DCC192

2025/2

Desenvolvimento de Jogos Digitais

A18: Câmeras 3D

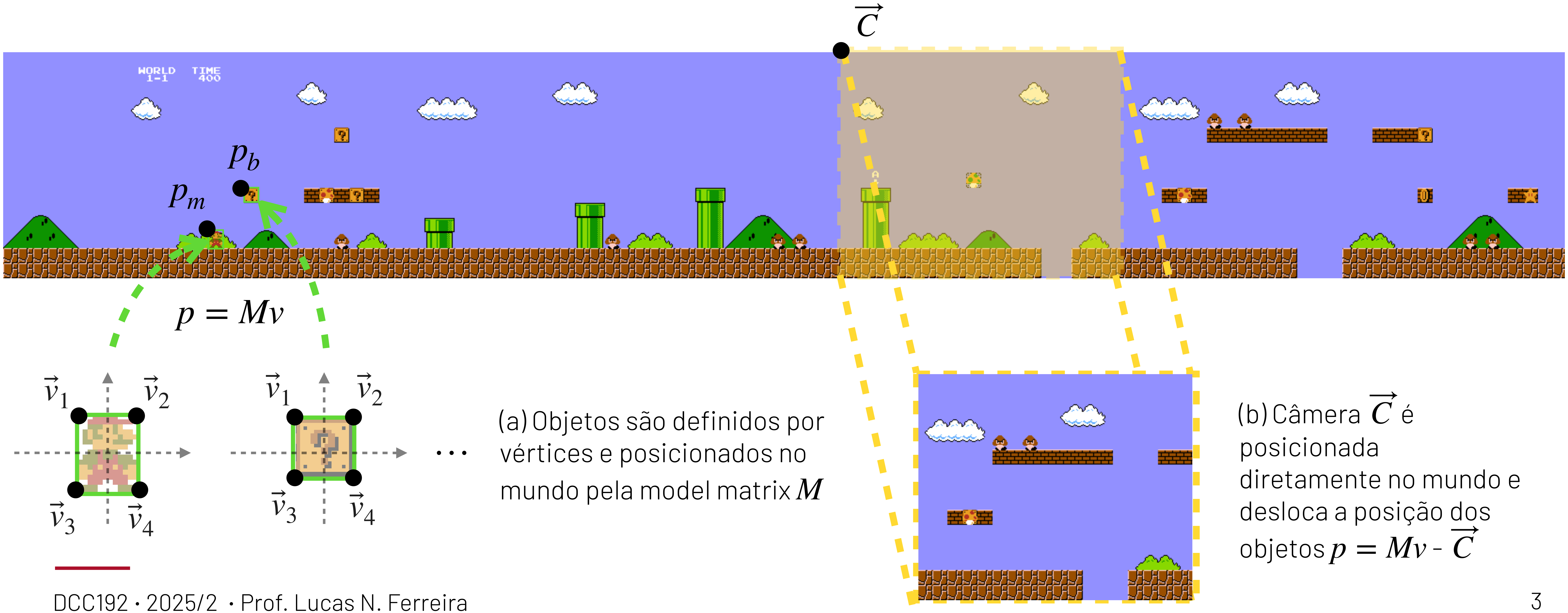
Prof. Lucas N. Ferreira

Plano de aula

- Cenas 2D e 3D
- Divisão de Perspectiva
- Matriz de Câmera (View Matrix)
- Matriz de Projeção (Projection Matrix)
- Câmera em Primeira Pessoa
- Câmera em Terceira Pessoa

Cenas 2D

Uma cena 2D é composta minimamente por **(a) um conjunto de objetos** e **(b) uma câmera**, onde os objetos são específicados por seus vértices e a câmera é definida em função



(a) Objetos são definidos por vértices e posicionados no mundo pela model matrix M

