Curso de Jogos Digitais

Disciplina de Computação Gráfica

Pipeline da Visualização 3D

Aula 05

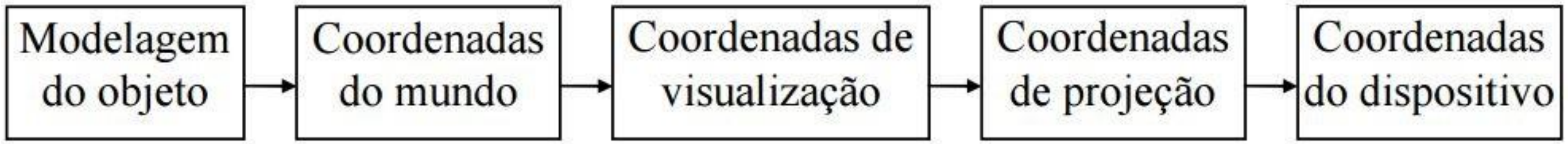
Professor: André Flores dos Santos

Santa Maria – RS 2025

SUMÁRIO

Introdução

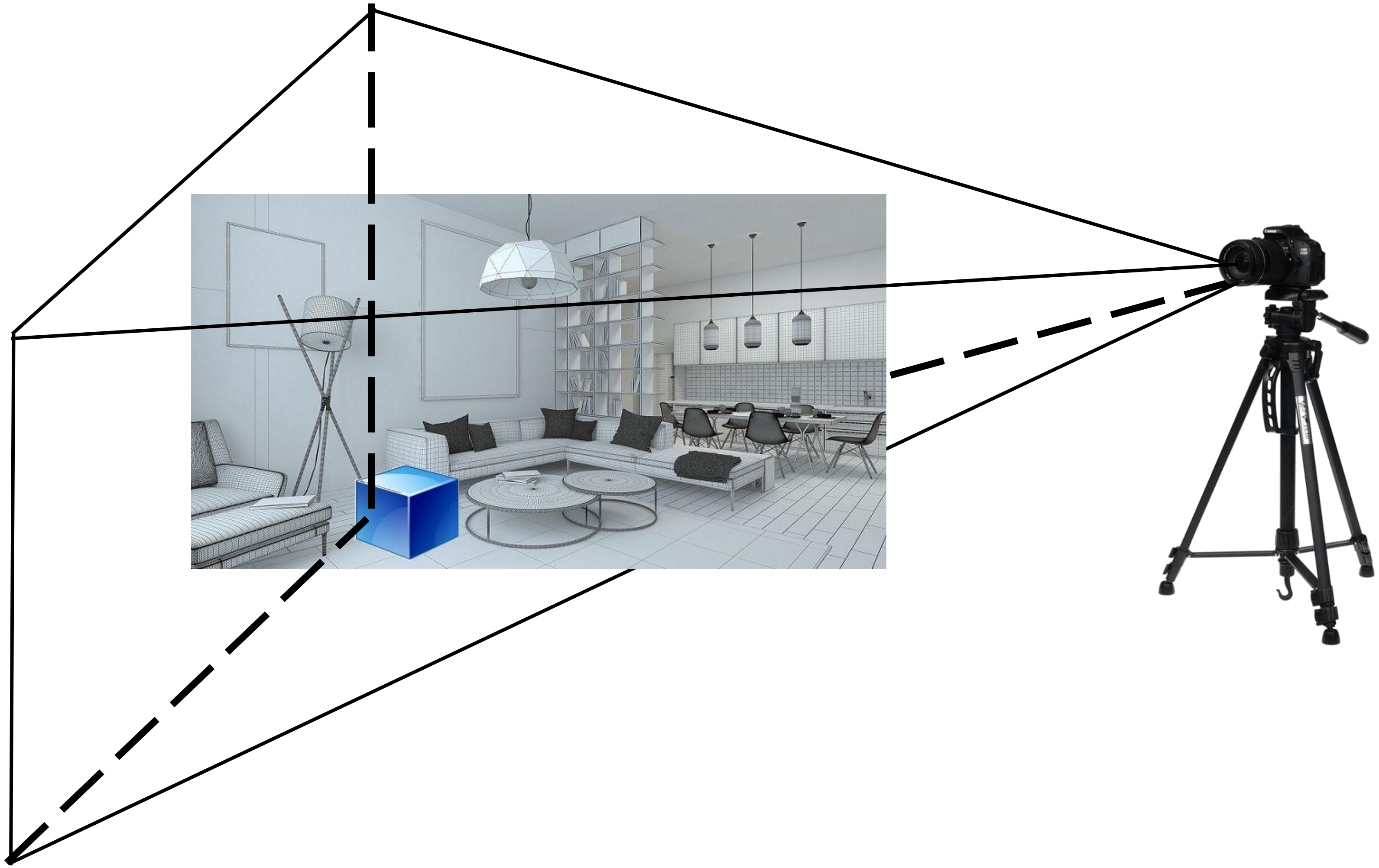
- O processo de visualização de uma cena 3D por computador é semelhante ao processo envolvido para se tirar uma fotografia:
  - Deve-se posicionar a câmera no espaço e definir sua orientação
    - 6 graus de liberdade: 3 eixos de translação e 3 eixos rotação.
- Como a maioria dos monitores existentes são bidimensionais, devemos fazer uso de processos para **conversão** de objetos do **espaço tridimensional** para uma **representação bidimensional.**
  - Este processo possui um termo conhecido como three-dimensional pipeline, ou pipeline 3D.
- As etapas deste pipeline são mostradas na seguinte figura, e, incluem **modelagem**, **visualização** e **conversão** de diferentes tipos de coordenadas:



2) Coordenadas do mundo (transformações geométricas)

3) Coordenadas de visualização (transformações geométricas).
O modelo é posicionado em relação com a câmera (view Space)

4) Coordenadas de projeção



3D → **2D**

Aqui ocorre a conversão de um espaço **tridimensional** para um **espaço bidimensional**.

5) Coordenadas do dispositivo (mapeamento)

Recorte (Clipping): Remove partes do objeto que estão fora do campo de visão.

Pipeline 3D

- **1) Modelagem do objeto:**
  - É neste passo em que definimos os **modelos de objetos** no seu “mini universo” próprio, cujo **centro de gravidade** geralmente é a **origem** do sistema de coordenadas;

- **2) Coordenadas do mundo:**
  - Após a definição dos modelos de objetos, realizamos o instanciamento dos mesmos e aplicamos **transformações geométricas** (translação, escala, rotação...) aos mesmos, de modo a posicioná-los no mundo/universo.

  - A matriz que representa a combinação de transformações geométricas que trazem os modelos para o universo é denominada **matriz de transformação do modelo**;

Pipeline 3D

- **3) Coordenadas de visualização:**
  - É nesta etapa onde definimos a chamada **câmera virtual**, que observa o universo a partir de uma determinada posição e orientação.
  - Por mais estranho que possa parecer, ao invés de posicionarmos uma câmera no universo, nós fazemos com que o **universo se posicione para a câmera.**
    - Ou seja, **não é a câmera que gira ao redor do mundo, e sim o mundo que gira ao redor da câmera.**
  - O posicionamento do universo em relação a câmera é feito, novamente, a partir de **transformações geométricas**;
  - A **matriz** que representa a combinação de transformações geométricas que fazem com que o universo se posicione em relação a câmera é denominada **matriz de visualização;**

Pipeline 3D

- **4) Coordenadas de projeção:**
  - É nesta etapa onde realizamos a **conversão** das coordenadas de visualização **3D** para o plano de projeção **2D**, que em seguida será mapeado para um dispositivo de saída (por exemplo, monitor);
  - Portanto, é nesta etapa do processo onde é realizada a conversão do espaço tridimensional para o bidimensional (tela do computador);
  - Esta transformação também é feita a partir de uma matriz de transformação, esta denominada **matriz de projeção**.
- **5) Coordenadas do dispositivo**:
  - Para realizar o **mapeamento** da imagem para a tela, realizamos operações de **recorte** da janela de seleção (window) para a janela de visualização (viewport).

Pipeline 3D

1) Modelagem do objeto

↓

2) Coordenadas do mundo

↓

3) Coordenadas de visualização

↓

4) Coordenadas de projeção

↓

5) Coordenadas do dispositivo

a. Matriz de transformação do modelo

b. Matriz de visualização

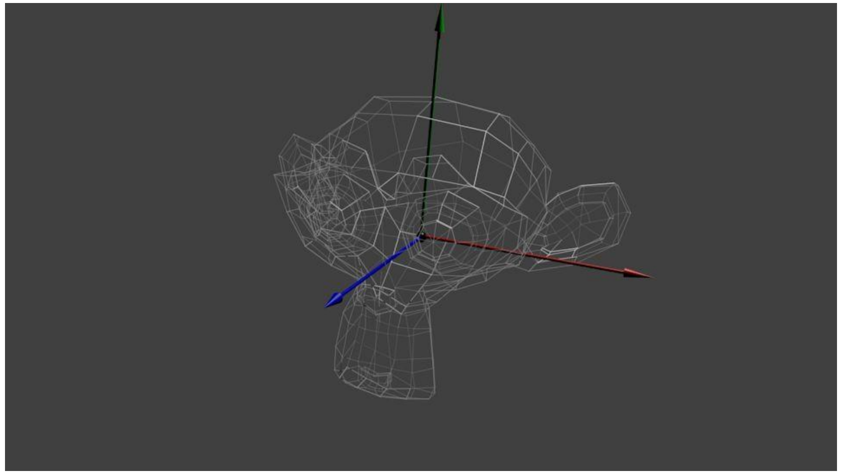
c. Matriz de projeção

d. Recorte, mapeamento...

a. Matriz de transformação do modelo

- Como vimos em aulas anteriores, podemos aplicar diversas **transformações geométricas** em um objeto através do uso de matrizes:
  - Por exemplo, se desejamos **rotacionar** um objeto, nós multiplicamos a matriz de rotação por todas as coordenadas dos vértices deste objeto;
- A ordem em que essas transformações são aplicadas é essencial:
  - **Transladar** um objeto e em seguida **rotacionar** gera um **resultado diferente** de rotacioná-lo em seguida transladá-lo.
- Vimos também que podemos combinar uma série de transformações em um única matriz, esta contendo todas as transformações na ordem que desejamos aplicar.
  - Esta matriz é chamada de **matriz de transformação do modelo**.

