CCI 36 – Computação Gráfica

OpenGL – Parte 2

Instituto Tecnológico de Aeronáutica

Prof. Carlos Henrique Q. Forster – Sala 121 IEC

forster@ita.br

Luiz Felipe Simões Hoffmann

Tópicos da Aula

- Índices
- Transformações Geométricas
- Sistema de Coordenadas
- Exemplo

Referências

Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane. OpenGL Programming Guide, 8th ed., Pearson Education, 2013.

Richard S. Wright Jr., Nicholas Haemel, Graham Sellers, Benjamin Lipchak. OpenGL Super Bible, 5th ed., Pearson Education, 2011.

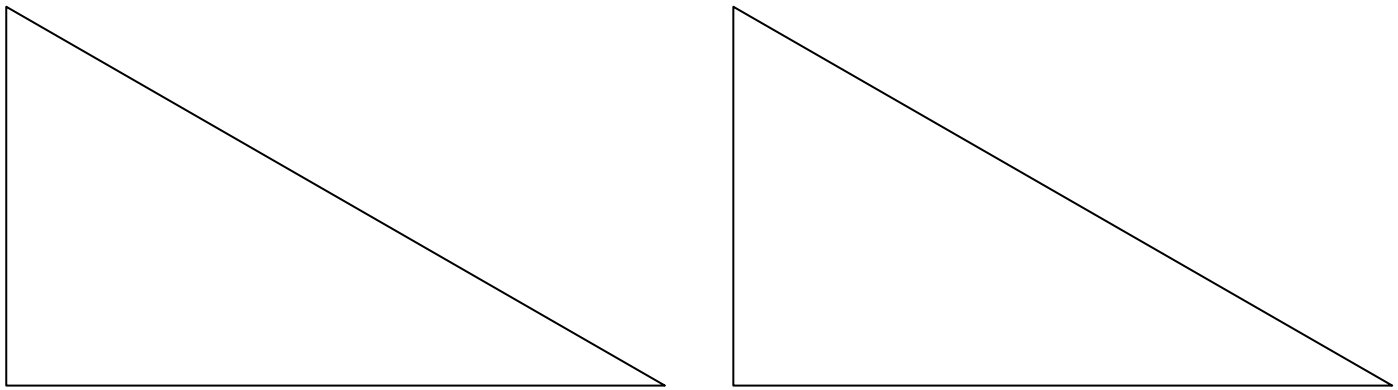
Joey de Vries. Learn OpenGL, Joey de Vries, 2015.

Índices

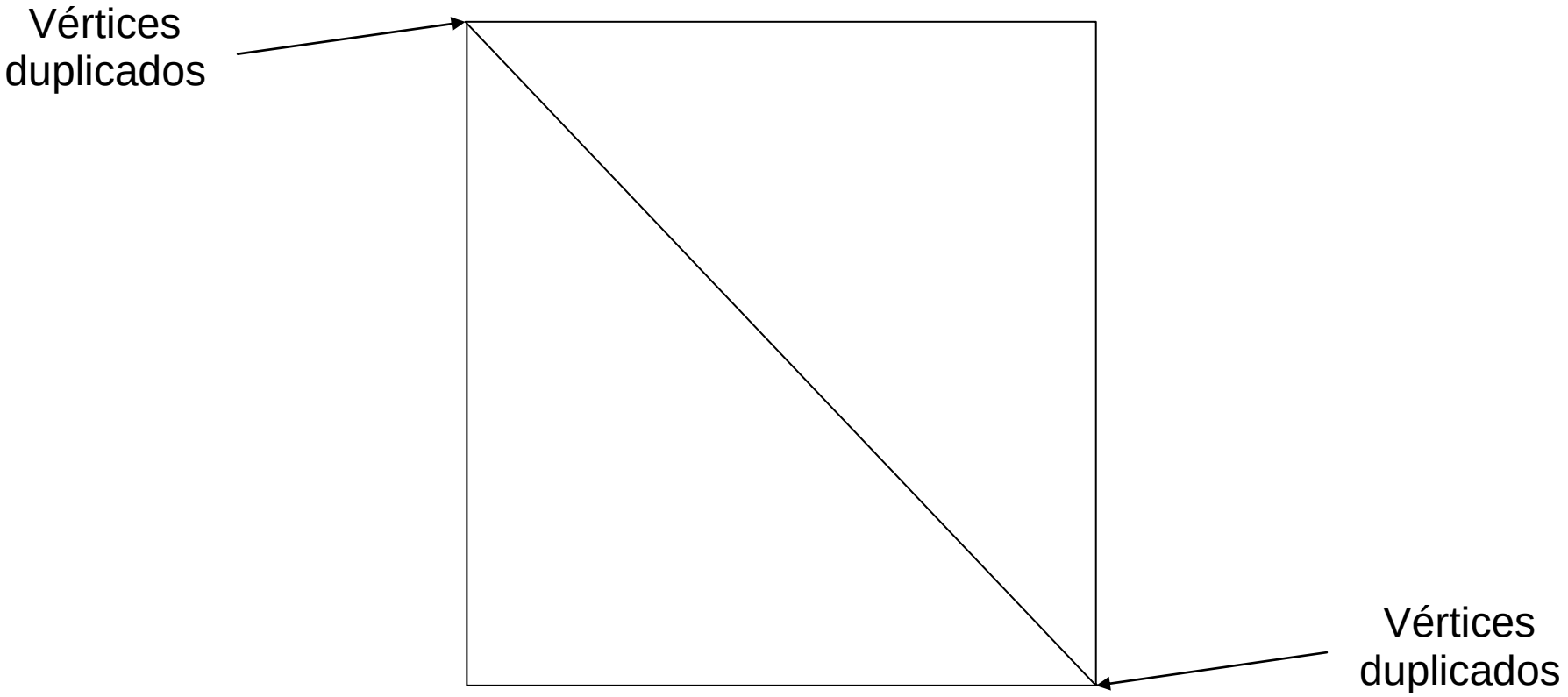
São utilizados pelo OpenGL como meio de se fazer referência aos vértices de primitivas durante a renderização de objetos.

Possibilita a otimização de memória (evita a repetição de vértices).

Exemplo: utilizar índices na construção de um retângulo, a partir de dois triângulos.



Exemplo: Retângulo



Exemplo: Retângulo

Passo 1: definir os vértices e índices.

```
vertices = np.array([ 0.5,  0.5,              # Superior Direito
                      0.5, -0.5,              # Inferior Direito
                     -0.5, -0.5,              # Inferior Esquerdo
                     -0.5,  0.5               # Superior Esquerdo
                    ], dtype=np.float32)

indices = np.array([0, 1, 3,                  # Primeiro Triângulo
                    1, 2, 3                   # Segundo Triângulo
                   ], dtype=np.uint8)
```

Exemplo: Retângulo

Passo 2: alocar memória na GPU.

```
# Cria um vertex buffer object
vbo = glGenBuffers(1)
glBindBuffer(GL_ARRAY_BUFFER, vbo)
glBufferData(GL_ARRAY_BUFFER, vertices.size * vertices.itemsize,
             vertices, GL_STATIC_DRAW)

# Cria um element buffer object
ebo = glGenBuffers(1)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ebo)
glBufferData(GL_ELEMENT_ARRAY_BUFFER,
             indices.size * indices.itemsize, indices,
             GL_STATIC_DRAW)
```

Exemplo: Retângulo

Passo 3: vincular os vértices.

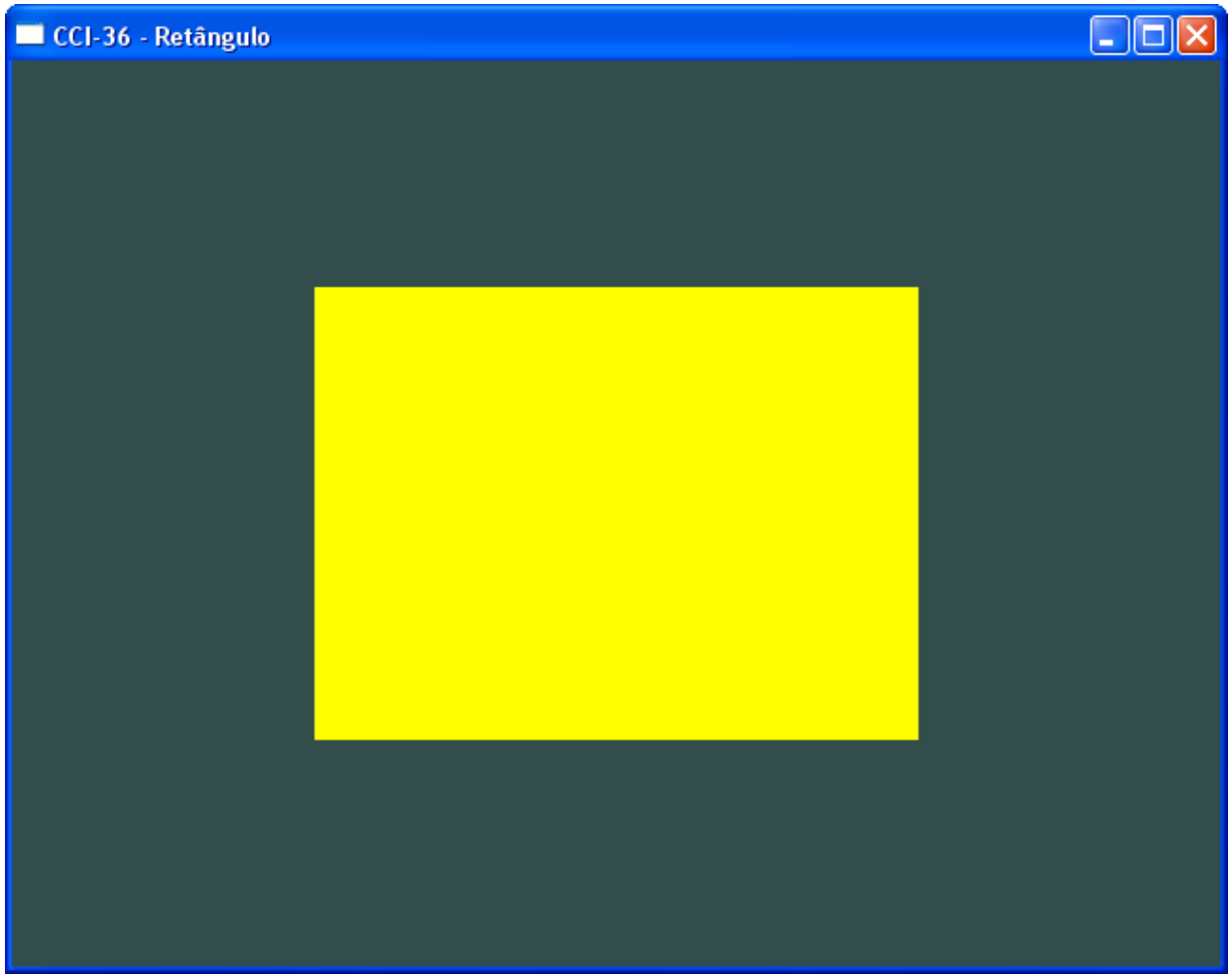
```
# Posição
glVertexAttribPointer(0,                                # Índice do vbo
                      2,                                # Tamanho (x,y)
                      GL_FLOAT,                         # Tipo
                      GL_FALSE,                         # Normalizar
                      2 * vertices.itemsize,            # Stride
                      None                              # Offset
                     )
glEnableVertexAttribArray(0)
```