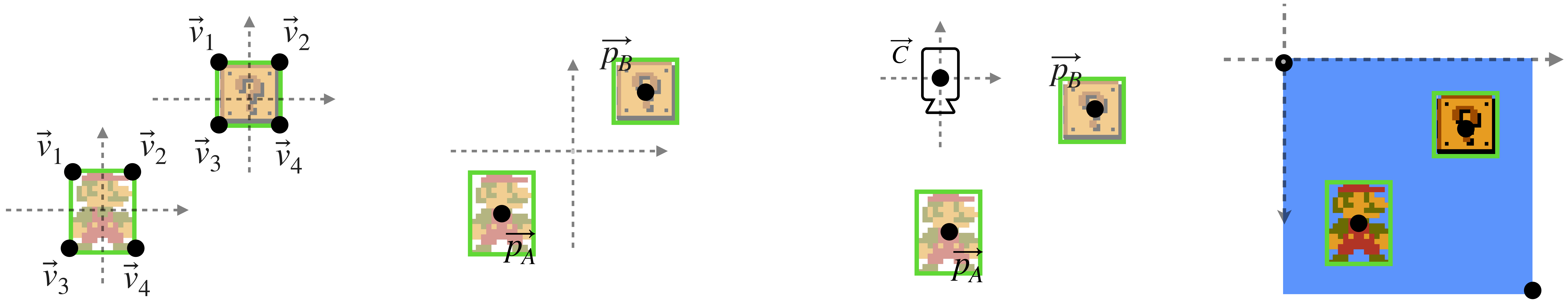
(b) Câmera $\vec{C}$ é posicionada diretamente no mundo e desloca a posição dos objetos $p = Mv - \vec{C}$

Cenas 2D

Analisando uma cena 2D formalmente, o que está acontecendo é a mudança de sistemas de coordenadas: Objeto → Mundo → Câmera → Recorte (Clip) → Tela

$$\vec{p} = M\vec{v} \qquad \vec{p}' = (M\vec{v}) - \vec{C} \qquad \vec{p}' = P((M\vec{v}) - \vec{C})$$

1. Sist. de Coords. do Objeto → (Model Matrix M) → 2. Sist. de Coords. do Mundo → (Posição da Câmera $\vec{C}$) → 3. Sist. de Coords. da Câmera → (Projection Matrix P) → 4. Sist. de Coords. de Recorte (Clip)



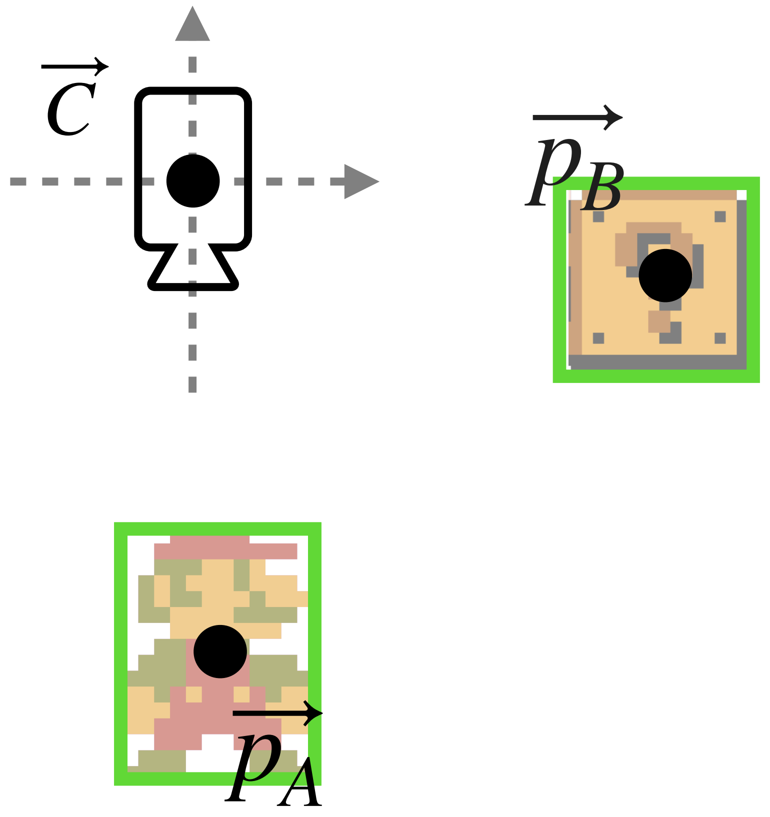
Matriz da Câmera (View Matrix V)

A subtração por $\vec{V}$ para passar de 2. para 3. é uma translação simples e, portanto, podemos reprentá-la por uma matriz também, realizando todas as etapas por multiplicação:

$\vec{p} = M\vec{v}$ $\cancel{\vec{p}' = (M\vec{v}) - \vec{C}}$ $\vec{p}' = VM\vec{v}$ $\vec{p}' = PVM\vec{v}$

