neto

Edilson Gonçalves Tavares
Vitor Luca Chagas

Fortaleza 2023

1. Introdução 3
2. Desenvolvimento 4
2.1 Sólidos 4
2.2 Sistema de coordenadas 4
2.2.1 Sistemas de coordenadas do objeto 5
2.1.1.1 Cilindro 5
2.1.1.2 Cone 7
2.1.1.3 Esfera 7
2.1.1.4 Tronco de pirâmide 8
2.1.1.5 Cubo 9
2.1.1.6 Toroide 10
2.2.2 Sistemas de coordenadas do mundo 11
2.2.3 Sistemas de coordenadas da câmera 12
3. Projeção em perspectiva 15
4. Conclusão 18
5. Referências bibliográficas 19

1. Introdução

Este trabalho tem como objetivo mostrar e analisar resultados obtidos na modelagem de sólidos em diversos sistemas de coordenadas. Para a realização deste trabalho foram utilizados conceitos apresentados na aula de computação gráfica, tais como matrizes, transformações e sistemas de coordenadas. Seu desenvolvimento foi feito na linguagem python e para a geração de imagens e manipulações matemáticas foram utilizadas as bibliotecas Matplotlib e Numpy respectivamente.
O relatório apresenta os resultados obtidos e faz a análise dos pontos propostos, assim como observações e ajustes necessários a se fazer.

2. Desenvolvimento

2.1 Sólidos

Um sólido é um subconjunto fechado e limitado do espaço tridimensional Euclidiano. Existem várias formas de representar um sólido, cada uma com as suas vantagens e desvantagens. Muitas vezes a solução ideal para a representação depende do contexto e pode ser feita de forma híbrida, isto é, conter mistura de alguns métodos mais simples e usados. As representações mais comuns são, representação aramada (wire frame), representação por faces (ou superfícies limitadas) e representação por faces poligonais.
A representação dos sólidos neste trabalho foi a representação aramada em que os objetos são descritos por um conjunto de arestas que definem a borda dos objetos. Esta forma é uma extensão em três dimensões do método de representação de objetos por contorno.
Sua principal vantagem é a simplicidade na exibição dos modelos, sendo necessário apenas exibir um conjunto de linhas conectando os vértices.

2.2 Sistema de coordenadas

A matemática, assim como qualquer outra linguagem, cria convenções para que, a partir dela, seja possível construir formulações mais complexas. Na base dos sistemas gráficos está a abstração de um sistema de coordenadas cartesiano onde é traçado no plano duas retas concorrentes, ditas eixos do x (eixo das abscissas) e y (eixo das ordenadas).

A abstração matemática dita sistema de coordenadas é explorada pela computação gráfica como ferramenta que permite escolher e alterar a representação de objetos gráficos da maneira que for mais conveniente a cada operação de processamento visual. No intuito de facilitar as tarefas envolvidas no processo de renderização de uma cena 3D, é comum a definição e uso dos seguintes sistemas de coordenadas, chamadas de sistemas de referência do pipeline gráfico:

- Sistemas de coordenadas do objeto
- Sistemas de coordenadas do Mundo
- Sistema de coordenadas da Câmera
- Sistema de coordenadas de Normalizado
- Sistema de coordenadas dos dispositivos

Neste trabalho o foco será apresentado apenas os 3 primeiros sistemas. O objeto é inicialmente composto sozinho, em seu sistema de coordenadas, em seguida formado uma cena ou mundo com vários objetos e por fim com a localização da câmera no mundo, podemos mostrar a cena através do sistema de coordenadas da câmera.

2.2.1 Sistemas de coordenadas do objeto

Os objetos foram construídos de maneira que cada um estivesse descrito em termos de seu próprio Sistema de Coordenadas do Objeto (SCO). Esse sistema de coordenadas é local ao objeto e é usado para descrever as posições relativas dos pontos e orientações dentro do próprio objeto. Para representar esses sólidos, foram criadas funções que criam e retornam uma matriz de vértices e uma matriz de arestas como mostrado no exemplo da *Fig.1* abaixo.

```
def cubo(tamanho_aresta):
    # vertices do cubo
    vertices = np.array([
        [-tamanho_aresta / 2, -tamanho_aresta / 2, 0],
        [tamanho_aresta / 2, -tamanho_aresta / 2, 0],
        [-tamanho_aresta / 2, tamanho_aresta / 2, 0],
        [-tamanho_aresta / 2, -tamanho_aresta / 2, tamanho_aresta],
        [tamanho_aresta / 2, tamanho_aresta / 2, 0],
        [tamanho_aresta / 2, -tamanho_aresta / 2, tamanho_aresta],
        [-tamanho_aresta / 2, tamanho_aresta / 2, tamanho_aresta],
        [tamanho_aresta / 2, tamanho_aresta / 2, tamanho_aresta]
    ])

    # arestas do cubo
    # definir quais vertices estao ligados entre si na lista de vertices
    arestas = [
        [0, 1], [0, 2], [0, 3], [1, 4],
        [1, 5], [2, 4], [2, 6], [3, 5],
        [3, 6], [4, 7], [5, 7], [6, 7]
    ]

    return vertices, arestas
```

Figura 1: Exemplo de função para retornar vértices e arestas do cubo

2.1.1.1 Cilindro

Cilindro com altura igual a 2 vezes o raio da tampa e com 10 divisões intermediárias entre as tampas. Para a geração dos pontos utilizamos a equação paramétrica do círculo dada por:

$x(r, \theta) = r \cdot cos(\theta)$
$y(r, \theta) = r \cdot sin(\theta)$
$z(r, \theta) = z$

Onde θ varia de 0 a 2π, e z varia de 0 a h com intervalos definidos pela quantidade de pontos intermediários do círculo. Podemos notar na diferença entre a Figura 2 e Figura 3 que a

quantidade de pontos do círculo influência na suavidade da circunferência. Isso acontece porque ela determina a quantidade de ângulo theta gerados para criar os vértices do círculo. Portanto, uma maior quantidade de pontos o círculo será mais suave e com menos quantidade de pontos teria uma forma mais poligonal.

```
# Gera os ângulos theta para os vértices do círculo
theta = np.linspace(0, 2 * np.pi, num_fatias)
```

Figura 2: Influência do num_fatias ou pontos na formação do círculo

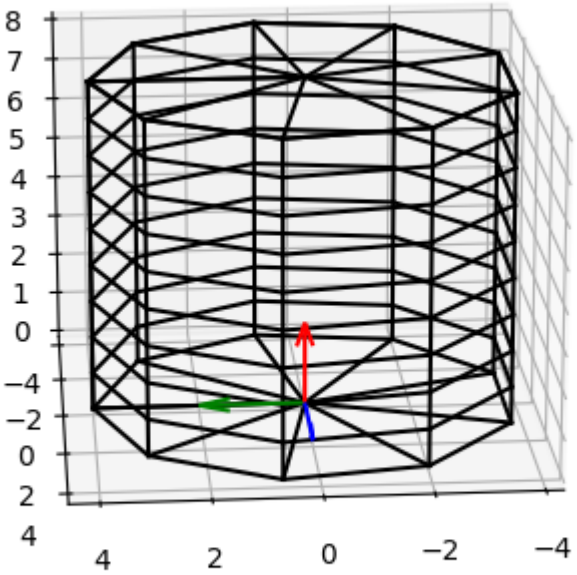

Figura 3: Modelagem do cilindro com r igual a 8 e com 10 pontos formando o círculo

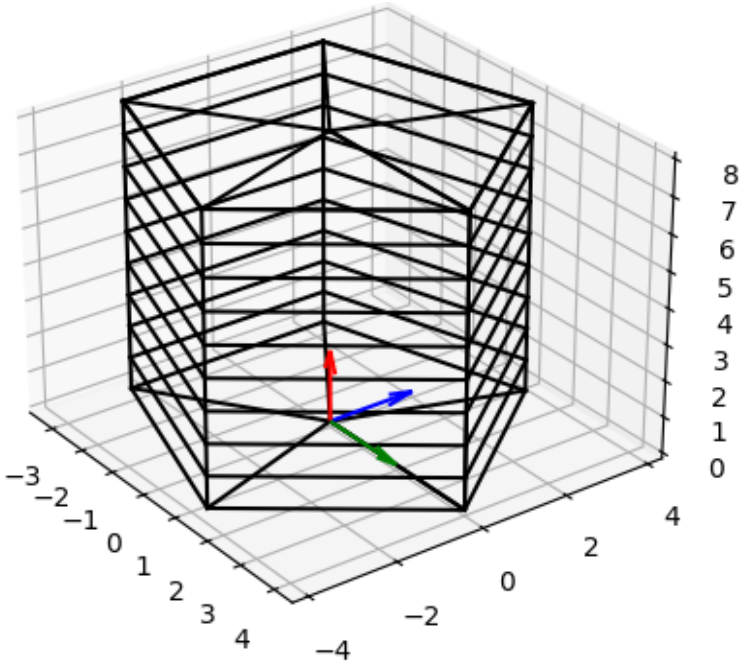

Fig 4 : Modelagem do cilindro com r igual a 8 e com 6 pontos formando o círculo

2.1.1.2 Cone

Cone com altura igual a três vezes o raio da tampa inferior e com 10 divisões intermediárias. A geração dos pontos se deu de maneira semelhante ao do cone, porém com o r variando de 0 a r.