a. Matriz de transformação do modelo

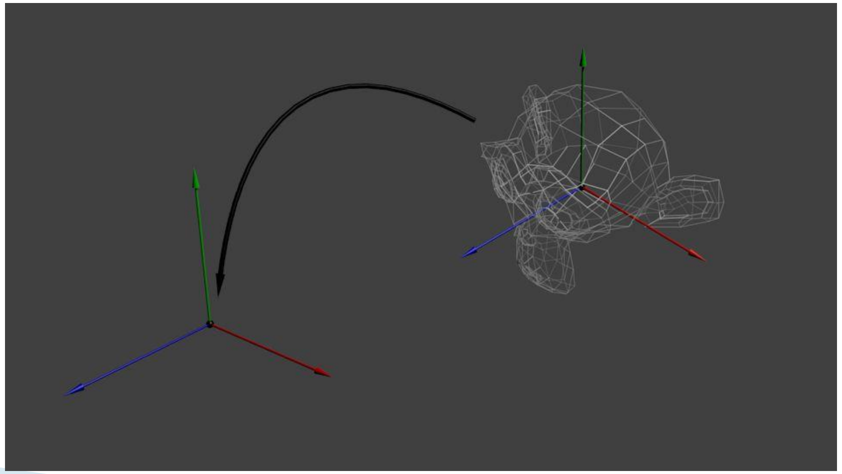


Coordenadas do modelo, em seu próprio “mini-universo”

Matriz de transformação do modelo, geralmente no formato:

M = S*R*T

S: matriz de transformação de escala
R: matriz de transformação de rotação
T: matriz de transformação de translação

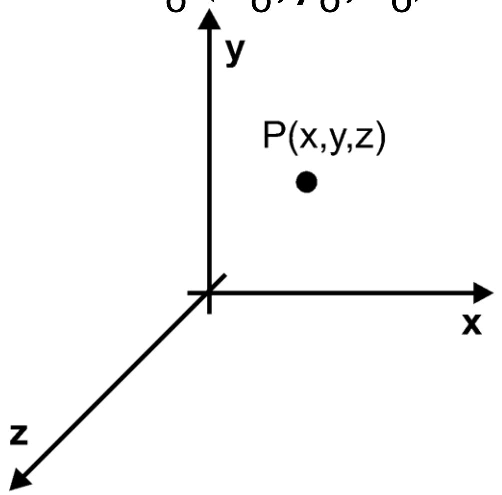


Coordenadas do objeto no universo, após serem aplicadas transformações geométricas no mesmo a partir da **matriz de transformação do modelo.**

a. Matriz de transformação do modelo

- Um modelo de objeto 3D é um conjunto de pontos, estes possuindo três coordenadas:
  - $P_o(x_o,y_o,z_o)$

- Ao ser aplicada uma **transformação geométrica** sobre um modelo, dá-se origem a um novo objeto, este agora no **universo**, cujas as coordenadas dos vértices $P_u(x_u,y_u,z_u)$ são obtidas através de transformações aplicadas sobre as coordenadas originais $P_o\,(x_o,y_o,z_o)$.



a. Translação

- A translação define a posição de um objeto no universo;

- A transformação de translação T sobre um ponto P_o, a partir do uso de coordenadas homogêneas, é dada por:
  - $P_u = TP_o$
  - $$P_u = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Translação

- Ex: translação de:
  - −3 unidades em x;
  - 2 unidades em y;
  - 4 unidades em z;

$$P_u = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Escala

- A escala altera as dimensões de um objeto no universo;

- A transformação de escala S sobre um ponto P_o, a partir do uso de coordenadas homogêneas, é dada por:

  - $P_u = SP_o$

  - $$P_u = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Escala

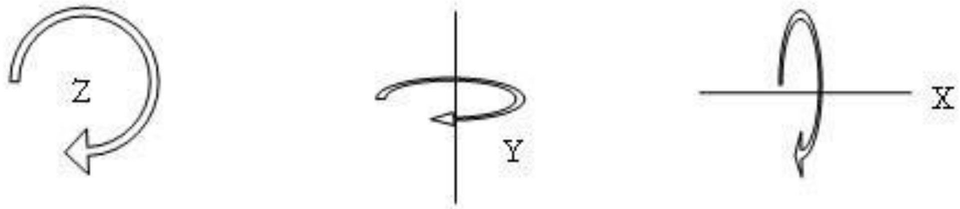
- Ex: escala de:
  - 2 vezes em x;
  - 1.5 vez em y;
  - 0.5 vez em z;

$$P_u = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Rotação

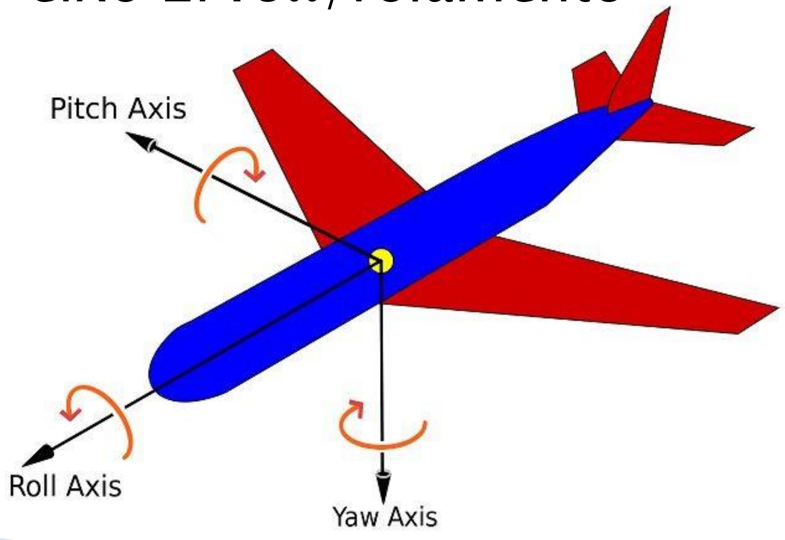
- A rotação de objetos em 3 dimensões é mais complexa que a rotação bidimensional, visto que é necessário definir **sobre qual eixo** a rotação ocorrerá:
  - A rotação bidimensional ocorre sempre sobre o mesmo eixo: o **eixo z**!

- Portanto, teremos **diferentes matrizes de transformação** de **rotação** para cada um dos eixox;



a. Rotação

- É comum utilizarmos termos da aeronáutica/aeroespaciais para denominarmos os 3 tipos de rotação tridimensional:
  - Rotação no eixo x: ***pitch***/passo/arfagem
  - Rotação no eixo y: ***yaw***/guinada
  - Rotação no eixo z: ***roll***/rolamento



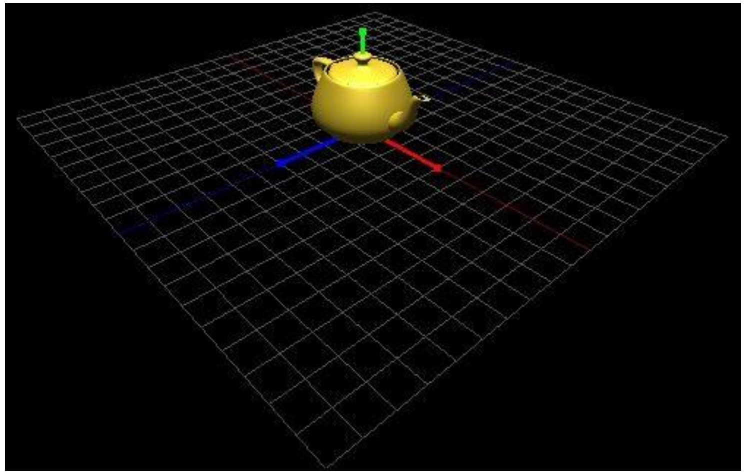
a. Rotação em x/*pitch*

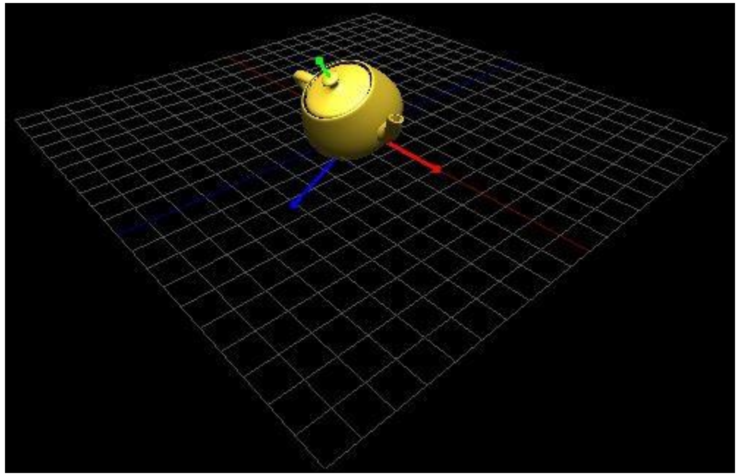
- A matriz de rotação Rx (em torno do eixo x) sobre um ponto P_o, a partir do uso de coordenadas homogêneas, é dada por:
  - $P_u = RxP_o$
    - $$P_u = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Rotação em x/*pitch*