1. Sist. de Coords. do Objeto $\xrightarrow{\text{Model Matrix } M}$ 2. Sist. de Coords. do Mundo $\xrightarrow{\text{View Matrix } V}$ 3. Sist. de Coords. da Câmera $\xrightarrow{\text{Projection Matrix } P}$ 4. Sist. de Coords. de Recorte (Clip)

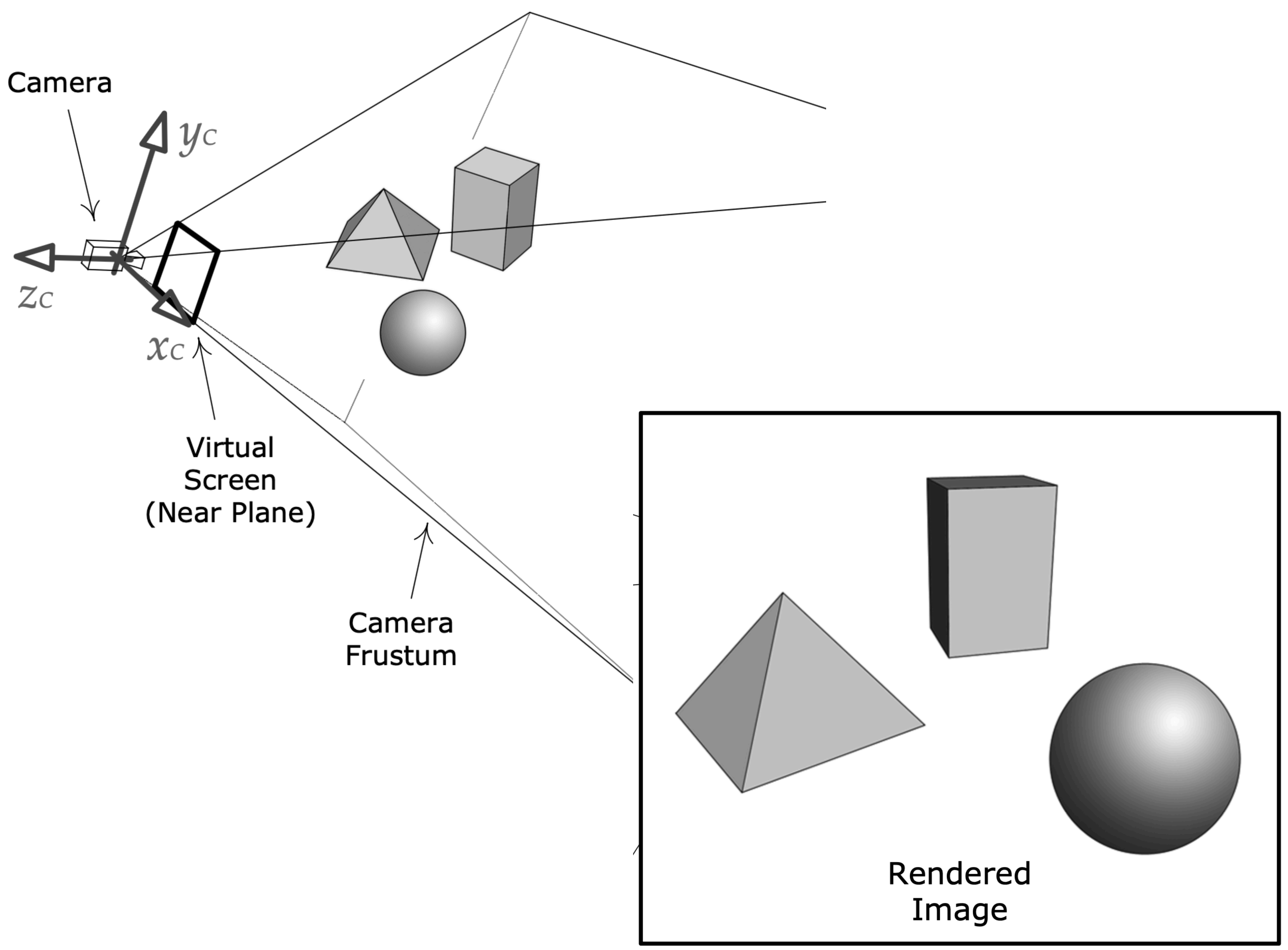
$$V = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Cenas 3D

Uma cena 3D também é composta minimamente por **(a) um conjunto de objetos** e **(b) uma câmera**, onde os objetos são específicados da mesma maneira, mas a câmera é diferente:

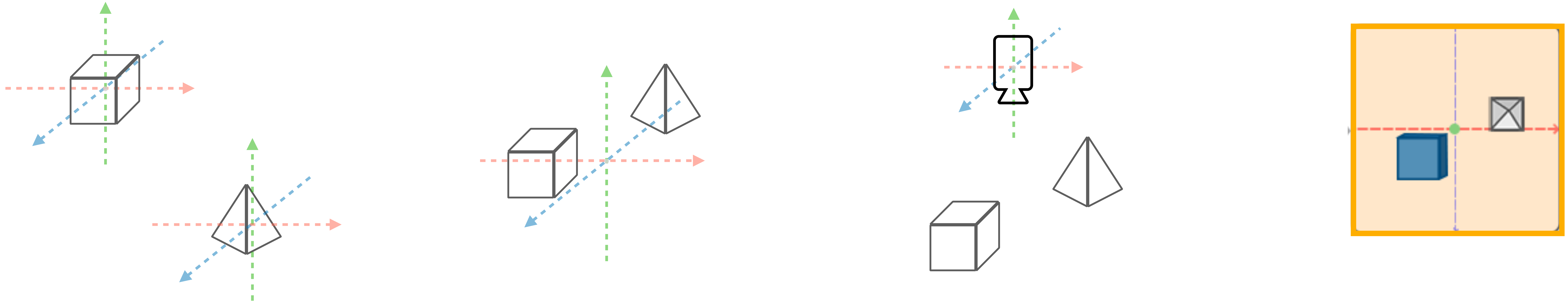
- **Objetos 3D**
  Geralmente representados por conjuntos de vértices.
- **Câmera**
  Geralmente representada por uma posição, orientação, distância focal e planos de recorte próximo e distante
- **Fonte de Luz**
  Vários tipos de fontes de luz podem ser específicadas: direcional, ambiente e spot.
- **Materiais**
  Propriedades visuais dos objetos, descrevendo como a luz deve interagir com os objetos.



Cenas 3D

O mesmo processo de transformações ocorre em cenas 3D, porém as matrízes de Câmera (view) e Projeção (Projection) são diferentes:

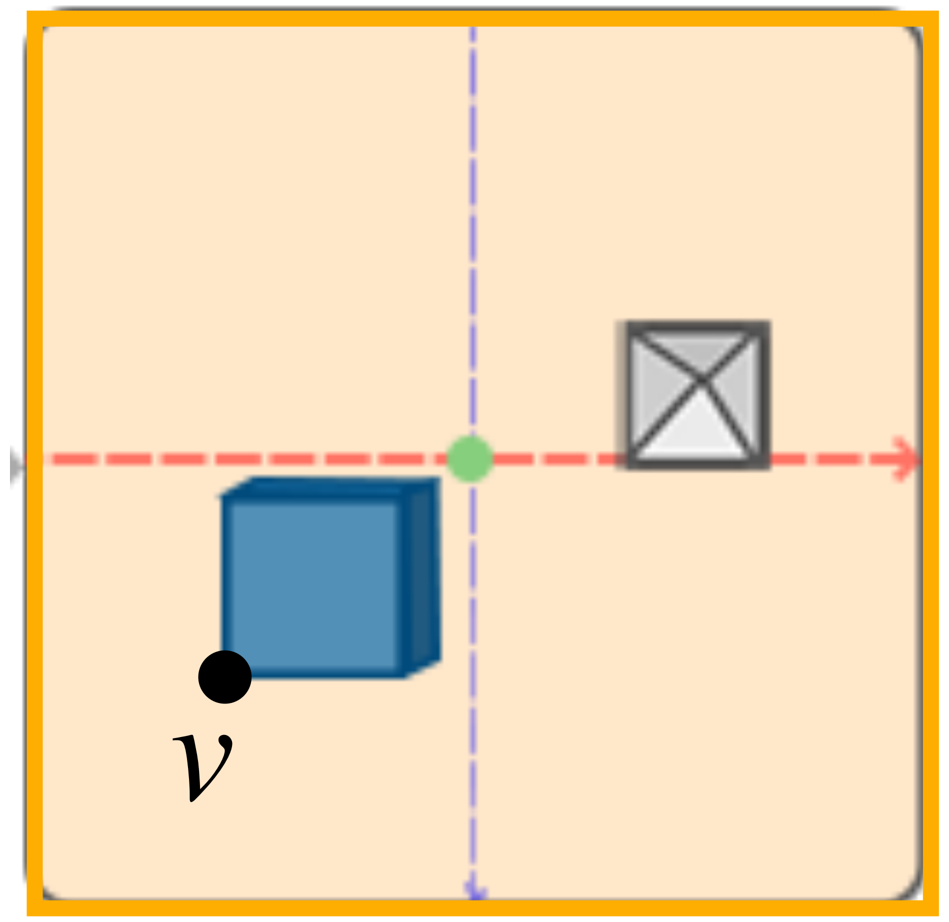
1. Sist. de Coords. do Objeto $\xrightarrow[\text{Model Matrix}]{\vec{p} = M\vec{v}}$ 2. Sist. de Coords. do Mundo $\xrightarrow[\text{View Matrix}]{\vec{p}' = VM\vec{v}}$ 3. Sist. de Coords. da Câmera $\xrightarrow[\text{Projection Matrix}]{\vec{p}' = PVM\vec{v}}$ 4. Sist. de Coords. De Recorte (Clip)