Exemplo: Retângulo

Passo 4: desenhar o retângulo.

```
glDrawElements(GL_TRIANGLES,                    # Tipo de primitiva
               6,                               # Número de índices
               GL_UNSIGNED_BYTE,                # Tipo dos índices
               None                             # Offset/Ponteiro
              )
```

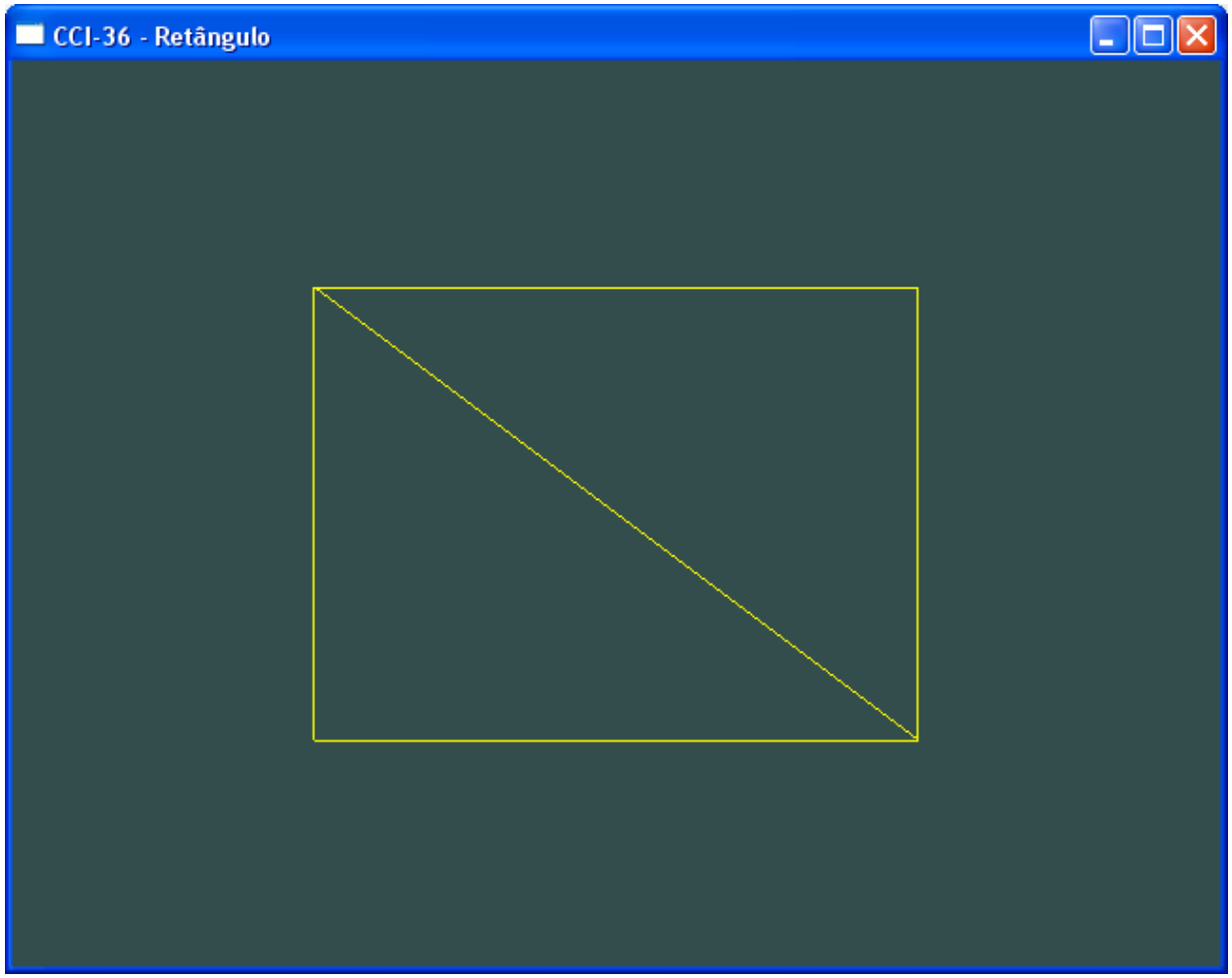
Exemplo: Retângulo



Exemplo: Retângulo

Wireframe

glPolygonMode(
GL_FRONT_AND_BACK,
GL_LINE)



Transformações Geométricas

Uma transformação é uma função que mapeia um ponto (ou vetor) a outro ponto (ou vetor).

É uma forma de mover os pontos que descrevem um ou mais objetos geométricos, para novos locais (atribuir esses objetos a um sistema de coordenadas).

No OpenGL, o *vertex shader* é responsável pela execução de operações (transformações geométricas) nos vértices de objetos.

Principais transformações geométricas: translação, escalamento e rotação.

Coordenadas Homogêneas

As coordenadas homogêneas possibilitam estender o espaço $\mathbb{R}^n$ para $\mathbb{R}^{n+1}$, por meio da adição de uma nova coordenada (*w*).

$$P = (x, y, z, w)$$

Por exemplo, o ponto P = (8, 2, 10) pode ser expresso como:

- (8, 2, 10, 1), *w* = 1.
- (16, 4, 20, 2), w = 2...

Para voltar ao espaço inicial, basta dividir cada termo pela coordenada *w* e descartá-la.

$$P = (x/w, y/w, z/w)$$

Coordenadas Homogêneas

Assume-se $w \geq 0$. Ponto normal, $w > 0$, e ponto no infinito, $w = 0$ (utilizado por vetores para impedir translação).

Qual é a matriz?

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x+3 \\ y \\ z \end{bmatrix}$$

Uma possível solução:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 3/y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x+3 \\ y \\ z \end{bmatrix}$$

Coordenadas Homogêneas

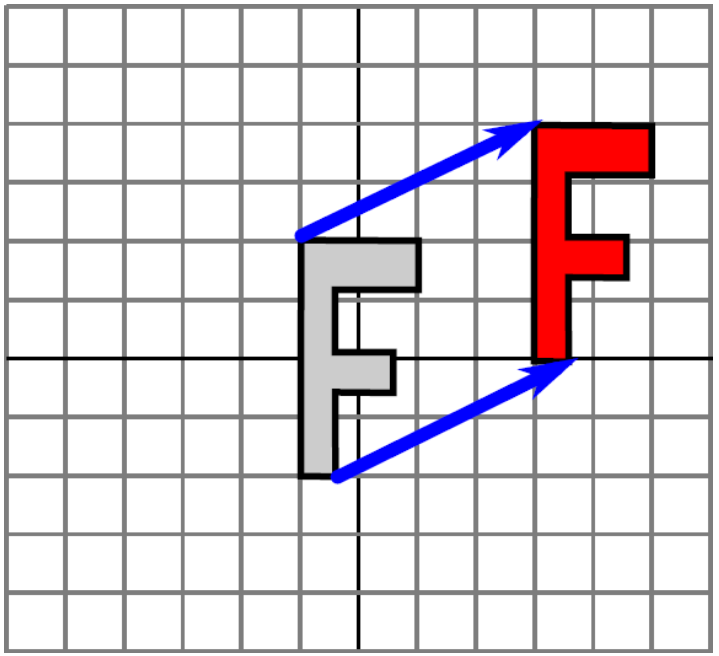
Porém, a matriz de transformação deve ser constante (não pode ser uma função de *x*, *y* ou *z*). Assim tem-se a seguinte matriz homogênea:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+3 \\ y \\ z \\ 1 \end{bmatrix}$$

Translação

É uma operação que desloca pontos, por uma distância fixa, em uma determinada direção e sentido.

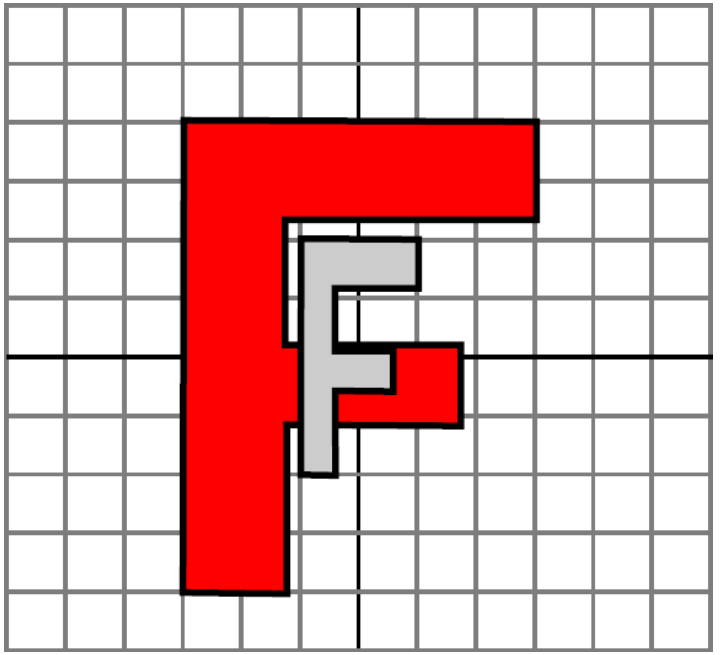
$$P' = TP = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Escalamento (Escala)

É uma operação que torna um objeto maior ou menor.

$$P' = SP = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



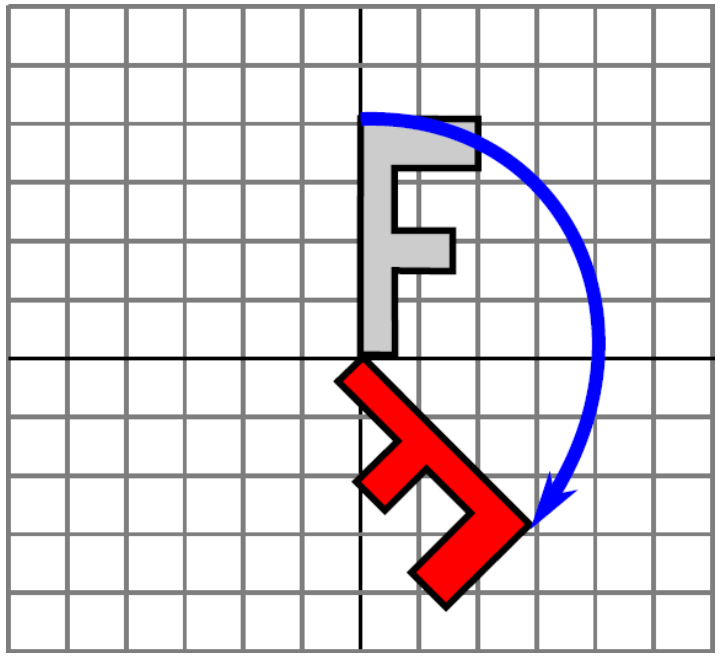
Rotação

É uma operação que rotaciona (gira) um ponto em torno da origem, por um ângulo.