- Ex: rotação em x de 30º

$$P_u = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$





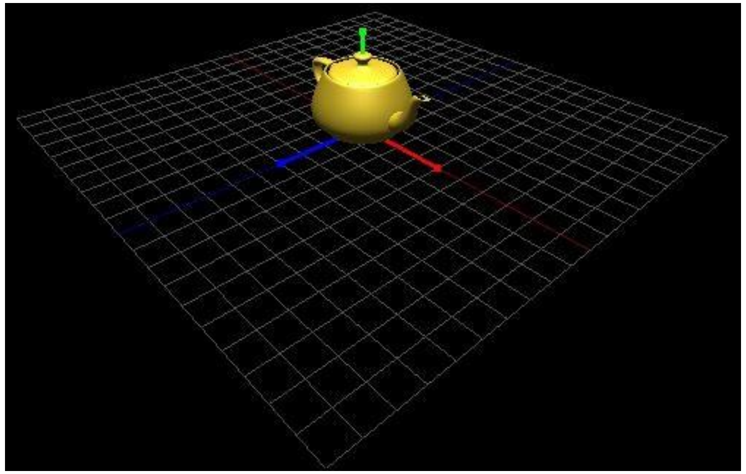
a. Rotação em y/*yaw*

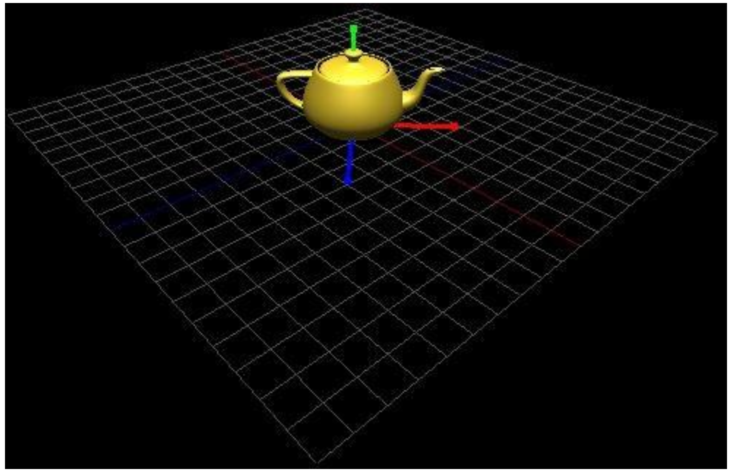
- A matriz de rotação Ry (em torno do eixo y) sobre um ponto P_o, a partir do uso de coordenadas homogêneas, é dada por:
  - $P_u = RyP_o$
    - $$P_u = \begin{bmatrix} cos\theta & 0 & sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sen\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Rotação em y/*yaw*

- Ex: rotação em y de 45º

$$P_u = \begin{bmatrix} \cos(45^o) & 0 & sen(45^o) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(45^o) & 0 & \cos(45^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$





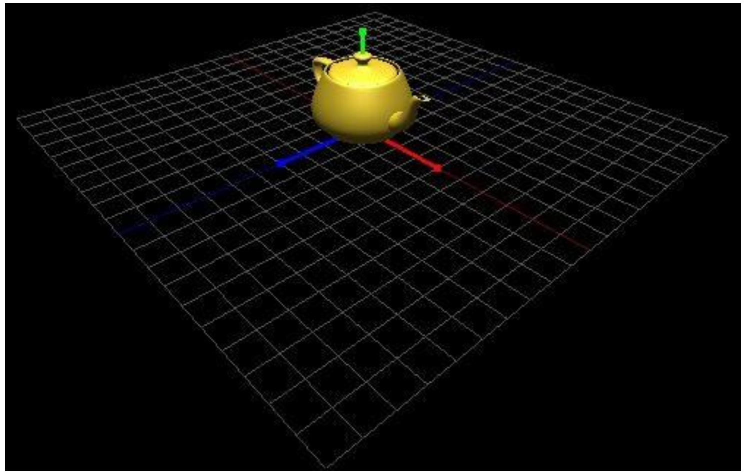
a. Rotação em z/*roll*

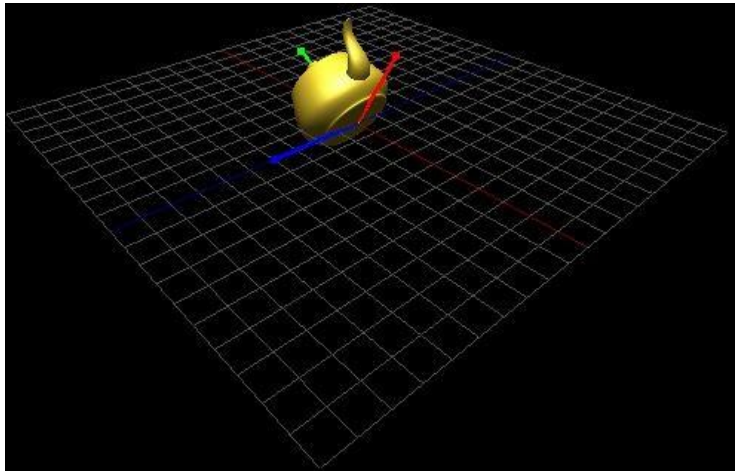
- A matriz de rotação **Rz** (em torno do eixo z) sobre um ponto P_o, a partir do uso de coordenadas homogêneas, é dada por:
  - $P_u = RzP_o$
    - $$P_u = \begin{bmatrix} cos\theta & -sen\theta & 0 & 0 \\ sen\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

a. Rotação em z/*roll*

- Ex: rotação em z de 60º

$$P_u = \begin{bmatrix} \cos(60^o) & -sen(60^o) & 0 & 0 \\ sen(60^o) & \cos(60^o) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$





a.Combinações

- Se desejarmos aplicar **várias transformações** sobre um mesmo objeto, podemos realizar a multiplicação de cada matriz de transformação sobre os pontos do objeto uma por uma, ou, preferencialmente, **combinar todas as transformações em uma única matriz.**

- É importante lembrar que a ordem em que as transformações são aplicadas importa!

a. Combinações

- Exemplo: aplicar, nesta ordem, as seguintes transformações sobre um objeto:
  - Translação de (–3,2,4);
  - Rotação em x de 30º;

- A matriz de transformação resultante da combinação das duas transformações acima é dada por:
  - $M_T = R_x T$
  - $M_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

a. Combinações

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.87 & -0.50 & 0 \\ 0 & 0.50 & 0.87 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{M_T = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 0.87 & -0.50 & -0.27 \\ 0 & 0.50 & 0.87 & 4.46 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Essa matriz representa uma transformação de translação de
(-2,3,4) seguida de uma rotação de 30° em x sobre um ponto

a. Combinações

- Exemplo: aplicar, nesta ordem, as seguintes transformações sobre um objeto:
  - Rotação em x de 30º;
  - Translação de (-3,2,4);

- A matriz de transformação resultante da combinação das duas transformações acima é dada por:
  - $M_T = TRx$
  - $M_T = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^{o}) & -sen(30^{o}) & 0 \\ 0 & sen(30^{o}) & \cos(30^{o}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

a. Combinações

$$M_T = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.87 & -0.50 & 0 \\ 0 & 0.50 & 0.87 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{M_T = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 0.87 & -0.50 & 2 \\ 0 & 0.50 & 0.87 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Essa matriz representa uma transformação de rotação de 30° em x seguida de uma translação de (-2,3,4)

a. Combinações

- Exemplo: aplicar, nesta ordem, as seguintes transformações sobre um objeto:
  - Translação de (–3,2,4);
  - Rotação em x de 30º;
  - Rotação em y de 45º;

- A matriz de transformação resultante da combinação das duas transformações acima é dada por:
  - $M_T = RyRxT$
  - $M_T = \begin{bmatrix} \cos(45^o) & 0 & sen(45^o) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(45^o) & 0 & \cos(45^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

a. Combinações

$$M_T = \begin{bmatrix} \cos(45^o) & 0 & sen(45^o) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(45^o) & 0 & \cos(45^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(30^o) & -sen(30^o) & 0 \\ 0 & sen(30^o) & \cos(30^o) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_T = \begin{bmatrix} 0.71 & 0 & 0.71 & 0 \\ 0 & 1 & 0 & 0 \\ -0.71 & 0 & 0.71 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.87 & -0.50 & 0 \\ 0 & 0.50 & 0.87 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Começamos sempre multiplicando as matrizes da direita para esquerda!

$$M_T =* \begin{bmatrix} 0.71 & 0 & 0.71 & 0 \\ 0 & 1 & 0 & 0 \\ -0.71 & 0 & 0.71 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 0.87 & -0.50 & -0.27 \\ 0 & 0.50 & 0.87 & 4.46 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

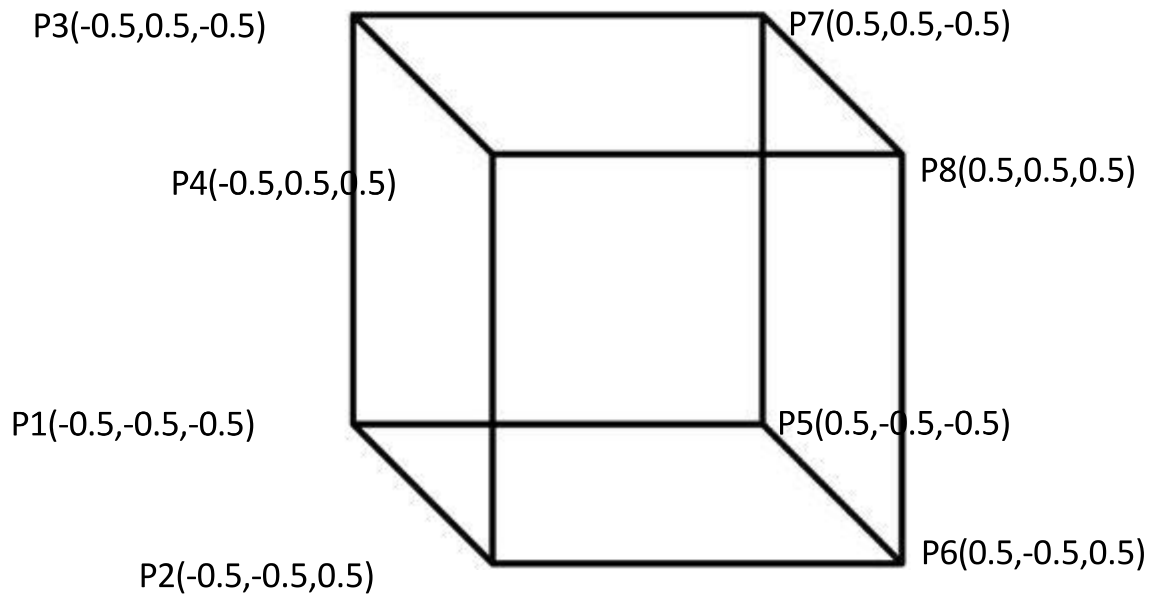
a. Combinações

$$M_T = \begin{bmatrix} 0.71 & 0 & 0.71 & 0 \\ 0 & 1 & 0 & 0 \\ -0.71 & 0 & 0.71 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 0.87 & -0.50 & -0.27 \\ 0 & 0.50 & 0.87 & 4.46 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_T = \begin{bmatrix} 0.71 & 0.35 & 0.61 & 1.04 \\ 0 & 0.87 & -0.50 & -0.27 \\ -0.71 & 0.35 & 0.61 & 5.28 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Essa matriz representa uma transformação de translação de
(-2,3,4) seguida de uma rotação de 30° em x, seguida de uma rotação de 45° em y sobre um ponto