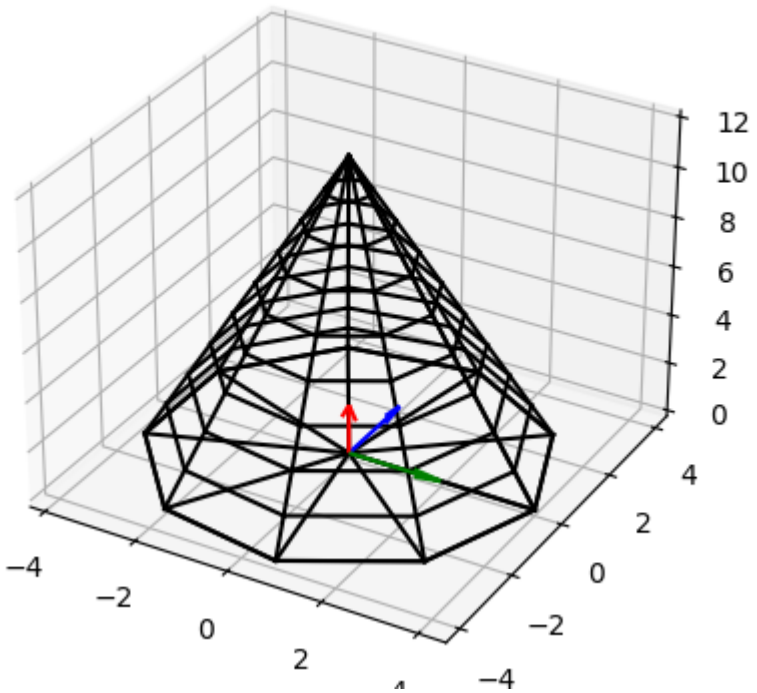

Figura 5: Modelagem do cone com r igual a 8

2.1.1.3 Esfera

Esfera definida através de seu raio r e com 10 divisões entre o menor e o maior ponto. Para descrever a esfera em coordenadas esféricas, utilizamos as seguintes equações paramétricas:

$x(r, \theta, \phi) = r \cdot sin(\phi) \cdot cos(\theta)$
$y(r, \theta, \phi) = r \cdot sin(\phi) \cdot sin(\theta)$
$z(r, \theta, \phi) = r \cdot cos(\phi)$

Onde r é o raio da esfera, θ é o ângulo azimutal (longitude) variando de 0 a 2π, e ϕ é o ângulo de inclinação (latitude) variando de 0 a π.

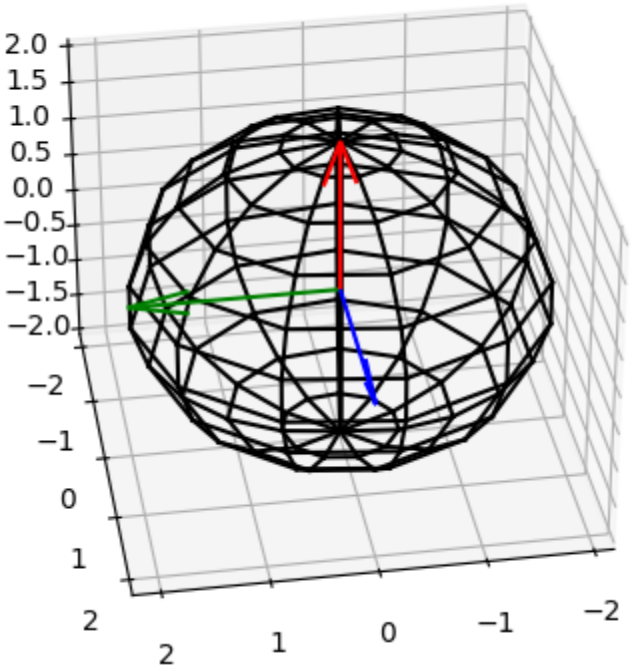

Figura 6: Modelagem da esfera com r igual a 4

2.1.1.4 Tronco de pirâmide

Tronco de pirâmide com altura 4, arestas da base inferior de tamanho 8 e arestas da base superior de tamanho 2.