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

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

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



Sistema de Coordenadas

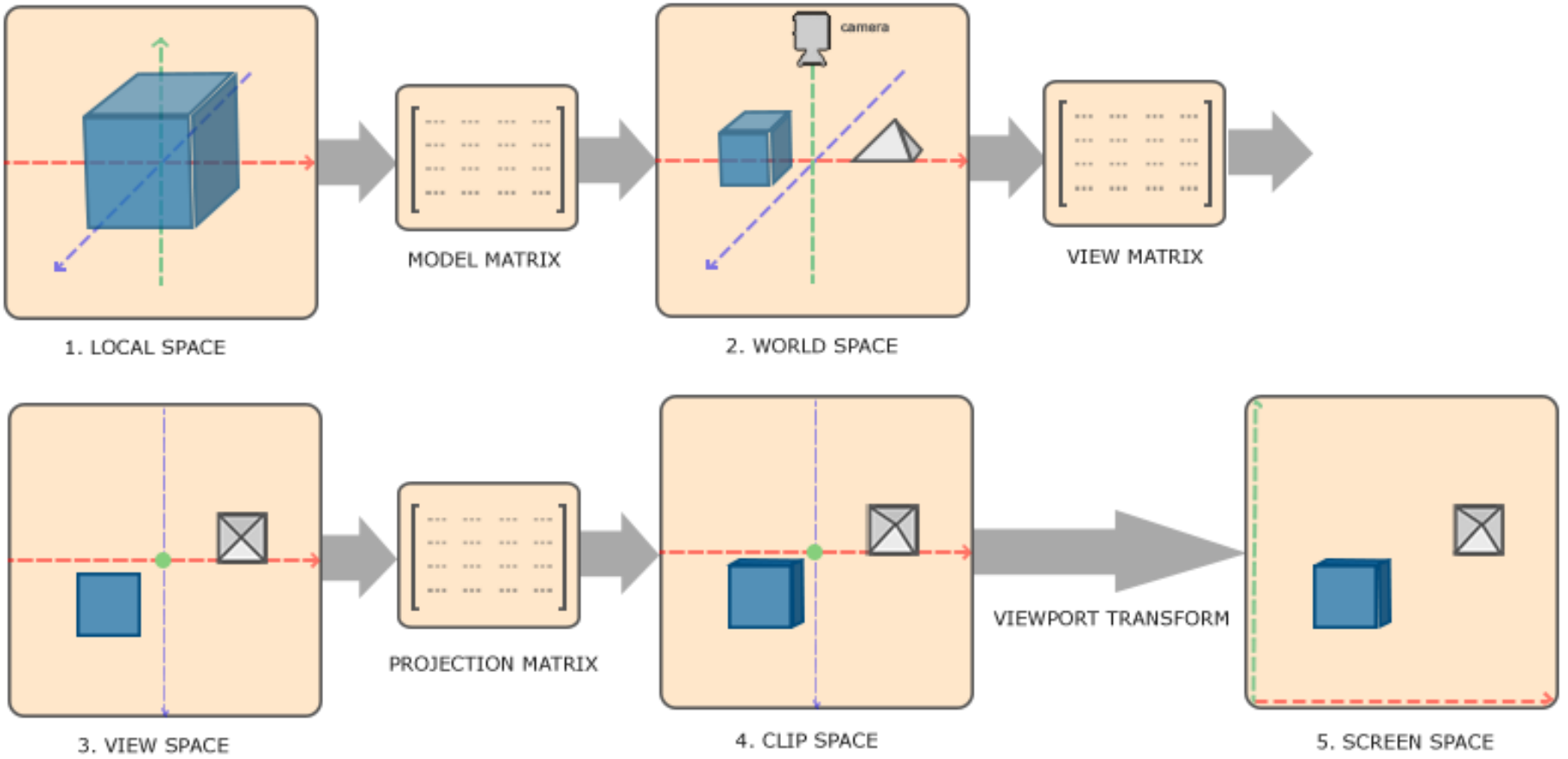
Em computação gráfica é comum a utilização de diferentes sistemas de coordenadas.

A mudança de sistema de coordenadas ocorre por meio de transformações geométricas (multiplicação de matrizes).

Principais sistemas de coordenadas:

- Espaço do Objeto (*Object Space* / *Local Space*).
- Espaço do Universo (*World Space*).
- Espaço da Câmera (*View Space* / *Camera Space* / *Eye Space*).
- Espaço do Recorte ou Projetivo (*Clip Space*).
- Espaço da Tela (*Screen Space*).

Sistema de Coordenadas



Espaço do Objeto e do Universo

Espaço do Objeto

Espaço de coordenadas local ao objeto. O objeto normalmente é criado na origem (0, 0, 0).

Espaço do Universo

É o espaço onde se define uma posição para cada objeto dentro do universo, de preferência de modo realista. Resulta da transformação do sistema de coordenadas do espaço do objeto, por meio da matriz de modelo.

A matriz de modelo move, dimensiona e / ou rotaciona objetos dentro do universo.

Espaço da Câmera

O espaço da câmera é o resultado da transformação do sistema de coordenadas do espaço do universo para o sistema de coordenadas que representa o ponto de vista do usuário (câmera).

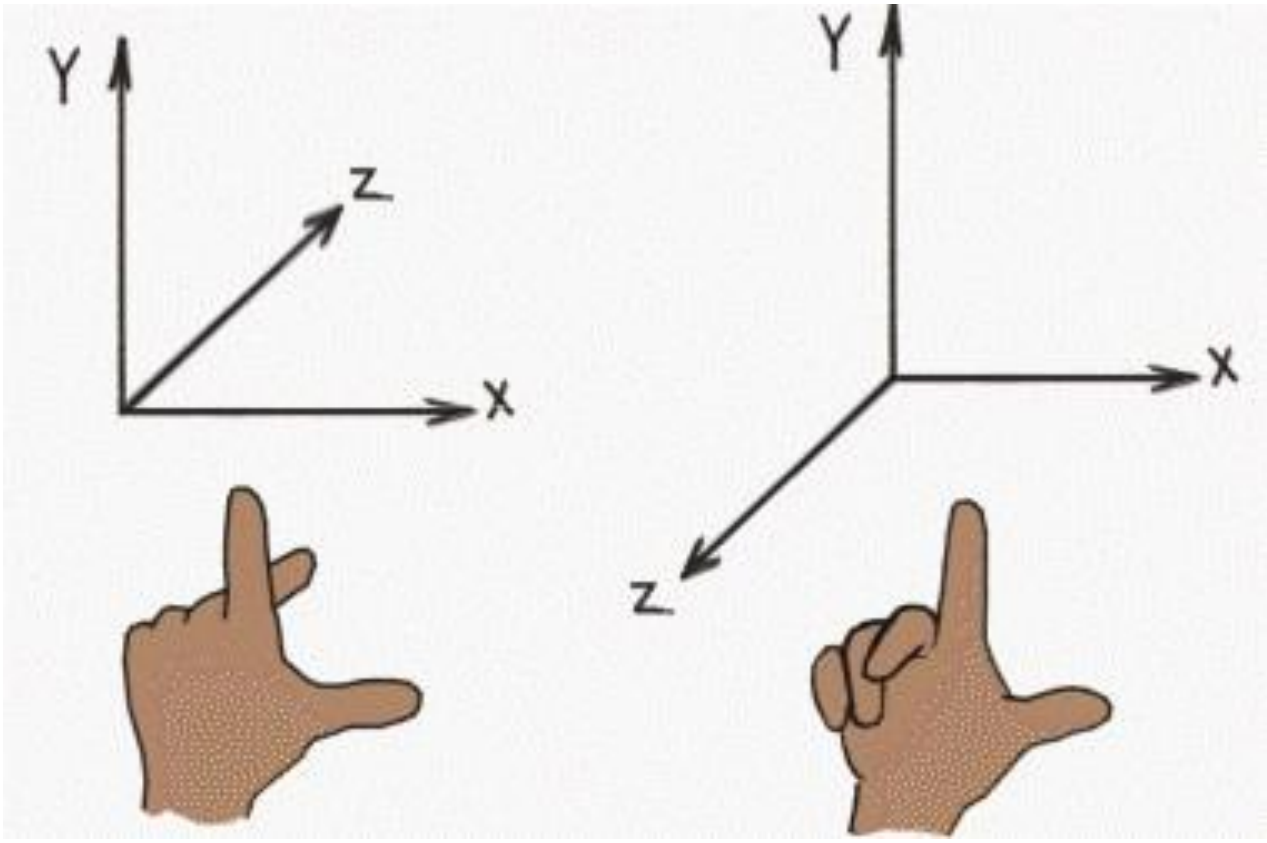
De fato, o OpenGL não define o conceito de câmera (a câmera não existe). Na prática, esse é um conceito simulado.

A câmera está sempre localizada na coordenada (0, 0, 0).

A simulação do movimento da câmera se da pelo movimento inverso da cena em foco.

A função *gluLookAt* é utilizada para construir a **matriz de visualização**.

Regra da Mão Esquerda / Direita



Mão Esquerda

Mão Direita

Matriz de Visualização

A matriz de visualização (*V*) consiste em duas transformações.

- Translação (*T*): mover a câmera até a origem.
- Rotação (*R*): rotacionar a cena (universo) inversamente ao movimento da câmera.