1) Exercício de Modelagem do objeto



1) Modelagem do objeto

Vamos modelar o nosso ‘cubo’ utilizando a linguagem python:

a) Passo 1 definir as coordenadas de cada ponto

```
# Definição dos pontos que definem os vértices do cubo
P1 = (-0.5, -0.5, -0.5)
P2 = (-0.5, -0.5, 0.5)
P3 = (-0.5, 0.5, -0.5)
P4 = (-0.5, 0.5, 0.5)
P5 = (0.5, -0.5, -0.5)
P6 = (0.5, -0.5, 0.5)
P7 = (0.5, 0.5, -0.5)
P8 = (0.5, 0.5, 0.5)
```

b) Definição das arestas do cubo

```
# Definição das arestas do cubo
arestas = [(P1, P2), (P2, P4), (P4, P3), (P3, P1),
           (P5, P6), (P6, P8), (P8, P7), (P7, P5),
           (P1, P5), (P2, P6), (P3, P7), (P4, P8)]
```

1) Modelagem do objeto

c) Utilizar a biblioteca matplotlib para plotar o nosso gráfico com auxílio de algumas ferramentas que ela oferece

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


# aqui parte do código dos pontos e arestas..............................




# Plotando o cubo
fig = plt.figure()  # Criando uma figura
ax = fig.add_subplot(projection='3d')  # Adicionando um subplot tridimensional à figura

# Plotando as arestas do cubo
for aresta in arestas:
    ponto1 = aresta[0]  # Obtendo as coordenadas do primeiro ponto da aresta (x,y,z)
    ponto2 = aresta[1]  # Obtendo as coordenadas do segundo ponto da aresta (x,y,z)
    # Plotando uma linha entre os dois pontos para representar a aresta
    ax.plot([ponto1[0], ponto2[0]], [ponto1[1], ponto2[1]], [ponto1[2], ponto2[2]], 'b’)
    #Coordenadas 3D (x,y,z)
```

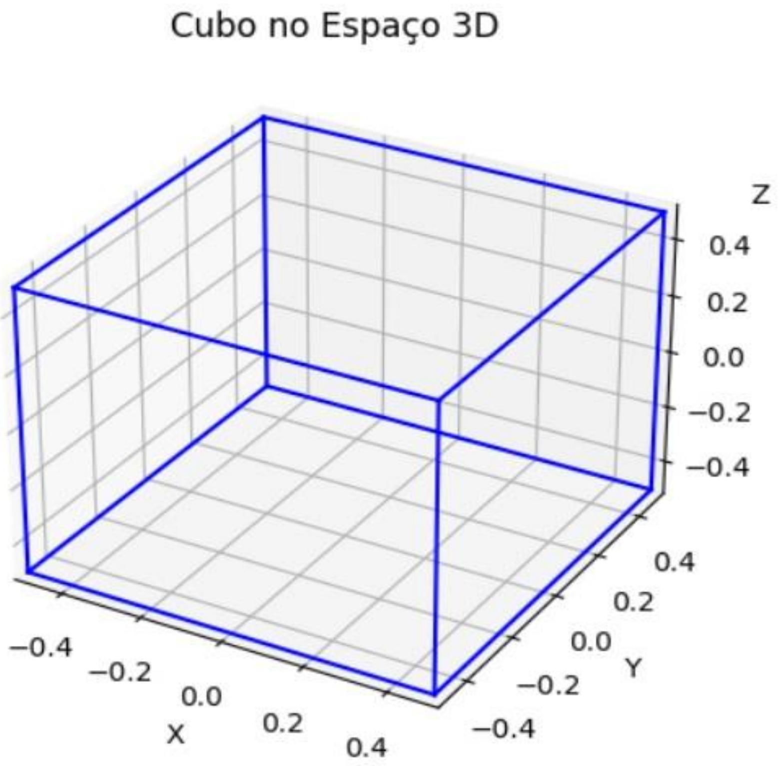
1) Modelagem do objeto

```
# Configurações do gráfico 3D
ax.set_xlabel('X')  # Configurando o rótulo do eixo x
ax.set_ylabel('Y')  # Configurando o rótulo do eixo y
ax.set_zlabel('Z')  # Configurando o rótulo do eixo z
ax.set_title('Cubo no Espaço 3D')  # Configurando o título do gráfico


ax.set_xlim(-0.5, 0.5)  # Limites do eixo X
ax.set_ylim(-0.5, 0.5)  # Limites do eixo Y
ax.set_zlim(-0.5, 0.5)  # Limites do eixo Z
# Adicionando manualmente uma legenda para o eixo Z
ax.text(0.7, 0.5, 0.6, 'Z', color='black')  # Adicionando o texto 'Z' na posição
desejada