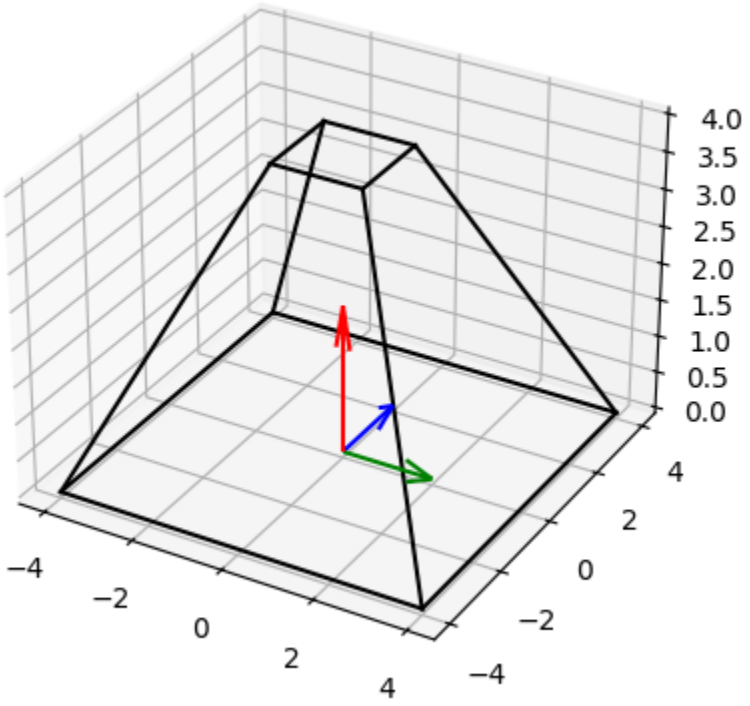

Figura 7: Modelagem do tronco de pirâmide

2.1.1.5 Cubo

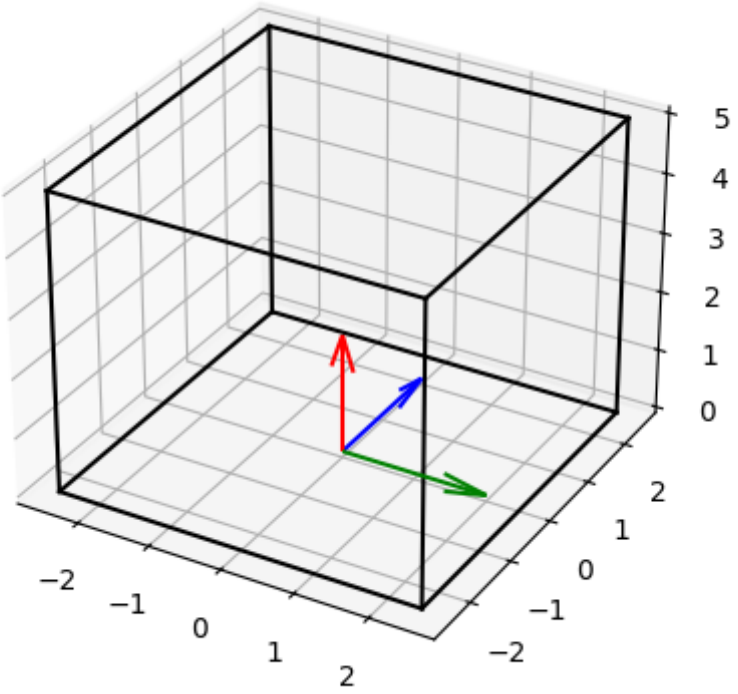

Figura 8: Modelagem do cubo com arestas de tamanho 4

2.1.1.6 Toroide

Toroide definido por R e r onde R é o raio da parte central do toróide (distância do centro até o centro do tubo), r é o raio do tubo. Em coordenadas paramétricas o toroide pode ser descrito por:

$$x(u,\ v)\ =\ (R\ +\ r\ \cdot\ cos(v))\ \cdot\ cos(u)$$
$$y(u,\ v)\ =\ (R\ +\ r\ \cdot\ cos(v))\ \cdot\ sin(u)$$
$$z(u,\ v)\ =\ r\ \cdot\ sin(v)$$

Onde u e v variam de 0 a 2π. As variáveis u e v representam os ângulos paramétricos que definem a posição de um ponto na superfície do toróide.

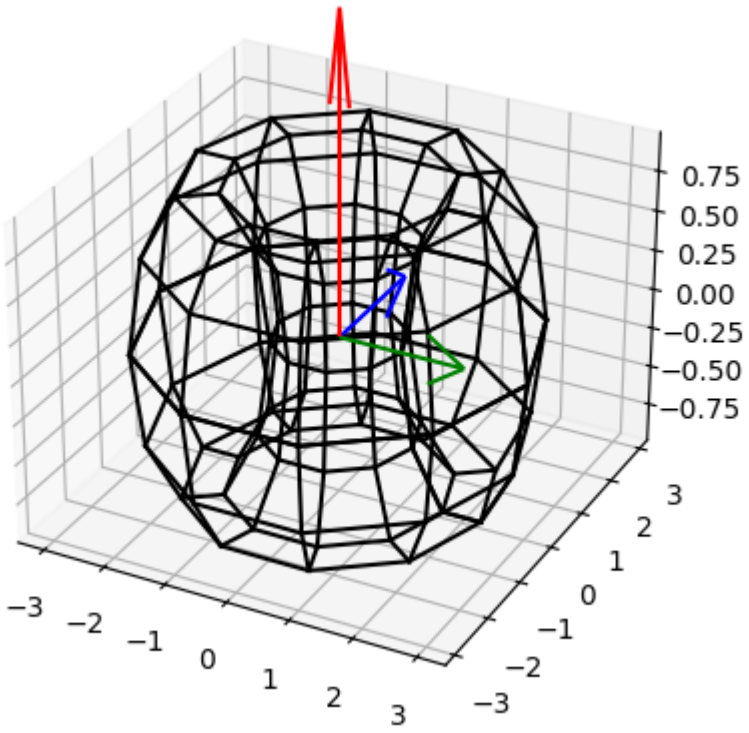

Figura 9: Modelagem do toro com r igual a 1 e R igual a 2

2.2.2 Sistemas de coordenadas do mundo

A cena ou cenário é formada pela composição de objetos posicionados e ajustados entre si em um ambiente virtual. Para possibilitar essa composição, é usado um Sistema de Coordenadas do Mundo (SCM) para estabelecer uma referência única e universal. Utilizamos transformações, como translação, rotação e escala em coordenadas homogêneas no espaço tridimensional R^3, para inserir os objetos modelados no Sistema de Coordenadas do Objeto no Sistema de Coordenadas do Mundo. Sendo as matrizes dadas por:

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz de escala

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz de translação

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

Matrizes de rotação em x, y e z

A composição dos objetos, no sistema de coordenadas do mundo, respeitou as seguintes condições:

A. O cilindro, o cubo e a esfera devem estar localizados em apenas um octante do espaço.
B. O cone, o tronco de pirâmide e o toro devem estar em apenas um octante, mas não no mesmo do cilindro e da esfera.
C. O maior valor possível para cada uma das componentes de um vértice no sistema de coordenadas do mundo deve ser igual a 10.

Inserimos o cubo, a pirâmide e a esfera no octante 1, ou seja, x > 0, y > 0 e z > 0 para todos os pontos dos objetos. Já o tronco de pirâmide, o cone e o toro foram colocados no octante 2 onde x < 0, y > 0 e z > 0 para todos os pontos. Observa-se ainda na Figura 10 que o toro se encontra um pouco distorcido e isso se deve ao fato de que antes de ser feito a escala e translação o objeto foi rotacionado em torno do eixo do y com um ângulo de (-pi/4), ou seja, no sentido horário no valor de 45 graus.