Assim, a câmera será posicionada na origem e voltada para o eixo -Z.

$$V = RT = \begin{bmatrix} r_0 & r_4 & r_8 & 0 \\ r_1 & r_5 & r_9 & 0 \\ r_2 & r_6 & r_{10} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

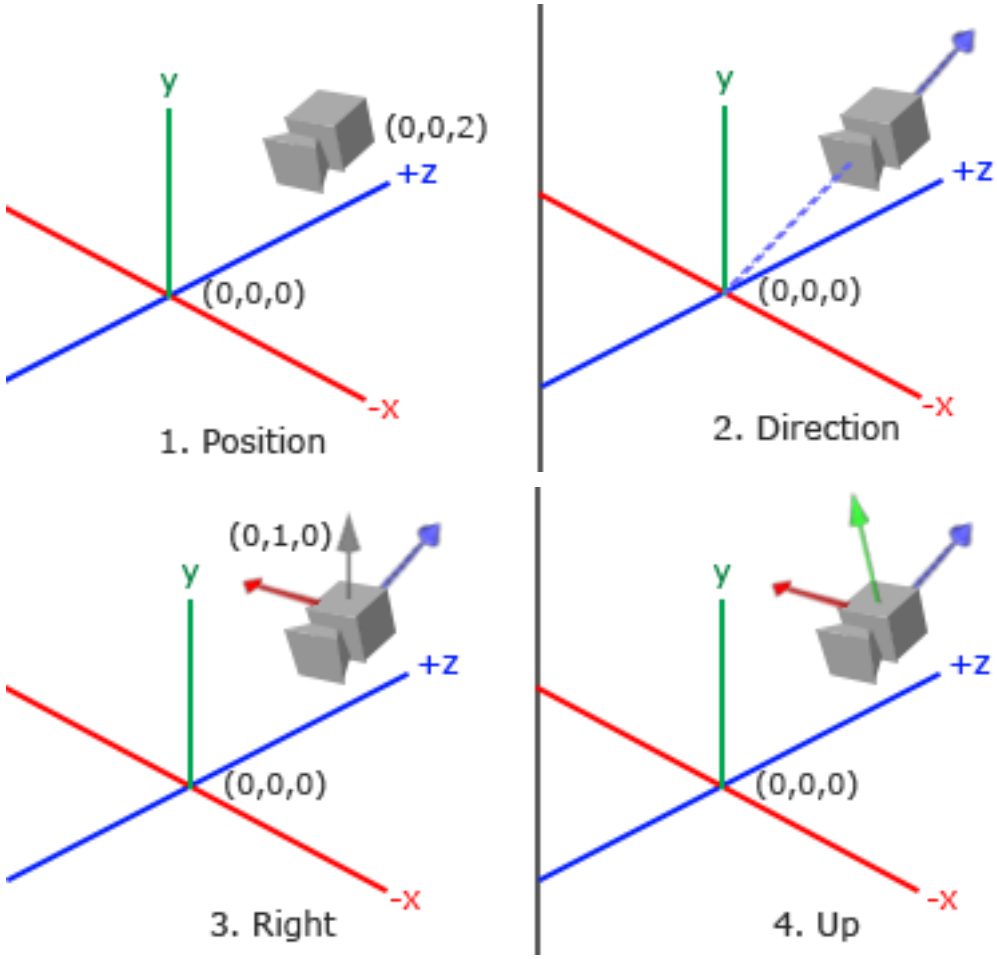
Função *gluLookAt*

gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);

Parâmetros (espaço do universo):

- *eye*: especifica a posição da câmera.
- *center*: especifica a posição do ponto de referência (centro da cena).
- *up*: especifica a direção / sentido do vetor *up*.

Sistema de Coordenadas da Câmera



Matriz de Translação

A matriz de translação (*T*) é:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

O valor de *eye* é invertido, pois, a ideia é mover a cena no sentido oposto ao da câmera.

Matriz de Rotação

Deve-se, primeiramente, definir um novo sistema de coordenadas para a câmera.

Nesse caso, uma base ortonormal em $\mathbb{R}^3$ (*u*, *v*, *w*). Processo de Gram-Schmidt.

Passo 1: calcular o vetor *w*.

$$w = \frac{eye - center}{\|eye - center\|}$$

Passo 2: calcular o vetor *u*.

$$u = \frac{up \times w}{\|up \times w\|}$$

Matriz de Rotação

Passo 3: calcular o vetor *v* (novo vetor *up*, normalizado).

$$v = w \times u$$

Passo 4: construir a matriz de rotação (*R*).

$$R = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{T} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Lembrar que a matriz de rotação deve ser invertida, pois, a cena (universo) está sendo movimentada no sentido contrário ao da câmera.

Matriz de Visualização

Finalmente, a matriz de visualização (*V*) é:

$$V = RT = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Espaço do Recorte ou Projetivo

Uma cena 3D deve ser projetada na tela do computador como uma imagem 2D. No OpenGL, esse processo se dá por meio da **matriz de projeção**.

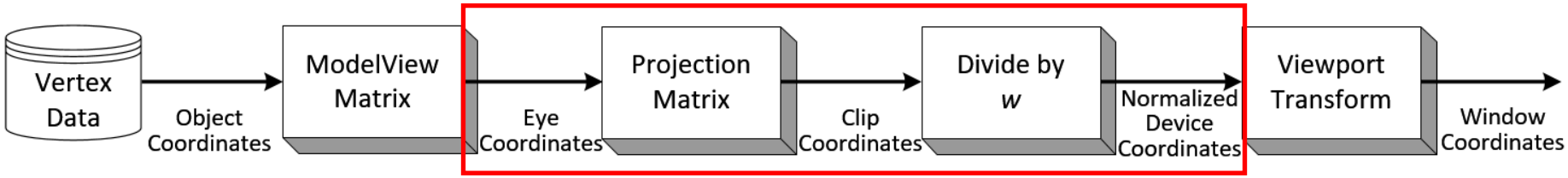
Ao final da execução de cada *vertex shader*, as coordenadas dos vértices devem estar dentro de um intervalo específico. As coordenadas fora desse intervalo são descartadas.

O espaço do recorte ou projetivo é o resultado da transformação do sistema de coordenadas do espaço da câmera para o sistema de coordenadas (do recorte) definido nesse intervalo.

Essas coordenadas são transformadas em coordenadas normalizadas do dispositivo (*Normalized Device Coordinates* – NDC), a partir da divisão dos componentes *x*, *y* e *z* pelo componente *w* das coordenadas do recorte.

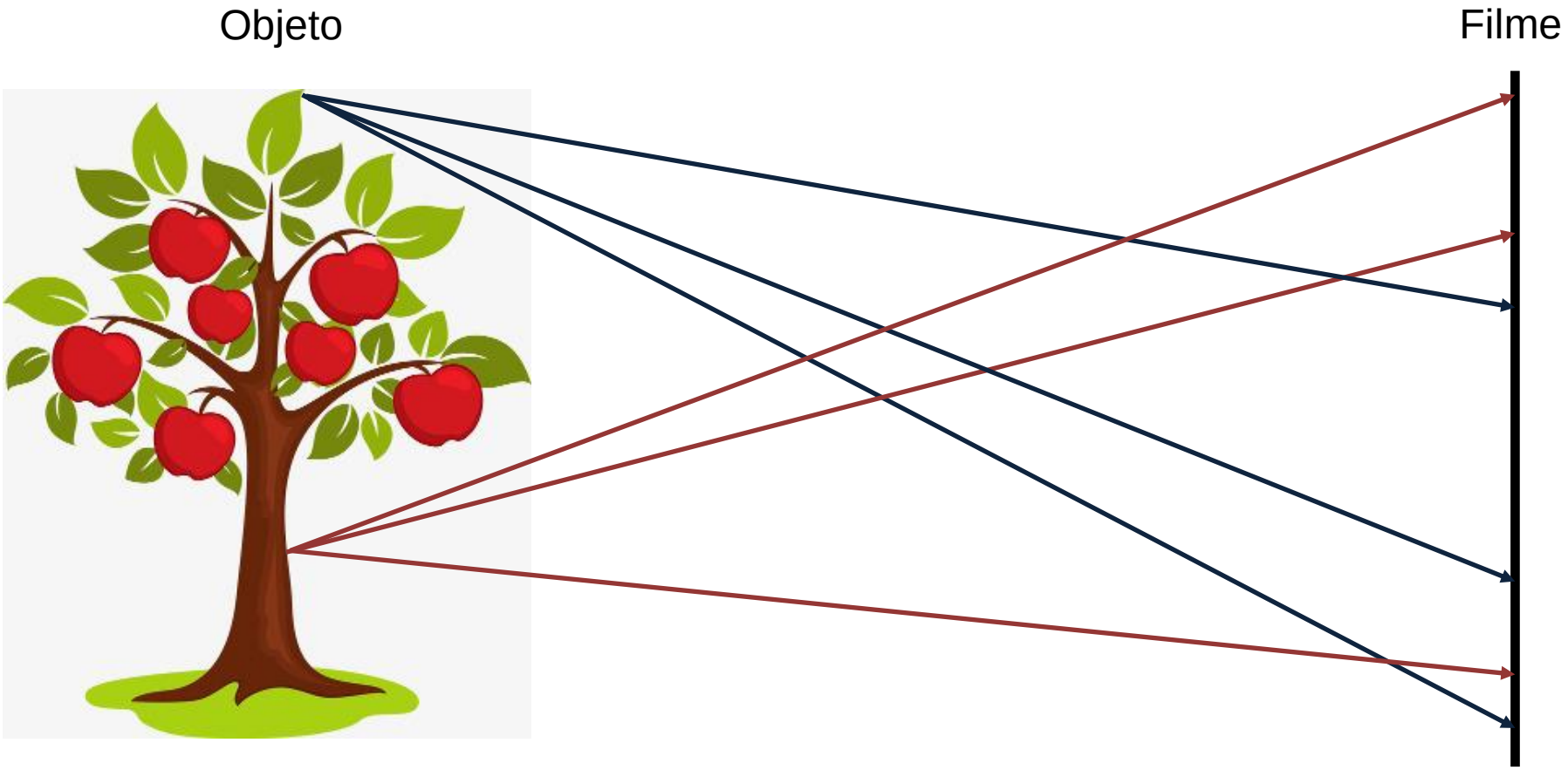
Espaço do Recorte ou Projetivo

Transformações dos vértices:

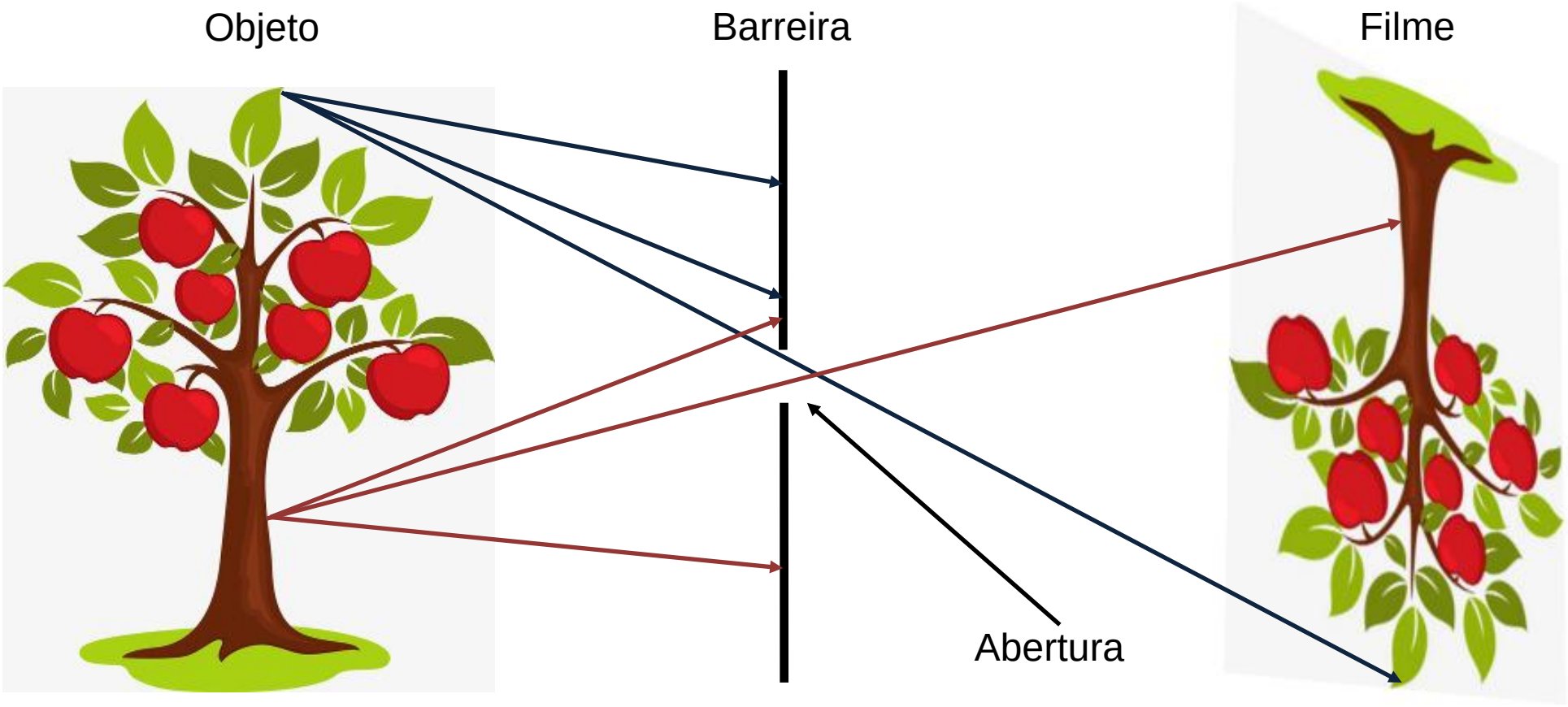


Dois tipos principais de projeções: **perspectiva** e **ortográfica**.

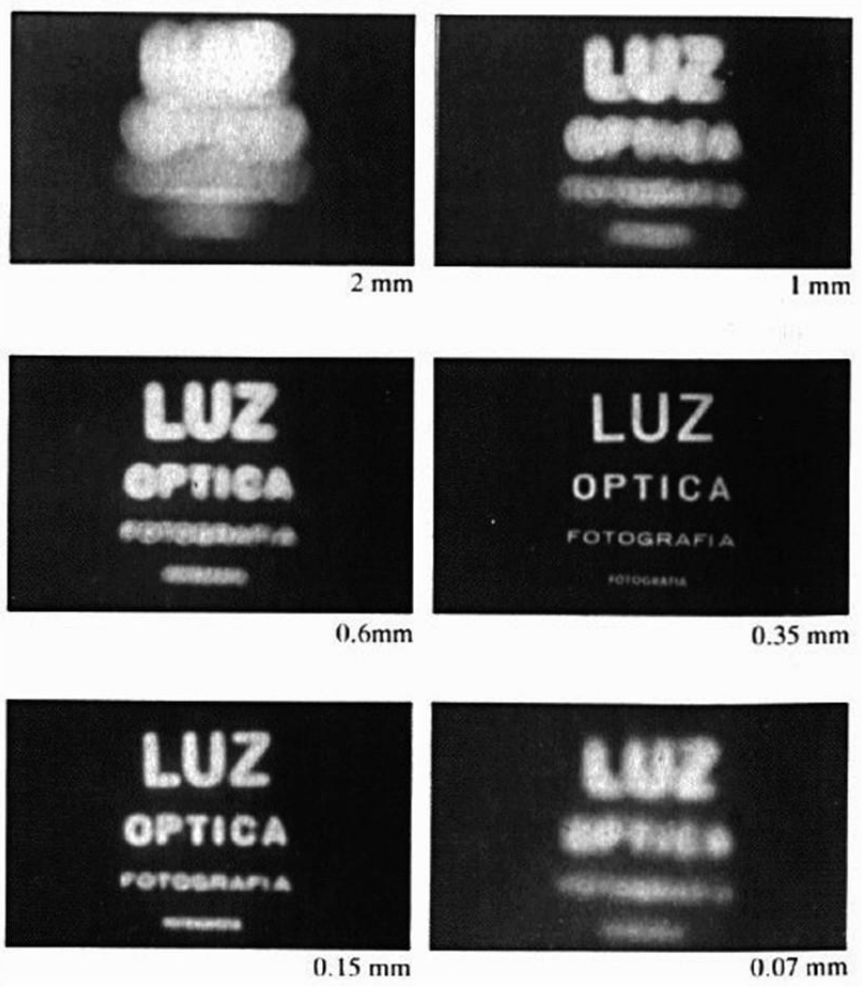
Formação de Imagem



Câmera Estenopeica (*Pinhole Camera*)



Tamanho da Abertura



Projeção Perspectiva

A projeção perspectiva é utilizada para mapear uma cena 3D em uma imagem 2D, de modo realista (como vista pelo ser humano).

Algumas propriedades:

- Objetos parecem menores à medida que as suas distâncias aumentam, em relação ao observador.
- *Foreshortening*: efeito visual que faz com que um objeto pareça menor, em função do ângulo de visualização.
- A projeção de um ponto não é única.
- Distâncias e ângulos não são preservados.
- Retas são preservadas, porém, de modo geral, retas paralelas não.
- Ponto de Fuga (*Vanishing Point*): interseção entre duas ou mais retas paralelas.

Foreshortening

Ponto de Fuga

Matriz de Projeção Perspectiva

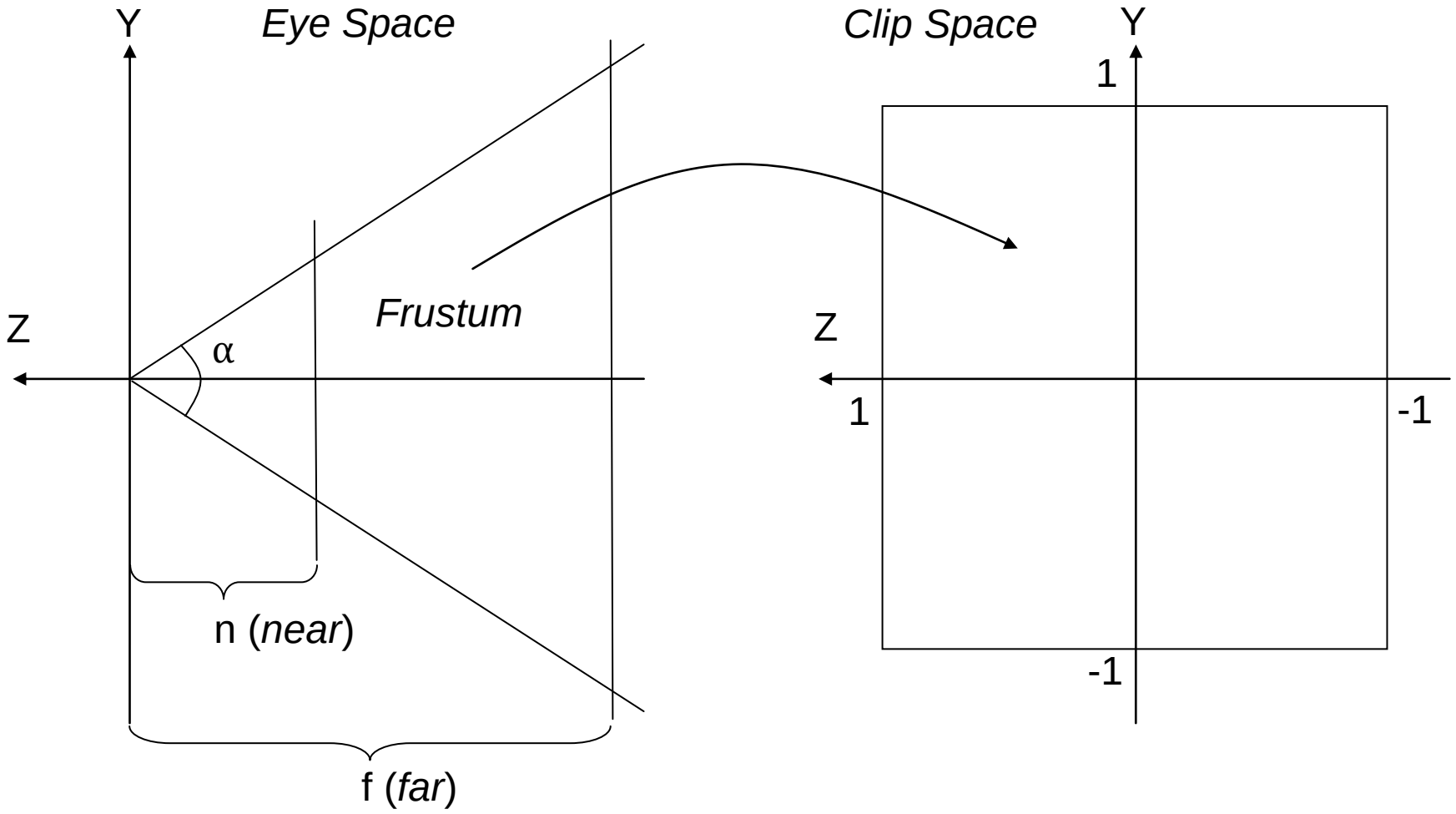
Construir a matriz de projeção perspectiva, tal que:

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix}$$

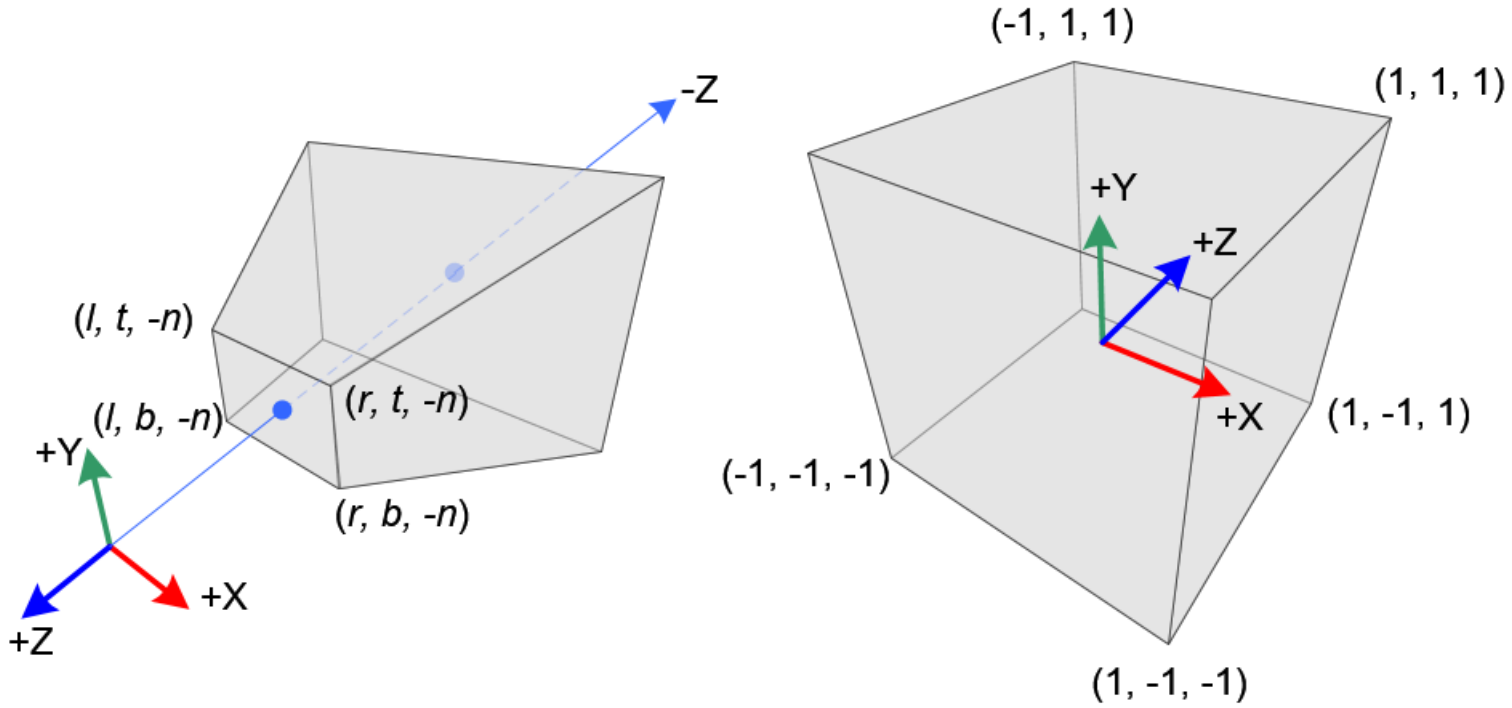
E:

$$\begin{bmatrix} x_{nde} \\ y_{nde} \\ z_{nde} \end{bmatrix} = \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{bmatrix}$$

Projeção Perspectiva



Projeção Perspectiva



Derivação da Matriz de Projeção Perspectiva

Construir a matriz de projeção a partir dos parâmetros:

left (*l*), *right* (*r*), *bottom* (*b*), *top* (*t*), *near* (*n*), *far* (*f*).

Coordenada *x*: [*l*, *r*] → [-1, 1]

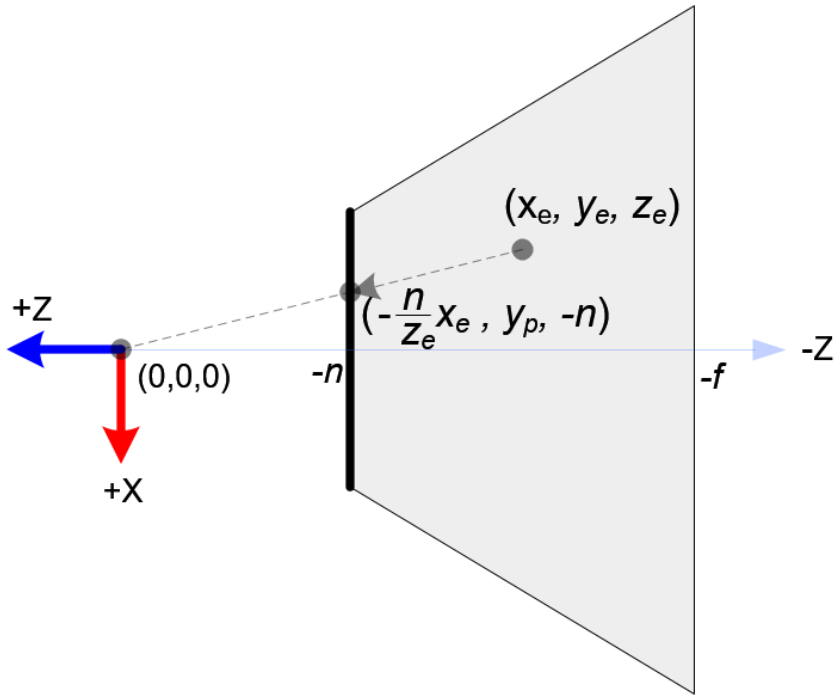
Coordenada *y*: [*b*, *t*] → [-1, 1]

Coordenada *z*: [*n*, *f*] → [-1, 1]

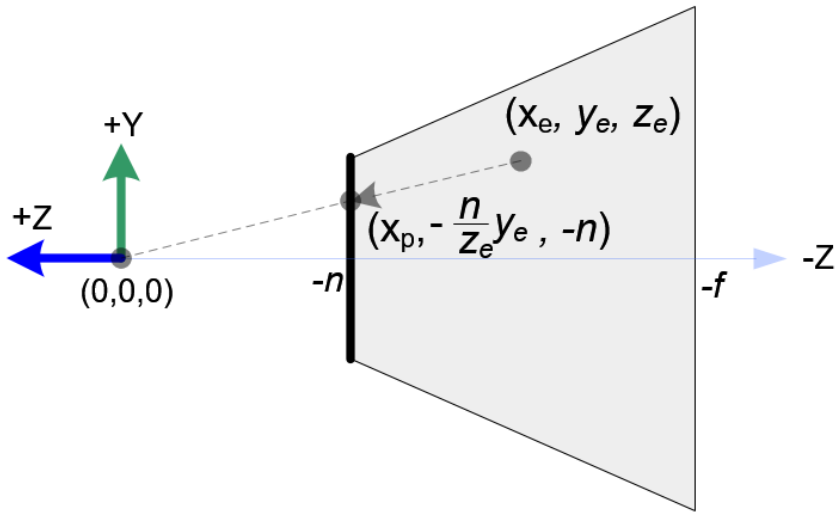
No OpenGL, um ponto 3D (x_e, y_e, z_e) no espaço da câmera é projetado em um ponto (x_p, y_p, z_p) no plano *near* (plano de projeção).

Derivação da Matriz de Projeção Perspectiva

Vista superior do *Frustum*



Vista lateral do *Frustum*



Derivação da Matriz de Projeção Perspectiva

Pela semelhança entre triângulos, tem-se:

$$\frac{x_p}{x_e} = \frac{-n}{z_e} \quad \therefore \quad x_p = \frac{n \cdot x_e}{-z_e} \qquad (1)$$

$$\frac{y_p}{y_e} = \frac{-n}{z_e} \quad \therefore \quad y_p = \frac{n \cdot y_e}{-z_e} \qquad (2)$$

Observa-se que x_p e y_p são inversamente proporcionais a $-z_e$.

Derivação da Matriz de Projeção Perspectiva

Antes:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}, \qquad \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \begin{bmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{bmatrix}$$

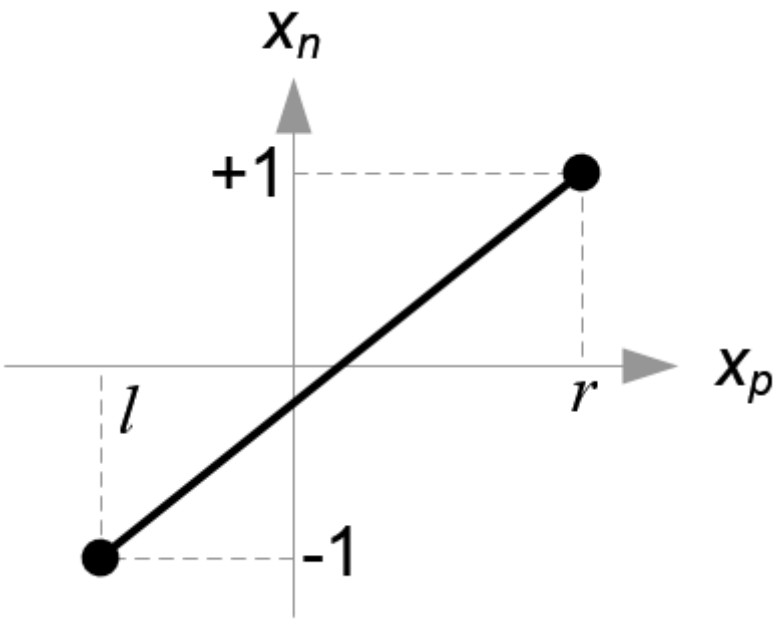
Agora:

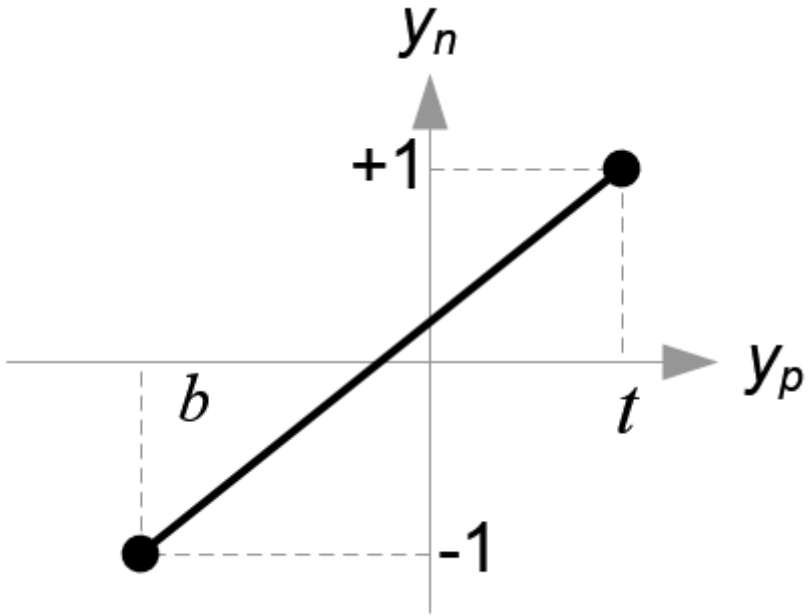
$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} \quad \therefore \quad w_c = -z_e$$