Divisão de Perspectiva (*Perspective Divide*)

Após o espaço de recorte, uma operação chamada **Divisão de Perspectiva** é realizada automaticamente no pipeline, mapeando as coordenadas homogêneas 4D dos vértices para o *normalized device coordinates* (NDC) da OpenGL $[-1,1]^3$:

4. Sist. de Coords. de Recorte

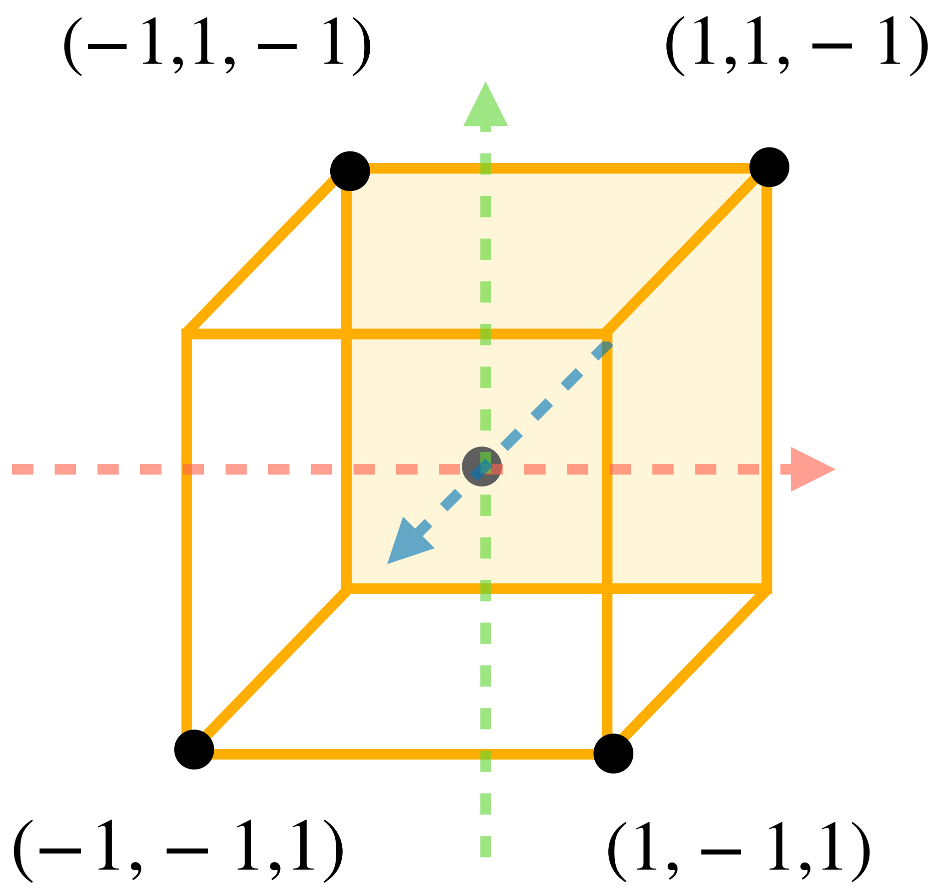


Coordenadas Homogêneas *4D*

$$v = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \xrightarrow{\text{Divisão Perspectiva}} \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{bmatrix}$$

NDC

3D



Note que se $w = 1$, a divisão perspectiva não terá efeito algum na posição dos vértices, mas se $w \neq 1$, elas serão reescalada pelo fator $1/w$. Isso será útil na projeção perspectiva!