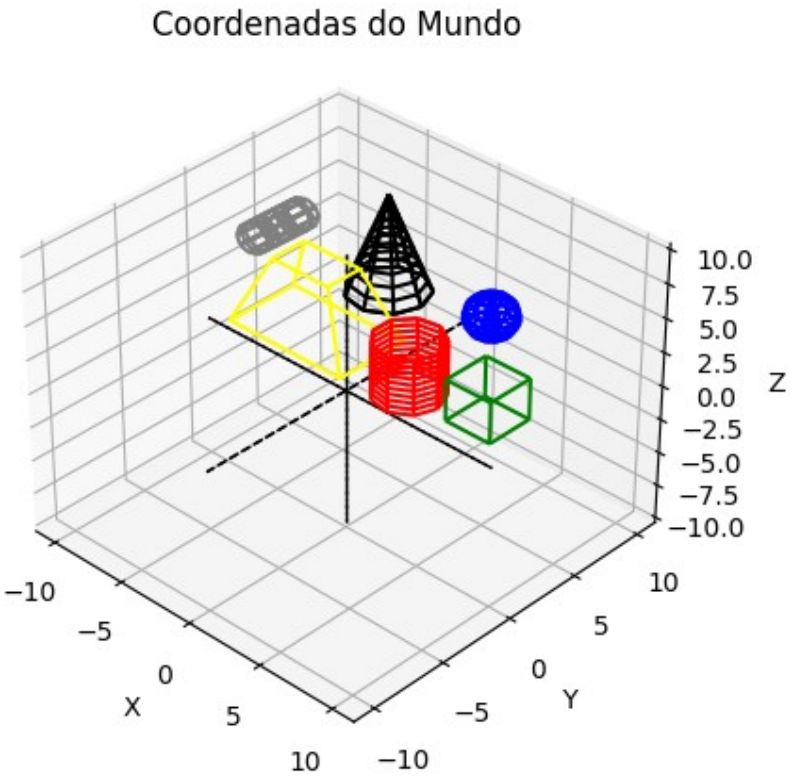

Figura 10: Sólidos no Sistema de Coordenadas do Mundo

2.2.3 Sistemas de coordenadas da câmera

O sistema de coordenadas da câmera é uma representação especial do espaço tridimensional que leva em conta a posição e orientação da câmera em relação ao mundo. Ele fornece um referencial para interpretar as projeções de objetos do mundo tridimensional para o plano de imagem da câmera.

O sistema de coordenadas da câmera é geralmente definido da seguinte maneira

Origem da Câmera (Eye ou Posição da Câmera): A origem do sistema de coordenadas da câmera é geralmente definida pela posição da câmera ou pelo ponto onde a câmera está localizada no espaço tridimensional. Essa posição é frequentemente representada pelo vetor (eye_x, eye_y, eye_z). Para o trabalho utilizamos o eye como: $(5,\ 5,\ -\ 8)$ ou seja, posicionada no quinto octante onde x > 0, y > 0 e z < 0 como mostra a Figura 11.

Vetor de Visualização (Look-At): Este vetor define a direção na qual a câmera está apontando. Ele é representado pelo vetor (at_x, at_y, at_z), que indica o ponto para o qual a câmera está direcionada. Como solicitado no trabalho utilizamos o vetor at como sendo o

centro de massa dos objetos, no caso selecionamos o octante 1 para ser o volume de visão onde estão localizados o cilindro, cubo e a esfera.

Vetor Up (para cima): Este vetor define a orientação vertical da câmera. Geralmente, é um vetor perpendicular à direção de visualização. Ele é representado pelo vetor (up_x, up_y, up_z), no trabalho utilizamos o vetor (0, 0, 1).

A construção do sistema de coordenadas da câmera envolve a determinação desses vetores e a definição de uma base ortogonal (conjunto de vetores mutuamente perpendiculares) que define a orientação da câmera. O vetor de visualização é normalmente normalizado para garantir que ele tenha comprimento 1.

Para obtermos a base ortogonal computamos o vetor $\overrightarrow{n}$ que determina a direção normal ao plano de projeção dado pela diferença do ponto de visão da câmera (eye) e o centro de massa dos sólidos (at), normalizados:

$$\vec{n} = \frac{eye - at}{\|eye - at\|}$$

O segundo vetor pode ser obtido com base no cômputo vetorial entre $\vec{n}$ e $\vec{up}$ $(\vec{n} \text{ x } \vec{up})$ obtendo assim o vetor $\vec{u}$ que é simultaneamente ortogonal a $\vec{n}$ e $\vec{up}$, após isso, normaliza-se o vetor de forma similar feita a $\vec{n}$:

$$\vec{u} = \frac{\vec{n} \text{ x } \vec{up}}{\|\vec{n} \text{ x } \vec{up}\|}$$

O terceiro vetor é obtido então pelo cômputo do produto vetorial de $\vec{n}$ e $\vec{u}$ também normalizado:

$$\vec{v} = \frac{\vec{n} \text{ x } \vec{u}}{\|\vec{n} \text{ x } \vec{u}\|}$$

Dessa forma, com os vetores $\vec{u}$, $\vec{n}$ e $\vec{v}$ temos a base do espaço vetorial para o sistema de coordenadas da câmera.