Derivação da Matriz de Projeção Perspectiva

Mapear x_p e y_p para x_n e y_n, respectivamente, por meio da relação linear:

$$[l, r] \rightarrow [-1, 1] \text{ e } [b, t] \rightarrow [-1, 1]$$





Derivação da Matriz de Projeção Perspectiva

Assim:

$$x_n = mx_p + b \quad \therefore \quad x_n = \frac{2x_p}{r-l} - \frac{r+l}{r-l}$$

$$y_n = my_p + b \quad \therefore \quad y_n = \frac{2y_p}{t-b} - \frac{t+b}{t-b}$$

Substituindo x_p e y_p nas equações (1) e (2):

$$x_n = \frac{\frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e}{-z_e} = \frac{x_c}{-z_e}$$

$$y_n = \frac{\frac{2n}{t-b}y_e + \frac{t+b}{t-b}z_e}{-z_e} = \frac{y_c}{-z_e}$$

Derivação da Matriz de Projeção Perspectiva

Atualizando as linhas 1 e 2 da matriz de projeção:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix}$$

Falta calcular os coeficientes α e β. Observa-se que no espaço da câmera, w_e = 1.

$$z_n = \frac{z_c}{w_c} = \frac{\alpha z_e + \beta w_e}{-z_e} = \frac{\alpha z_e + \beta}{-z_e}$$

Derivação da Matriz de Projeção Perspectiva

Utilizar a relação (z_e, z_n): (-*n*, -1) e (-*f*, 1).

$$\frac{-\alpha n + \beta}{n} = -1, \qquad \frac{-\alpha f + \beta}{f} = 1$$

Portanto:

$$\alpha = -\frac{f+n}{f-n}, \qquad \beta = -\frac{2fn}{f-n}$$

E:

$$z_n = \frac{-\frac{f+n}{f-n} z_e - \frac{2fn}{f-n}}{-z_e}$$

Derivação da Matriz de Projeção Perspectiva

Finalmente, a matriz de projeção perspectiva (*P*) é igual a:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

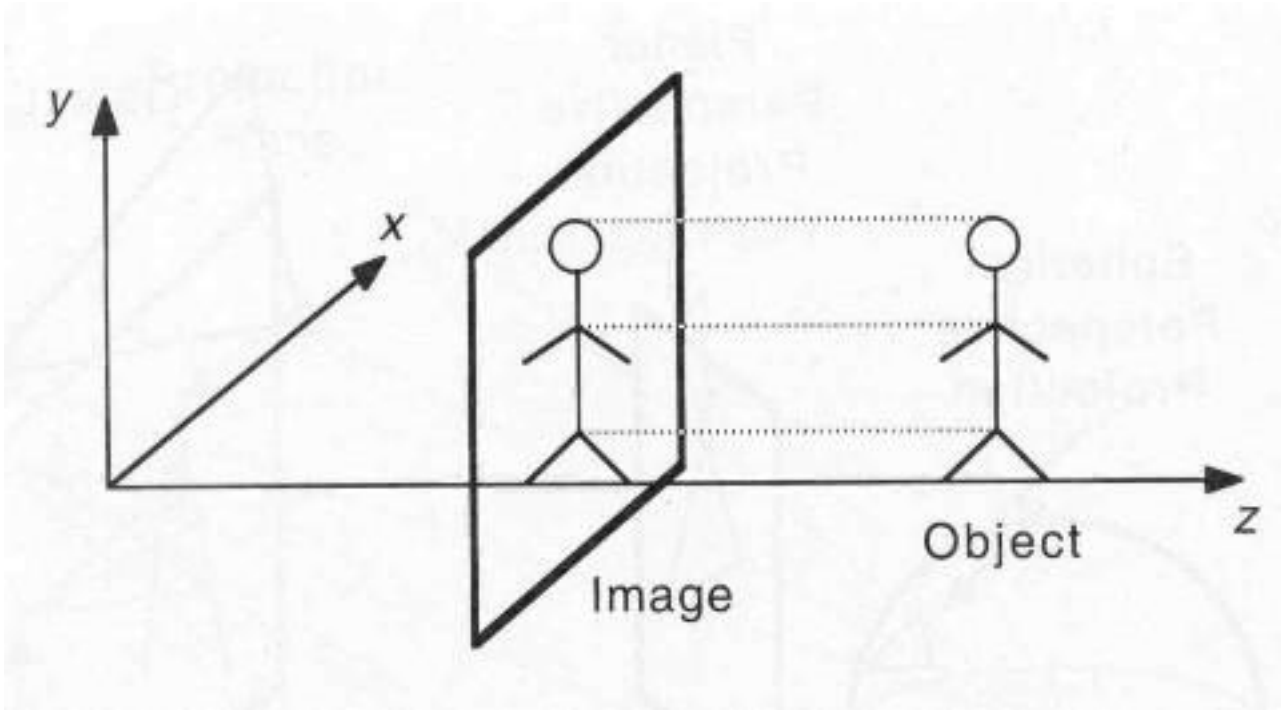
Projeção Ortográfica ou Paralela

A projeção ortográfica é utilizada para mapear uma cena 3D em uma imagem 2D, sendo que o plano da imagem é paralelo ao plano XY e as linhas de projeção, são paralelas ao eixo Z.

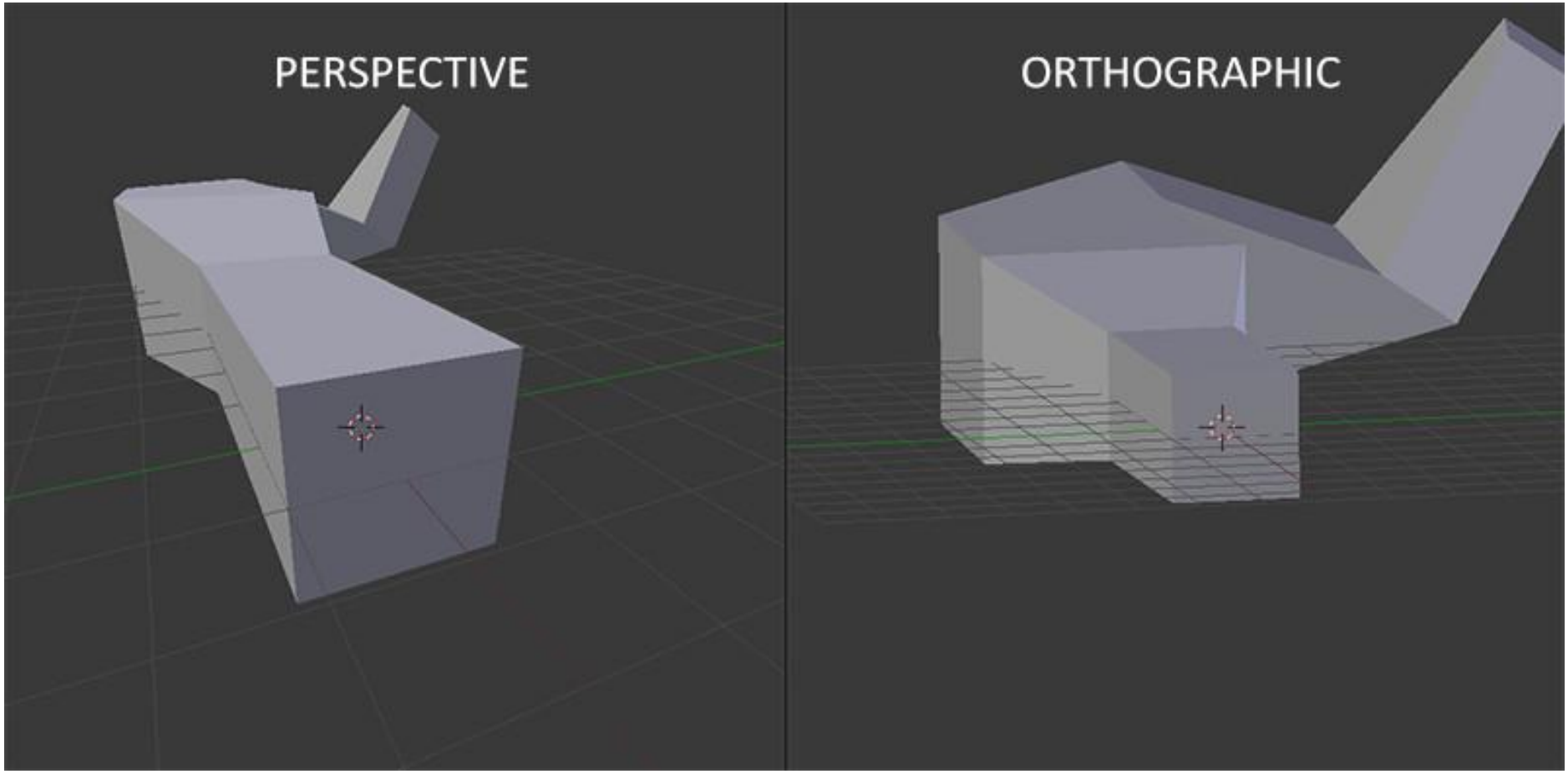
Algumas propriedades:

- Distância entre o centro da projeção e o plano da imagem é infinita.
- As dimensões dos objetos não mudam à medida que as suas distâncias aumentam, em relação ao observador.
- Retas paralelas são preservadas.
- Utilizada em projetos de engenharia.