# Mostrando o gráfico
plt.show()
```

1) Modelagem do objeto

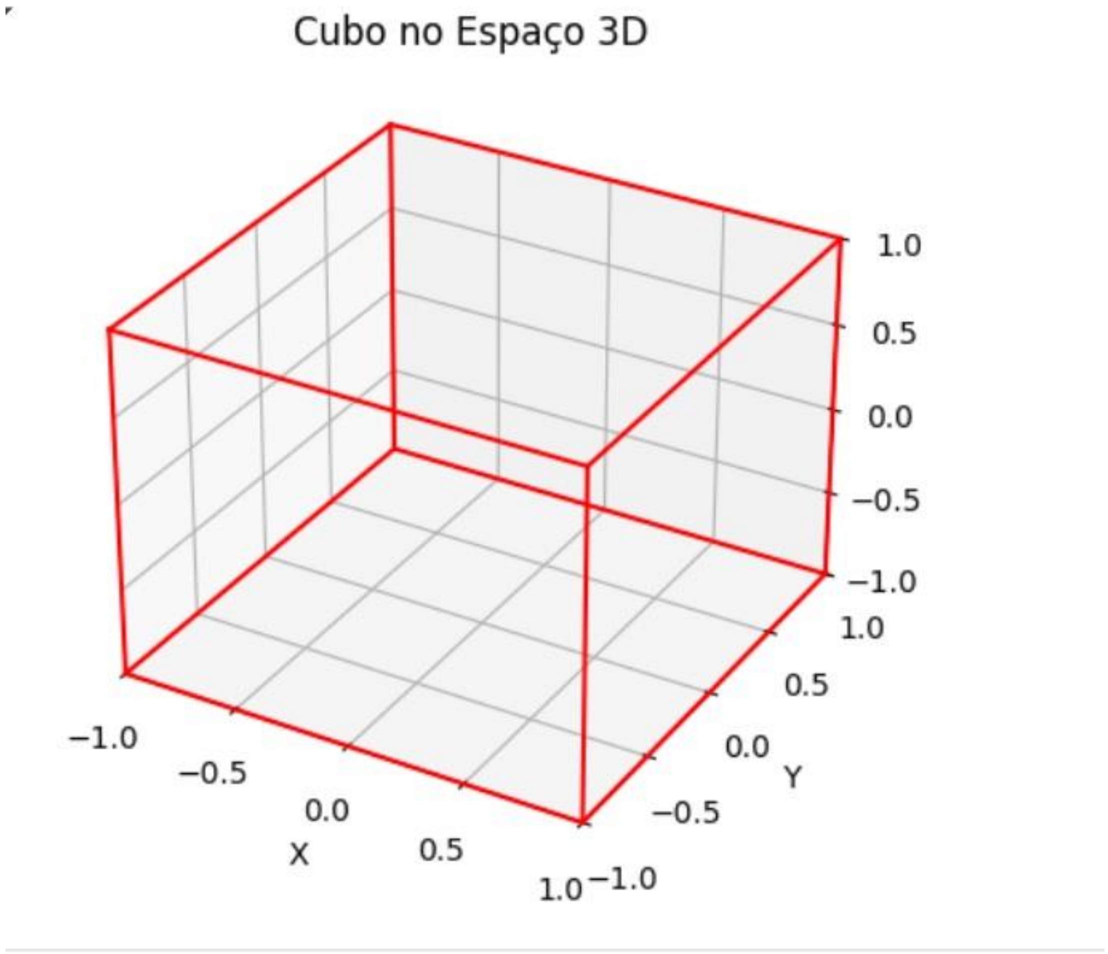
Resultado:

Exercício de aula: Após gerar a imagem idêntica deste cubo, altere as coordenadas dos pontos e gere outro cubo maior, ou seja,
Resultado = pontos originais * 2, e as linhas ou arestas na cor ‘vermelha’



1) Modelagem do objeto

Resultado: (colocar o link do github junto com os outros exercícios de aula na atividade da aula de hoje)



Referências e material de apoio

Material do Professor Guilherme Chagas Kurtz, 2023.

GOMES, Jonas; VELHO, Luiz. Computação gráfica. Rio de Janeiro: Impa, 1998.
HEARN, Donald; Baker, M. Pauline. Computer grafhics: C version. London: Prentice Hall, 1997.
HETEM JUNIOR, Annibal. Computação gráfica. Rio de Janeiro, RJ: LTC, 2006. 161 p. (Coleção Fundamentos de
Informática).
HILL Jr, Francis S. Computer graphics using open GL. New Jersey: Prentice Hall, 2001.
WATT, Alan. 3D computer graphics. Harlow: Addison-Wesley, 2000
Material Prof. Guilherme Chagas Kurtz, 2023.

Thank you for your attention!!

Email: andre.flores@ufn.edu.br

Santa Maria – RS 2025

Computação Gráfica – Ciência da Computação– Pipeline da Visualização 3D

Aula 05
Matriz de Visualização

Professor: André Flores dos Santos

Santa Maria – RS 2025

b. Matriz de visualização

- A matriz de visualização controla o modo como observamos uma cena a partir de uma câmera virtual;

- Esta matriz simula o “movimento” de uma câmera:
  - Mas que na verdade é o mundo que se move ao redor dela;

  - Portanto, como já se dizia no seriado Futurama:

    - “*The engines don't move the ship at all. The ship stays where it is and the engines move the universe around it.*”

    - “Os motores não movem o navio. O navio permanece onde está e os motores movem o universo ao seu redor”

b. Matriz de visualização

- A matriz de visualização geralmente é gerada a partir de uma combinação de:
  - Uma **matriz de translação**, que posiciona a câmera no universo;
  - Uma **matriz de rotação**, que orienta a câmera e "aponta" para onde ela está olhando.

- Ex: posicionar a câmera em (-3,2,4) e rotacioná-la 45° em torno de y:

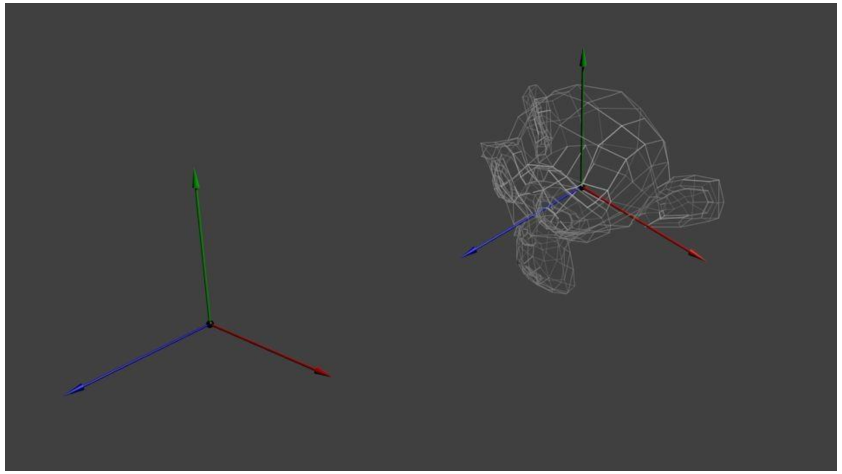
$$M_v = \begin{bmatrix} cos(-45) & 0 & sen(-45) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(-45) & 0 & cos(-45) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_v = \begin{bmatrix} 0.71 & 0 & -0.71 & 4.95 \\ 0 & 1 & 0 & -2 \\ 0.71 & 0 & 0.71 & -0.71 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Os valores de **translação** e o **ângulo de rotação** são **multiplicados por -1,** pois estamos posicionando o universo em relação a câmera, e não a câmera no mundo!

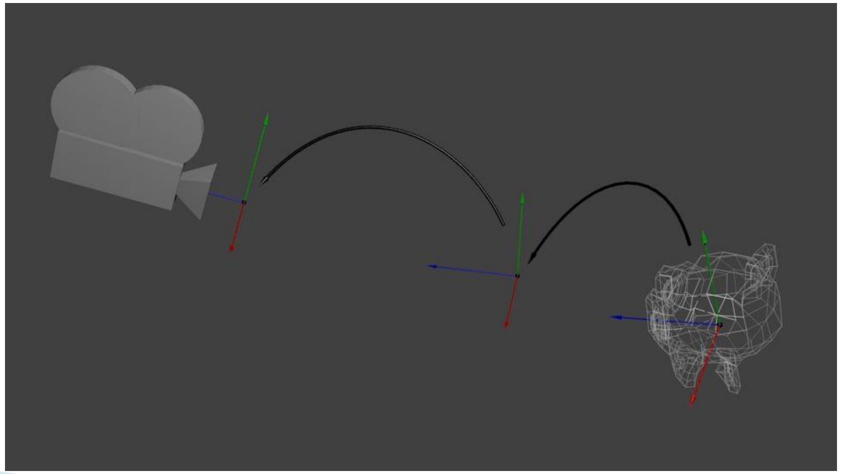
Semelhante como uma imagem pode parecer invertida no espelho e precisamos corrigir mentalmente.

b. Matriz de visualização



Coordenadas do objeto no universo (todos os vértices do objeto posicionados em relação ao **centro do universo**)

Matriz de visualização, que posiciona o mundo em relação a câmera



Coordenadas do objeto no espaço da câmera (todos os vértices do objeto posicionados em relação a **câmera**)

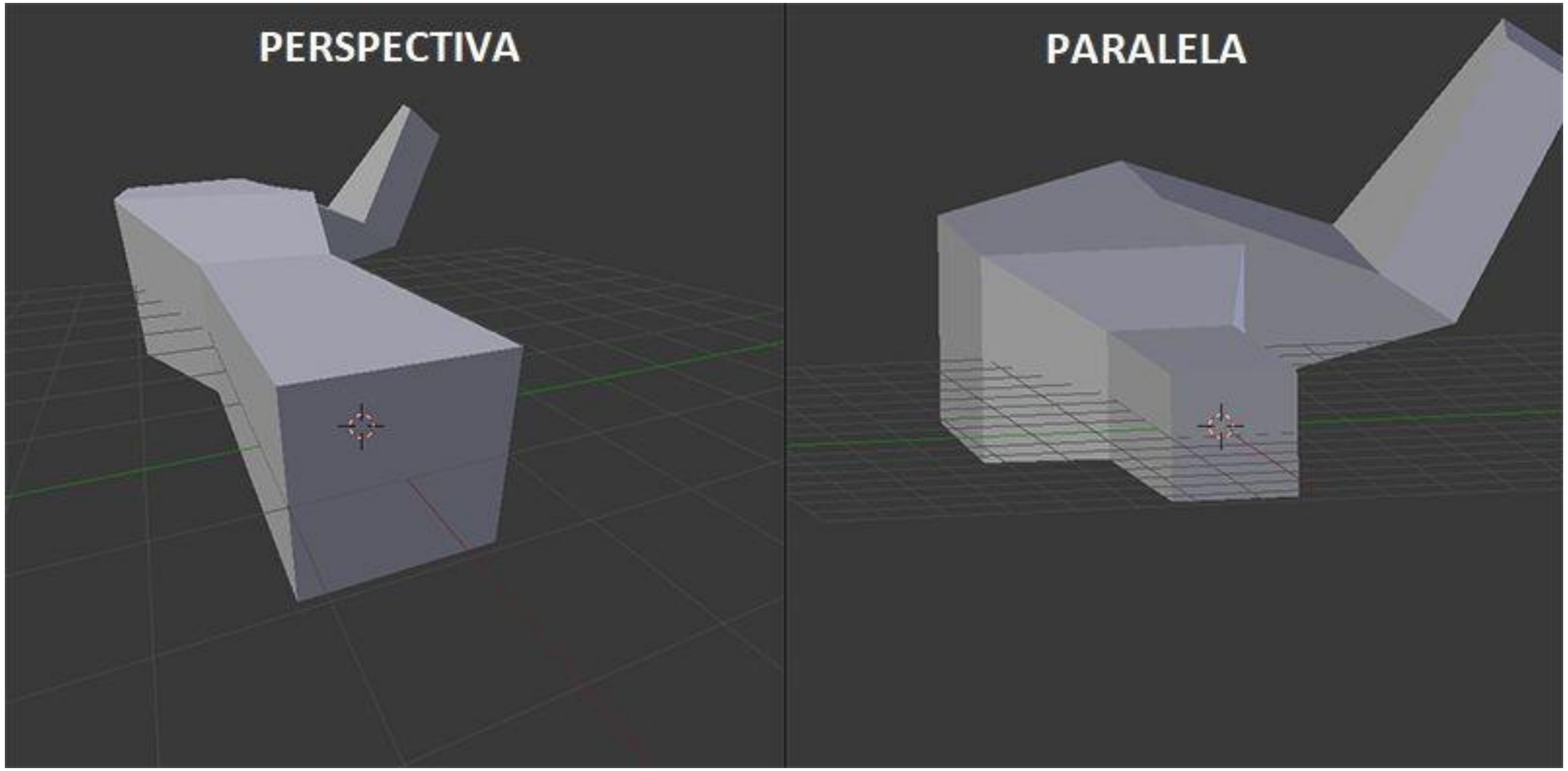
c. Matriz de projeção

- A matriz de projeção define **como os objetos no universo 3D serão projetados em um plano 2D**, fazendo a conversão de seus vértices através do uso de coordenadas homogêneas;

- O modo como essa matriz é construída depende do **tipo de projeção** que desejamos utilizar:

  - Projeção Paralela
  - Projeção Perspectiva.

c. Matriz de projeção

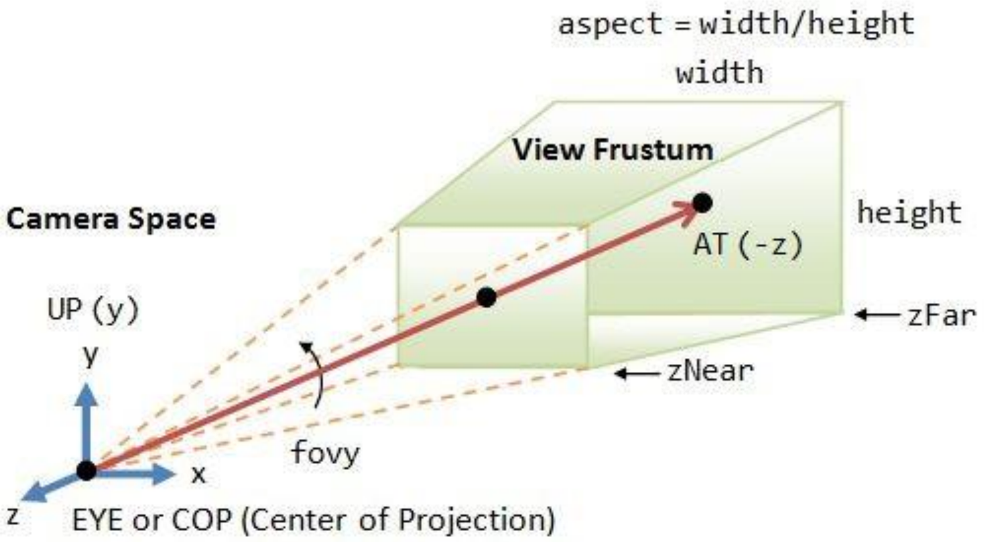


c. Matriz de projeção

- Projeção Perspectiva
  - Na projeção perspectiva, a **câmera** possui um campo de visão limitado, denominado *view frustum*, que é especificado por 4 parâmetros:
    - fovy: Especifica o campo de visão, em graus, na direção vertical;
    - aspect: Determina a razão da largura pela altura da área de visualização;
    - zNear: Distância do observador até o plano frontal deve ser um valor positivo maior que zero
    - zFar: Distância do observador até o plano traseiro deve ser um valor positivo maior que zero

c. Matriz de projeção

- Projeção Perspectiva



Perspective Projection: The camera's view frustum is specified via 4 view parameters: fovy, aspect, zNear and zFar.

c. Matriz de projeção

- Projeção Perspectiva
  - Existem vários modos de se definir e construir a **matriz de projeção perspectiva.**

  - A maneira como o OpenGL constrói, com base nos parâmetros citados anteriormente, é da seguinte forma:

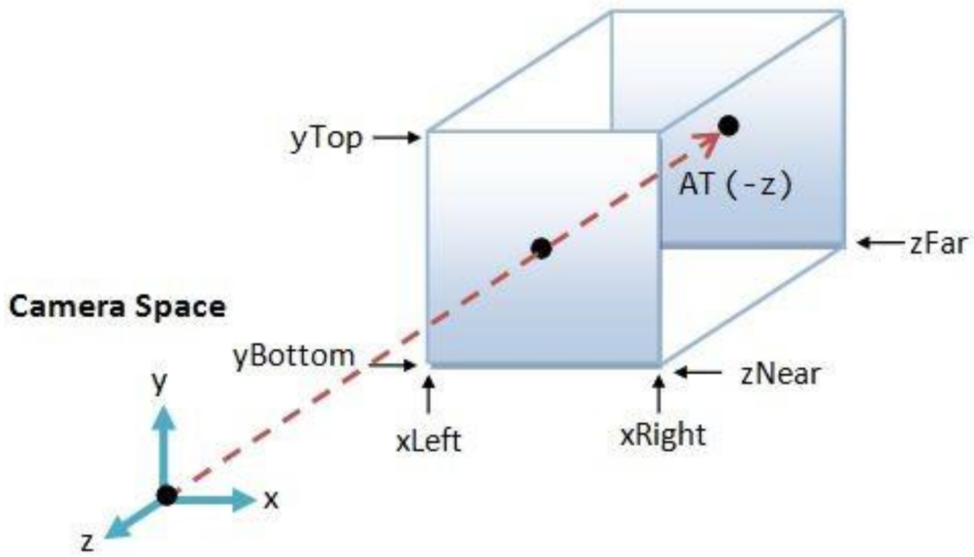
$$M_p = \begin{bmatrix} \frac{1}{tan\left(\frac{fovy}{2}\right)*aspect} & 0 & 0 & 0 \\ 0 & \frac{1}{tan\left(\frac{fovy}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{far+near}{near-far} & \frac{2*far*near}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

c. Matriz de projeção

- Projeção Paralela
  - Apesar da projeção perspectiva ser a mais comumente utilizada, existe outro tipo de projeção chamada de **projeção ortogonal** ou **projeção paralela**;
  - Nesta projeção, a câmera esta posicionada **muito distante do mundo**, como se fosse uma lente de um telescópio;
  - Desta forma, os quatro lados do volume de visualização e os planos frontal e traseiro formam um **paralelepípedo.**
  - Sendo assim, os **parâmetros de visualização** deste tipo de projeção simplesmente definem o volume de recorte:
    - Em x (left e right);
    - Em y (bottom e top);
    - Em z (near e far).

c. Matriz de projeção

- Projeção Paralela



Orthographic Projection: Camera positioned infinitely far away at $z = \infty$

c. Matriz de projeção

- Projeção Paralela
  - Assim com na projeção perspectiva, xistem vários modos de se definir e construir a **matriz de projeção paralela.**

  - A maneira como o OpenGL constrói, com base nos parâmetros citados anteriormente, é da seguinte forma:

$$M_p = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

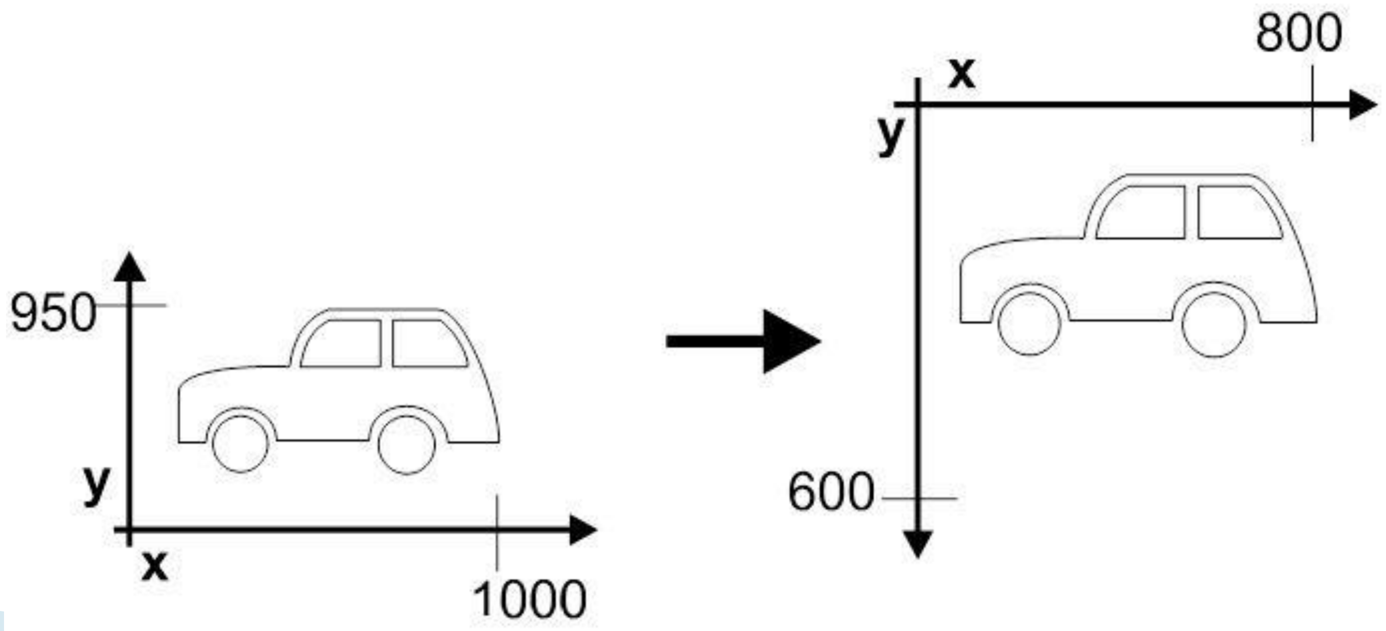
d. Mapeamento (1)

- Após o processo de **Instanciamento** e de **Recorte**, as coordenadas resultantes não são (em geral) compatíveis com as **coordenadas da tela**.

- Normalmente, quando se cria um modelo, as informações gráficas armazenadas (coordenadas, tamanhos, cores, espessuras, etc.) dizem respeito **à aplicação** e não ao **dispositivo** que está sendo usado.

d. Mapeamento (2)

- Para permitir a visualização dos modelos faz-se necessário realizar uma conversão dos valores do modelo para valores compatíveis com as dimensões da tela, é o que se chama de **mapeamento**:

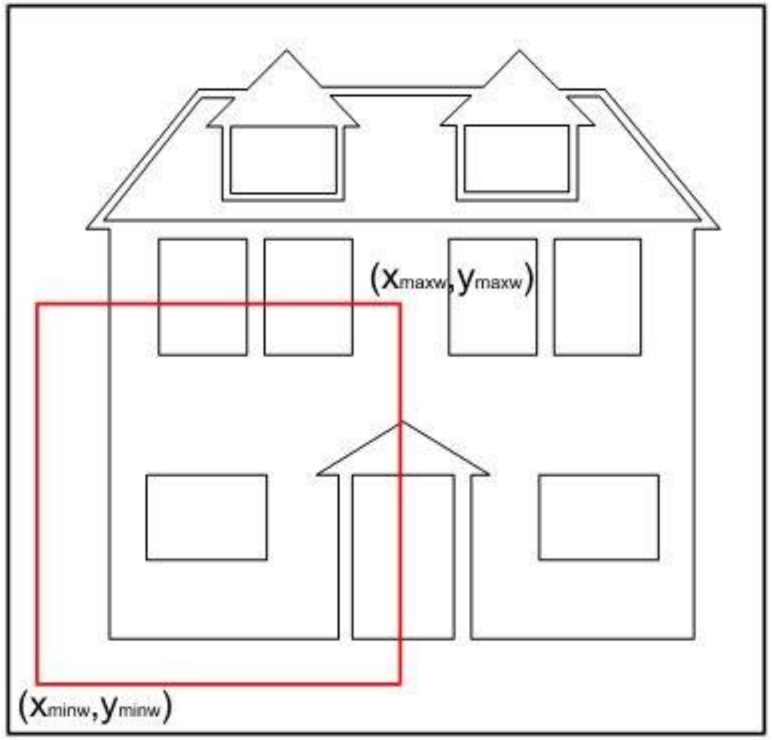


d. Mapeamento (3)

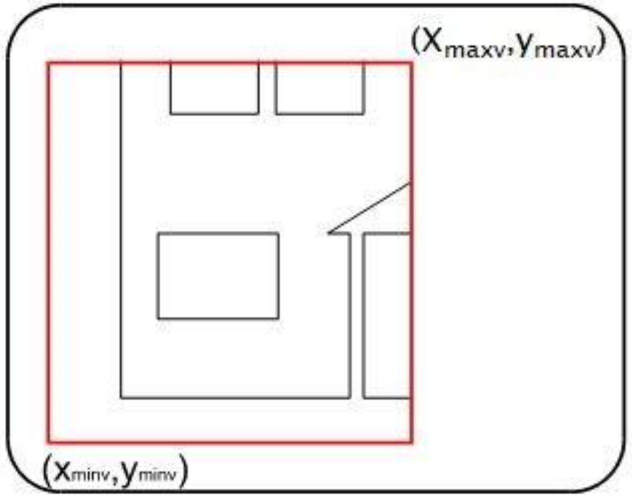
- Window e Viewport:
  - Quando define-se a área de trabalho da aplicação, que vai de (x_{minw}, y_{minw}) a (x_{maxw}, y_{maxw}), está selecionando-se a **WINDOW**, ou seja, a **região do plano cartesiano com a qual se deseja trabalhar**.
  - Quando define-se a **área de exibição**, dentro da tela, que vai de (x_{minv}, y_{minv}) a (x_{maxv}, y_{maxv}), está especificando-se a **VIEWPORT**.

d. Mapeamento (4)

window



viewport



d. Mapeamento (5)

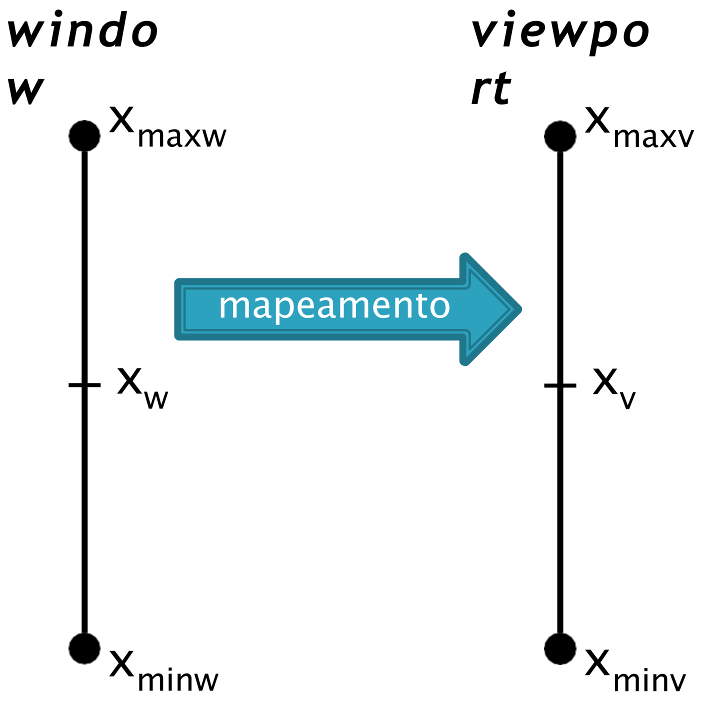
- O mapeamento entre dois Sistemas de Referência é feito por uma **conversão de escala**, para x e para y;

- Tendo um ponto P_w de coordenadas (x_w,y_w) na WINDOW, obtêm-se o ponto P_v (x_v, y_v) na VIEWPORT.

d. Mapeamento (6)

- Para x:

dedução...



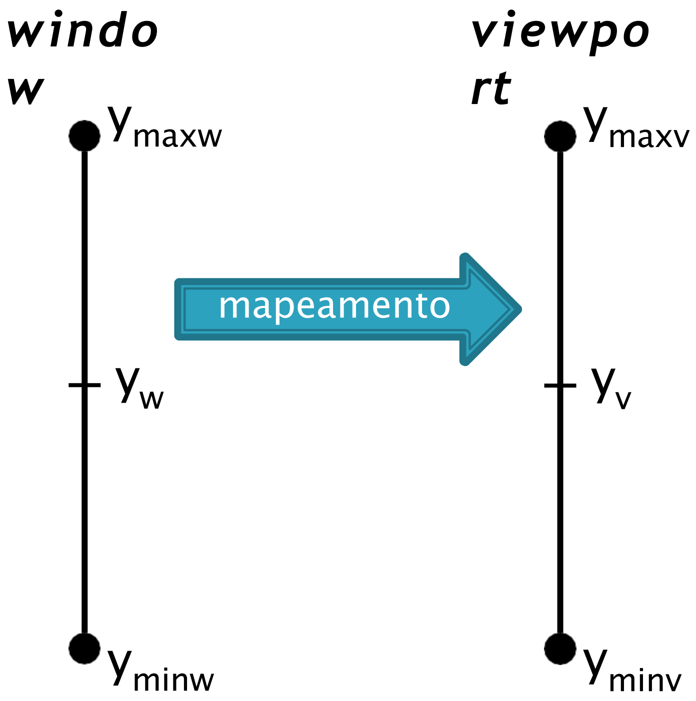
$$\frac{x_w - x_{minw}}{x_{maxw} - x_{minw}} = \frac{x_v - x_{minv}}{x_{maxv} - x_{minv}}$$