A matriz de visualização é frequentemente usada para representar a transformação do sistema de coordenadas do mundo para o sistema de coordenadas da câmera. Essa matriz leva em consideração a posição e orientação da câmera no espaço tridimensional. Para criar a matriz de transformação do mundo para a câmera, precisamos combinar a matriz de rotação (R) e a matriz de translação (T). A matriz de visualização (V) é a multiplicação da matriz de rotação pela matriz de translação.

$$R = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eyez \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ -n_x & -n_y & -n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$V = R \bullet T$

A câmera aponta ao longo do eixo z negativo do sistema de coordenadas do mundo, de modo que quando um ponto é convertido do espaço do mundo para o espaço da câmera (e posteriormente do espaço da câmera para o espaço da tela), se o ponto estiver à esquerda do eixo y do sistema de coordenadas do mundo, o ponto também será mapeado à esquerda do eixo y do sistema de coordenadas da câmera. Em outras palavras, precisamos que o eixo x do sistema de coordenadas da câmera aponte para a direita quando o eixo x do sistema de coordenadas do mundo também aponta para a direita.
Por conta disso, o sinal da coordenada z dos pontos é invertido quando passamos de um sistema para outro.

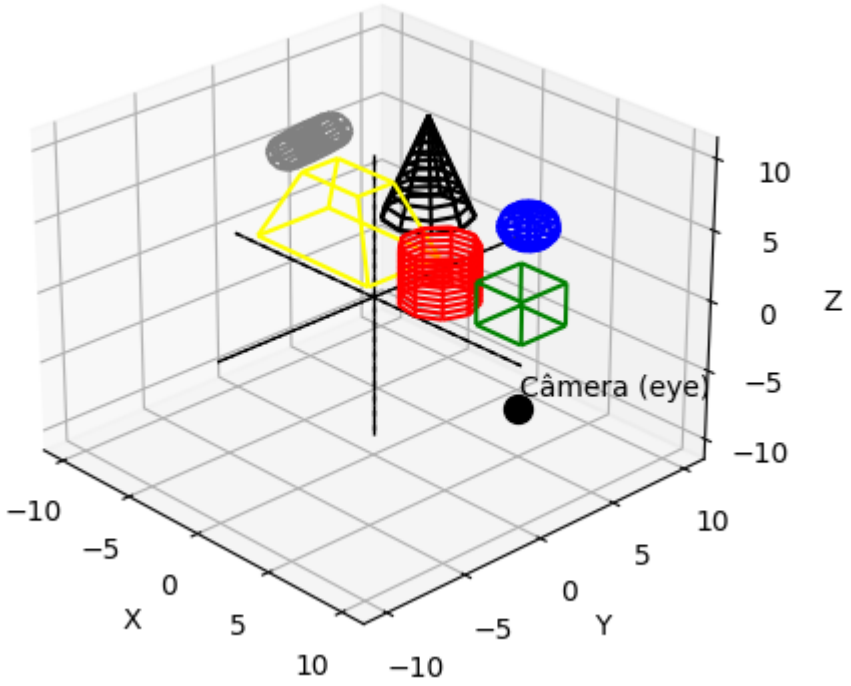

Figura 11: Objetos no sistema de coordenadas do mundo com o eye mostrando onde a câmera está localizada.

Para verificarmos se a transformação foi feita de maneira correta, verificamos se a distância de um ponto de um sólido até a origem da câmera (eye) no Sistema de Coordenadas do mundo, é igual à distância entre o mesmo ponto e a origem do Sistema de Coordenadas da Câmera , então a transformação ocorreu de forma correta e tendo em conta que as transformações de rotação e translação preservam a distância entre dois pontos.

```
Distância entre um ponto do cubo e a origem da câmera (eye): 8.74642784226795
Distância entre um ponto do cubo e a origem do sistema de coordenadas da câmera: 8.74642784226795
```

Figura 12: Verificação após transformações

Também plotamos um ponto representando a origem no Sistema de Coordenadas do Mundo agora no Sistema de Coordenadas da Câmera para verificar se os pontos continuavam coerentes após as transformações.