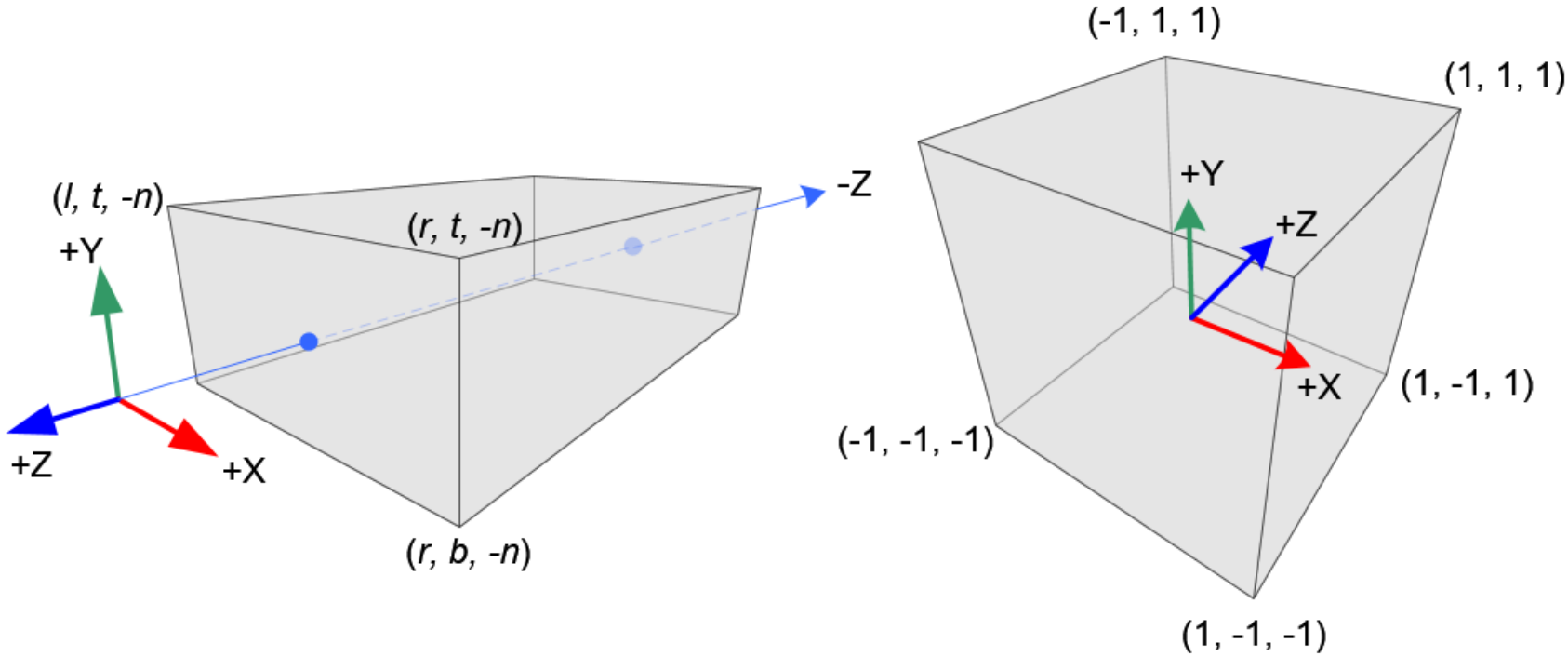
Projeção Ortográfica ou Paralela



Projeção Perspectiva X Ortográfica



Matriz de Projeção Ortográfica



Matriz de Projeção Ortográfica

A matriz de projeção ortográfica (*P*) é igual a:

$$P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exemplo: Cubo

Passo 1: definir os vértices e índices.

```
vertices = np.array([
                        # Frente              Cores
                        -0.5,  0.5,  0.5, 1.0, 0.0, 0.0,
                         0.5,  0.5,  0.5, 0.0, 1.0, 0.0,
                        -0.5, -0.5,  0.5, 0.0, 0.0, 1.0,
                         0.5, -0.5,  0.5, 1.0, 1.0, 1.0,

                        # Fundo               Cores
                        -0.5,  0.5, -0.5, 1.0, 0.0, 0.0,
                         0.5,  0.5, -0.5, 0.0, 1.0, 0.0,
                        -0.5, -0.5, -0.5, 0.0, 0.0, 1.0,
                         0.5, -0.5, -0.5, 1.0, 1.0, 1.0,
```

Exemplo: Cubo

Passo 1: definir os vértices e índices.

```
# Esquerda          Cores
-0.5,  0.5,  0.5, 1.0, 0.0, 0.0,
-0.5,  0.5, -0.5, 0.0, 1.0, 0.0,
-0.5, -0.5,  0.5, 0.0, 0.0, 1.0,
-0.5, -0.5, -0.5, 1.0, 1.0, 1.0,

# Direita           Cores
 0.5,  0.5,  0.5, 1.0, 0.0, 0.0,
 0.5,  0.5, -0.5, 0.0, 1.0, 0.0,
-0.5, -0.5,  0.5, 0.0, 0.0, 1.0,
-0.5, -0.5, -0.5, 1.0, 1.0, 1.0,
```

Exemplo: Cubo

Passo 1: definir os vértices e índices.

```
                # Topo                Cores
                -0.5,  0.5, -0.5, 1.0, 0.0, 0.0,
                 0.5,  0.5, -0.5, 0.0, 1.0, 0.0,
                -0.5,  0.5,  0.5, 0.0, 0.0, 1.0,
                 0.5,  0.5,  0.5, 1.0, 1.0, 1.0,

                # Base                Cores
                -0.5, -0.5, -0.5, 1.0, 0.0, 0.0,
                 0.5, -0.5, -0.5, 0.0, 1.0, 0.0,
                -0.5, -0.5,  0.5, 0.0, 0.0, 1.0,
                 0.5, -0.5,  0.5, 1.0, 1.0, 1.0
            ], dtype=np.float32)
```

Exemplo: Cubo

Passo 1: definir os vértices e índices.

```
indices = np.array([ 0,  1,  2,  2,  3,  0,
                     4,  5,  6,  6,  7,  4,
                     8,  9, 10, 10, 11,  8,
                    12, 13, 14, 14, 15, 12,
                    16, 17, 18, 18, 19, 16,
                    20, 21, 22, 22, 23, 20
                   ], dtype=np.uint32)
```

Exemplo: Cubo

Passo 2: criar o *vertex shader*.

```
VERTEX_SHADER = '''#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 color;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
out vec3 vertex_color;
void main()
{
    gl_Position = projection * view * model * vec4(position,
                                                   1.0f);
    vertex_color = color;
}'''
```

Exemplo: Cubo

Passo 3: criar o *fragment shader*.

```
FRAGMENT_SHADER = '''#version 330 core
in vec3 vertex_color;
out vec4 color;
void main()
{
    color = vec4(vertex_color, 1.0f);
}'''
```

Exemplo: Cubo

Passo 4: definir a matriz de modelo.

```
model = pyrr.Matrix44.from_scale(scale=(1.1, 1.1, 1.1),
                                 dtype=np.float16)
model *= pyrr.Matrix44.from_x_rotation(np.radians(-45.),
                                       dtype=np.float16)
model *= pyrr.Matrix44.from_y_rotation(np.radians(-45.),
                                       dtype=np.float16)
model = model.astype(np.float32)
```

Exemplo: Cubo

Passo 5: definir a matriz de visualização.

```
view = pyrr.Matrix44.look_at(
          pyrr.Vector3([0., 0., 3.], dtype=np.float16), # eye
          pyrr.Vector3([0., 0., 0.], dtype=np.float16), # center
          pyrr.Vector3([0., 1., 0.], dtype=np.float16), # up
          dtype=np.float32)
```

Exemplo: Cubo

Passo 6: definir a matriz de projeção.

```
projection = pyrr.Matrix44.perspective_projection(
                             45.,                    # fovy
                             width / height,         # aspect ratio
                             .1,                     # near
                             100,                    # far
                             dtype=np.float32)
```

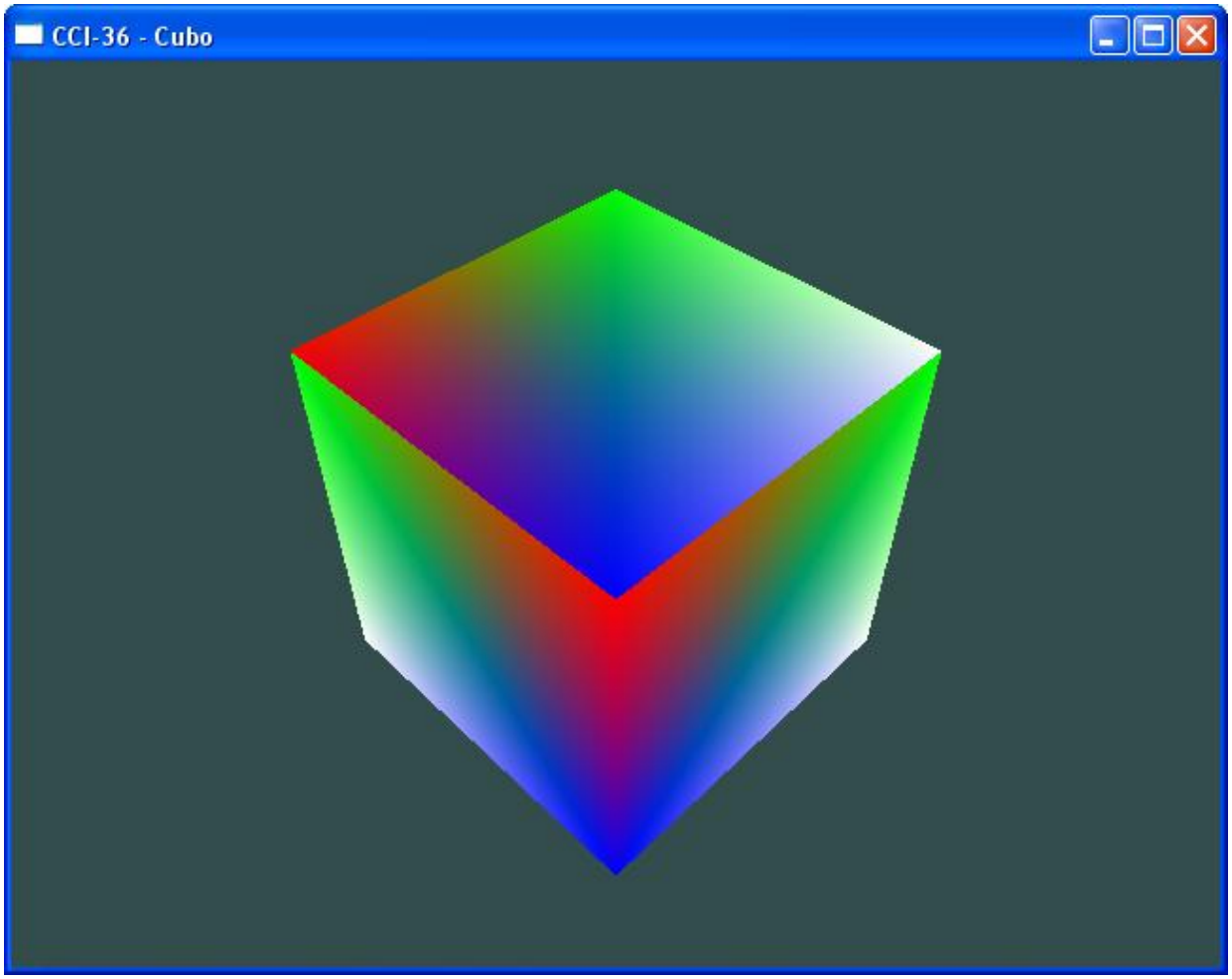
Exemplo: Cubo

Passo 7: alocar as matrizes (variáveis uniformes).

```
model_loc = glGetUniformLocation(program, 'model')
view_loc = glGetUniformLocation(program, 'view')
proj_loc = glGetUniformLocation(program, 'projection')

glUniformMatrix4fv(model_loc, 1, GL_FALSE, model)
glUniformMatrix4fv(view_loc, 1, GL_FALSE, view)
glUniformMatrix4fv(proj_loc, 1, GL_FALSE, projection)
```

Exemplo: Cubo



Exemplo Sugerido – Esfera

Uma esfera pode ser definida, a partir dos ângulos azimute (θ) e polar (φ), conforme a seguinte equação paramétrica:

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