$$\frac{(x_w - x_{minw}) * (x_{maxv} - x_{minv})}{x_{maxw} - x_{minw}} = x_v - x_{minv}$$

$$\boxed{x_v = \frac{(x_w - x_{minw}) * (x_{maxv} - x_{minv})}{x_{maxw} - x_{minw}} + x_{minv}}$$

d. Mapeamento (7)

- Para y:



dedução...

$$\frac{y_w - y_{minw}}{y_{maxw} - y_{minw}} = \frac{y_v - y_{minv}}{y_{maxv} - y_{minv}}$$

$$\frac{(y_w - y_{minw})*(y_{maxv} - y_{minv})}{x_{maxw} - x_{minw}} = y_v - y_{minv}$$

$$\boxed{y_v = \frac{(y_w - y_{minw}) * (y_{maxv} - y_{minv})}{y_{maxw} - y_{minw}} + y_{minv}}$$

d. Mapeamento (5)

- Portanto, aplicam-se as seguintes equações:

$$x_v = \frac{(x_w - x_{\min w}) * (x_{\max v} - x_{\min v})}{x_{\max w} - x_{\min w}} + x_{\min v}$$

$$y_v = \frac{(y_w - y_{\min w}) * (y_{\max v} - y_{\min v})}{y_{\max w} - y_{\min w}} + y_{\min v}$$

d. Mapeamento (6)

- Onde:
  - x_{minw} é a coordenada x do canto inferior esquerdo da window
  - y_{minw} é a coordenada y do canto inferior esquerdo da window
  - x_{maxw} é a coordenada x do canto superior direito da window
  - y_{maxw} é a coordenada y do canto superior direito da window
  - x_{minv} é a coordenada x do canto inferior esquerdo da viewport
  - y_{minv} é a coordenada y do canto inferior esquerdo da viewport
  - x_{maxv} é a coordenada x do canto superior direito da viewport
  - y_{maxv} é a coordenada y do canto superior direito da viewport

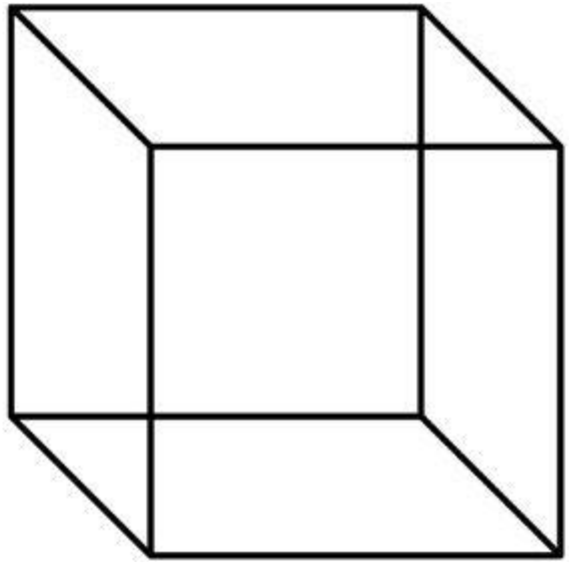
d. Mapeamento (7)

- Quando o SRD tem as coordenadas 0,0 no **canto superior esquerdo** e o valor das coordenadas y aumentam para baixo, o cálculo da coordenada y deve ser realizado com a aplicação da seguinte equação:

$$y_v = \frac{(y_w - y_{\min w}) * (y_{\min v} - y_{\max v})}{y_{\max w} - y_{\min w}} + y_{\max v}$$

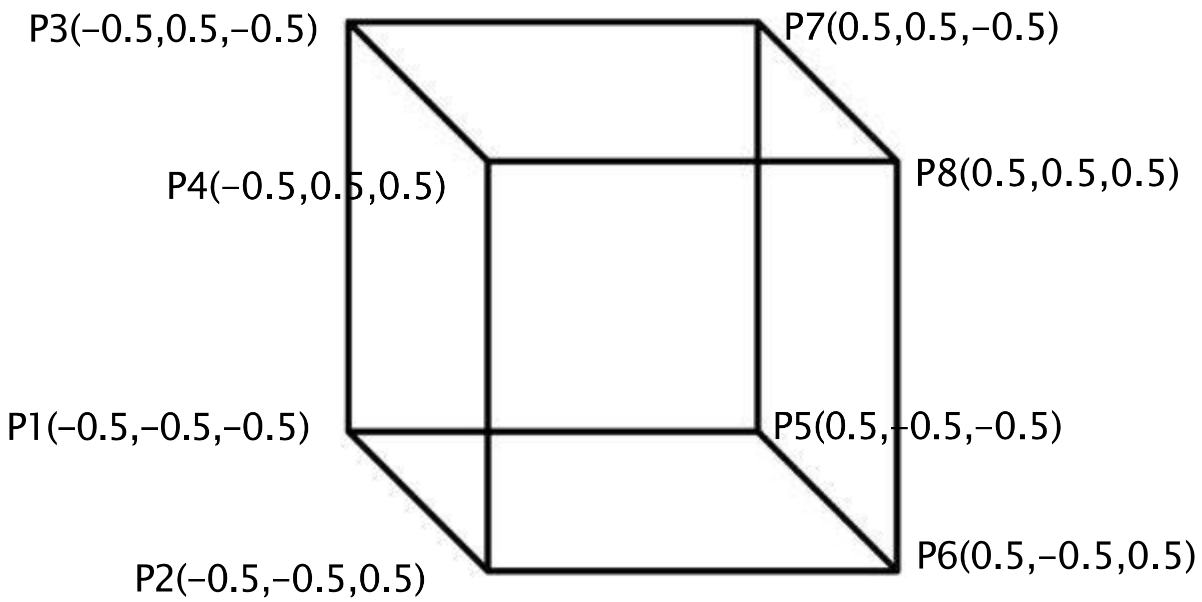
Exemplo completo

- Neste exemplo, vamos realizar o **processo completo de renderização de um cubo**, desde sua definição até o momento em que temos as coordenadas para desenhá-lo na tela.



Exemplo completo

- Passo 1: **Modelagem do objeto**:
  - Definição dos vértices do modelo



Exemplo completo

- Passo 2: **Coordenadas do mundo**:
  - Aplicação das transformações geométricas no objeto;
  - Neste exemplo, vamos rotacionar o cubo 60º em y. Portanto, a matriz de **transformação do modelo** é:

$$M = Ry$$

$$M = \begin{bmatrix} cos(60) & 0 & sen(60) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(60) & 0 & cos(60) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 0.5 & 0 & 0.86 & 0 \\ 0 & 1 & 0 & 0 \\ -0.86 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exemplo completo

- Passo 2: **Coordenadas do mundo**:
  - Com a **matriz de transformação do modelo** calculada, multiplicamos a mesma pelos vértices do modelo, de modo a obtermos os **vértices** do objeto no **universo**:

  - $P1_u = M*P1 = (-0.68,-0.50,0.18)$
  - $P2_u = M*P2 = (0.18,-0.50,0.68)$
  - $P3_u = M*P3 = (-0.68,0.50,0.18)$
  - $P4_u = M*P4 = (0.18,0.50,0.68)$
  - $P5_u = M*P5 = (-0.18,-0.50,-0.68)$
  - $P6_u = M*P6 = (0.68,-0.50,-0.18)$
  - $P7_u = M*P7 = (-0.18,0.50,-0.68)$
  - $P8_u = M*P8 = (0.68,0.50,-0.18)$

Exemplo completo

- Passo 3: **Coordenadas de visualização:**
  - Neste passo, “posicionaremos” a câmera virtual no universo, e então definiremos a **matriz de visualização**;
  - Neste exemplo, vamos “posicionar” a câmera em (0,0,2), sem aplicar nenhuma rotação na mesma.
  - Logo, a matriz de visualização será:
    - $M_v = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Lembre-se, os valores de **translação** são **multiplicados por -1,** pois estamos posicionando o universo em relação a câmera, e não a câmera no universo!

Exemplo completo

- Passo 3: **Coordenadas de visualização**:
  - Com a **matriz de visualização** calculada, multiplicamos a mesma pelos vértices do objeto no universo, calculado anteriormente, de modo a obtermos os **vértices** do objeto em **relação** a **câmera**:

  - $P1_v = M_v * P1_u = (-0.68, -0.50, -1.82)$
  - $P2_v = M_v * P2_u = (0.18, -0.50, -1.32)$
  - $P3_v = M_v * P3_u = (-0.68, 0.50, -1.82)$
  - $P4_v = M_v * P4_u = (0.18, 0.50, -1.32)$
  - $P5_v = M_v * P5_u = (-0.18, -0.50, -2.68)$
  - $P6_v = M_v * P6_u = (0.68, -0.50, -2.18)$
  - $P7_v = M_v * P7_u = (-0.18, 0.50, -2.68)$
  - $P8_v = M_v * P8_u = (0.68, 0.50, -2.18)$

Exemplo completo

- Passo 4: **Coordenadas de projeção**:
  - Neste passo, aplicaremos uma transformação de projeção a partir de uma matriz de projeção;
  - Para este exemplo, utilizaremos uma projeção perspectiva, com:
    - campo de visão de 67º;
    - zNear = 0.1
    - zFar = 100
    - Aspecto = 1
  - A matriz de projeção, calculada de acordo com os parâmetros acima, é:
    - $M_p = \begin{bmatrix} 1.51 & 0 & 0 & 0 \\ 0 & 1.51 & 0 & 0 \\ 0 & 0 & -1 & -0.2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$

Exemplo completo

- Passo 4: **Coordenadas de projeção**:
  - Com a **matriz de projeção** calculada, multiplicamos a mesma pelos vértices do objeto no em relação a câmera, calculado anteriormente, de modo a obtermos os **vértices** do objeto no **plano de projeção 2D**:

  - $P1_p = M_p{*}P1_v = (-1.03, -0.76, 1.62, 1.82)$
  - $P2_p = M_p{*}P2_v = (0.28, -0.76, 1.12, 1.32)$
  - $P3_p = M_p{*}P3_v = (-1.03, 0.76, 1.62, 1.82)$
  - $P4_p = M_p{*}P4_v = (0.28, 0.76, 1.12, 1.32)$
  - $P5_p = M_p{*}P5_v = (-0.28, -0.76, 2.49, 2.68)$
  - $P6_p = M_p{*}P6_v = (1.03, -0.76, 1.99, 2.18)$
  - $P7_p = M_p{*}P7_v = (-0.28, 0.76, 2.49, 2.68)$
  - $P8_p = M_p{*}P8_v = (1.03, 0.76, 1.99, 2.18)$

Lembre-se, ao calcular os vértices no plano de projeção, precisamos considerar também o valor de w, pois iremos, na sequência, dividir os valores de x, y e z por ele

Exemplo completo

- Passo 4: **Coordenadas de projeção**:
  - Dividindo as coordenadas homogêneas por w e descartando z, temos:

  - $P1_p=(-0.57,-0.42)$
  - $P2_p=(0.21,-0.57)$
  - $P3_p=(-0.57,0.42)$
  - $P4_p=(0.21,0.57)$
  - $P5_p=(-0.10,-0.28)$
  - $P6_p=(0.47,-0.35)$
  - $P7_p=(-0.10,0.28)$
  - $P8_p=(0.47,0.35)$

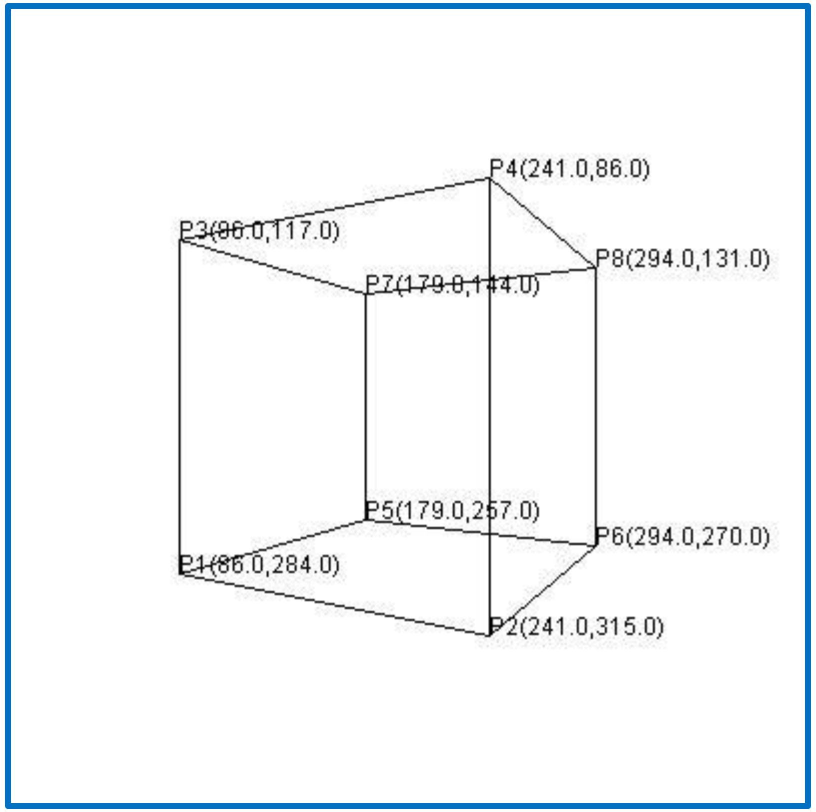
Lembre-se, ao calcular os vértices no plano de projeção, precisamos considerar também o valor de w, pois iremos, na sequência, dividir os valores de x, y por ele!

Exemplo coompleto

- Passo 5: **Coordenadas do dispositivo**:
  - Neste passo, realizamos o **recorte** e **mapeamento** para as coordenadas do dispositivo. Para simplificarmos, **não recortaremos nada.**
  - Para o **mapeamento**, utilizaremos:
    - uma **window** de (-1,-1, 1, 1);
    - uma **viewport** de (0,0,400,400), com y crescendo de cima pra baixo;
  - Os pontos, no dispositivo, ao serem mapeados, ficam:
  - $P1_d$=(86,284)
  - $P2_d$=(241,315)
  - $P3_d$=(86,117)
  - $P4_d$=(241,86)
  - $P5_d$=(179,257)
  - $P6_d$=(294,270)
  - $P7_d$=(179,144)
  - $P8_d$=(294,131)

Exemplo completo

- Resultado da plotagem dos pontos numa janela de 400x400 píxeis:



Referências e material de apoio

Material do Professor Guilherme Chagas Kurtz, 2023.

GOMES, Jonas; VELHO, Luiz. Computação gráfica. Rio de Janeiro: Impa, 1998.
HEARN, Donald; Baker, M. Pauline. Computer grafhics: C version. London: Prentice Hall, 1997.
HETEM JUNIOR, Annibal. Computação gráfica. Rio de Janeiro, RJ: LTC, 2006. 161 p. (Coleção Fundamentos de Informática).
HILL Jr, Francis S. Computer graphics using open GL. New Jersey: Prentice Hall, 2001.
WATT, Alan. 3D computer graphics. Harlow: Addison-Wesley, 2000
Material Prof. Guilherme Chagas Kurtz, 2023.

Thank you for your attention!!

Email: andre.flores@ufn.edu.br

Santa Maria – RS 2025