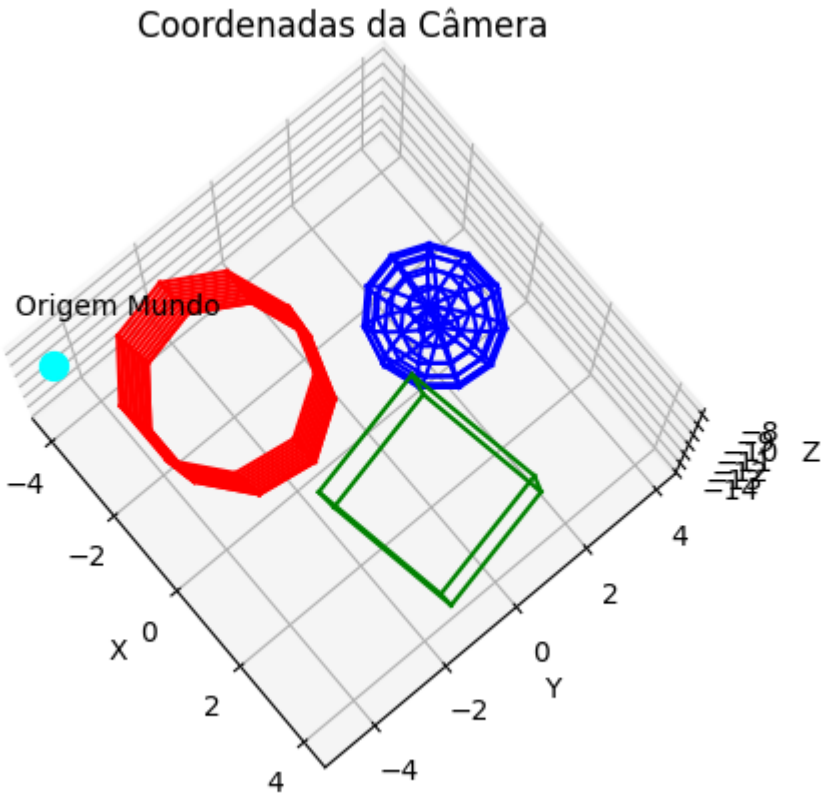

Figura 13: Objetos no sistema de coordenadas da Câmera, o ponto azul representa a origem no sistema de coordenadas do mundo agora convertido para o sistema de coordenadas da câmera

3. Projeção em perspectiva

Uma vez que os objetos passem a ser representados no sistema de coordenadas da câmera, a próxima tarefa do processo de transformação de formação de imagem é a transformação de tais coordenadas 3D em coordenadas 2D (sobre o plano de imagem), obtidas pela transformação dita projeção.
Assume-se que nessa etapa a transformação é feita para um sistema padrão, sem se preocupar com características específicas do objeto.
O modelo de projeção mais utilizado para renderização de cenas 3D é o de Projeção em perspectiva, o qual simula a forma como observamos o mundo: quando mais longe um objeto, menor o observamos.
Observe ainda que na construção da câmera de orifício os objetos aparecem de cabeça para baixo e espelhadas no plano de projeção como mostra a figura 13.

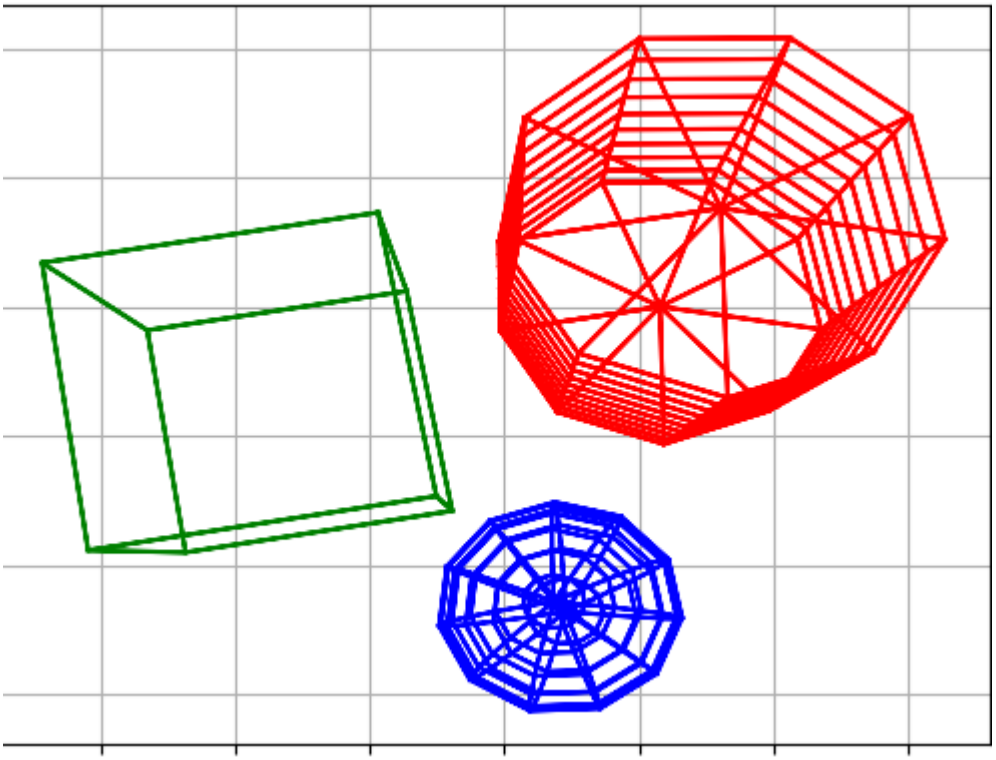

Figura 14: Objeto projetado em perspectiva de cabeça para baixo

Embora seja uma limitação da física da construção de câmaras de orifícios, em câmeras virtuais a gente pode resolver esse problema na etapa da construção das coordenadas de câmera onde a gente inverte o sinal do vetor n (vetor que diz a direção onde a câmera está olhando), garantindo que os objetos fossem mostrados como se tivessem atrás da câmera e com isso a projeção não aparece de forma invertida.

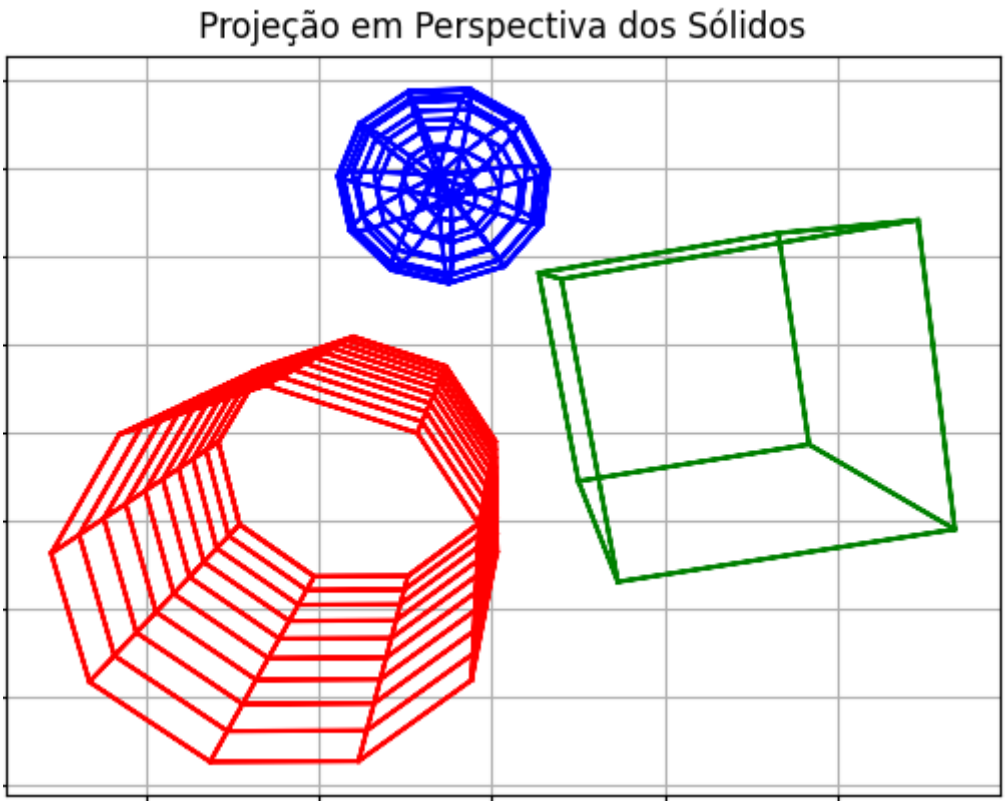

Figura 15: Objeto na projeção em perspectiva

Para a geração dos pontos na projeção em perspectiva foi utilizado uma matriz de projeção apresentado na figura 14 e multiplicado pelos pontos que fazem parte do campo de visão da câmera, onde os pontos são representados em coordenadas homogêneas, permitindo a multiplicação de matrizes.
Ela é definida por quatro parâmetros: **near**, **far**, **largura** e **altura**. Os parâmetros near e far são importantes porque determinam a profundidade de campo da imagem resultante, enquanto os parâmetros largura e altura são importantes porque determinam a proporção de aspecto da imagem resultante.

```
# Defina a matriz de projeção em perspectiva
matriz_projecao = np.array([
    [near / (largura / 2), 0, 0, 0],
    [0, near / (altura / 2), 0, 0],
    [0, 0, -(far + near) / (far - near), -2 * far * near / (far - near)],
    [0, 0, -1, 0]
])
```

Figura 16: Matriz de projeção em perspectiva

4. Conclusão

O objeto representado no sistema de coordenadas local só precisamos de calcular os vértices e fazer a ligação entre elas, sem se preocupar com mais detalhes, visto que só elas estão presentes na cena. Ainda em alguns sólidos vimos que a quantidade de pontos da circunferência influenciam na suavidade da circunferência. Desta forma tendo os objetos representados em suas coordenadas pudemos formar uma cena, onde por meio de transformações vistas em sala de aula aplicamos elas sobre os objetos formando sistema de coordenadas do mundo respeitando as regras evitando sobreposições e fazendo escalonamento para respeitar as regras do maior valor permitido.
Para a transformação em sistemas de coordenadas da câmera escolhemos o quinto octante como mostrado na figura 11, para ser a localização da câmera e o campo de visão apenas o primeiro octante, desse jeito os sólidos que estão fora são consideradas fora do volume de visão não sendo projetadas nas coordenadas da câmera.
Na projeção em perspectiva tivemos que fazer alguns ajustes nas matrizes para que a imagem possa ser mostrada sem percepção de estar invertida como apresentada na figura 13.
Observamos ainda que o objeto mais distante da câmera, que é a esfera, é menor do que os outros, respeitando a regra da transformação em perspectiva.

5. Referências bibliográficas

Aura Conci, Cristina Vasconcelos, Eduardo Azevedo. Computação Gráfica Teoria e prática: Geração de Imagens. vol. 1

Aula 07: Projeção. Disponível em <http://profs.ic.uff.br/~anselmo/cursos/CGI/slidesNovos/projecao.pdf>. Acesso em: 09 de dez. de 2023

Matplotlib: Python Plotting –- Matplotlib 3.8.2 documentation. Disponível em <https://matplotlib.org/stable/tutorials/pyplot.html>. Acesso em: 08 de dez. de 2023

Scratchapixel. Computing Pixel Coordinates of 3D Point. Scratchapixel. <https://www.scratchapixel.com/lessons/3d-basic-rendering/computing-pixel-coordinates-of-3d-point/mathematics-computing-2d-coordinates-of-3d-points.html#:~:text=To%20convert%20points%20from%20world,coordinate%20system's%20negative%20z%2Daxis> Acesso em: 10 de dez. de 2023