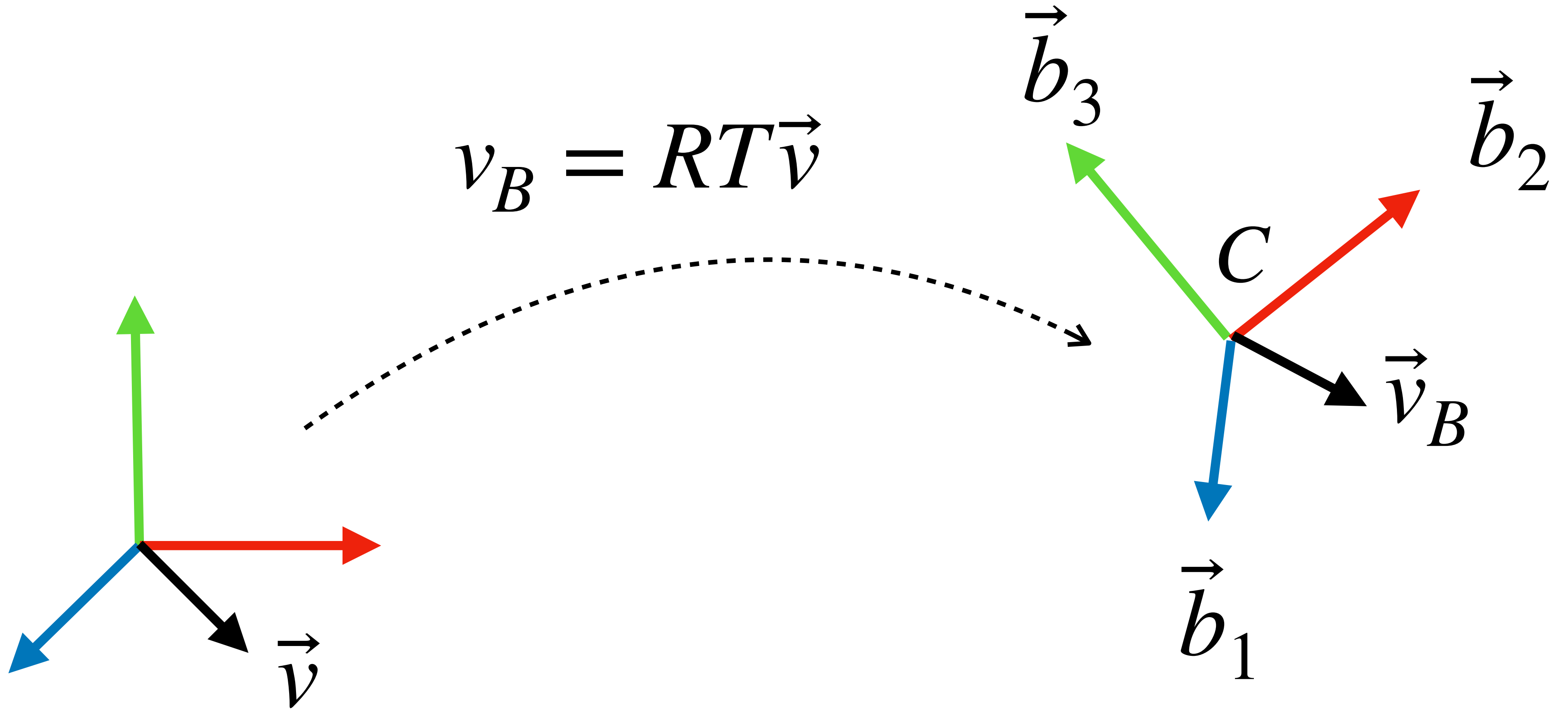
Matriz da Câmera (View Matrix V)

A **Matriz de Câmera** (View Matrix V) faz uma **mudança de base** entre o sistema de coordenadas do mundo e o sistema de coordenadas da câmera, definido :

Lembrando que para fazer a mudança de base de um vetor $\vec{v}$ para uma nova base $B = (\vec{b}_1, \vec{b}_2, \vec{b}_3)$:

1. Criar uma matriz T de translação para mudar a origem para o ponto C (translação)
2. Criar uma matriz R com os três eixos $(\vec{b}^1, \vec{b}^2, \vec{b}^3)$ para projetar $\vec{v}$ em $\vec{b}_1$, $\vec{b}_2$ e $\vec{b}_3$ (rotação)
3. Definir a matriz como $V = R \cdot T$ e multiplicar por $\vec{v}$

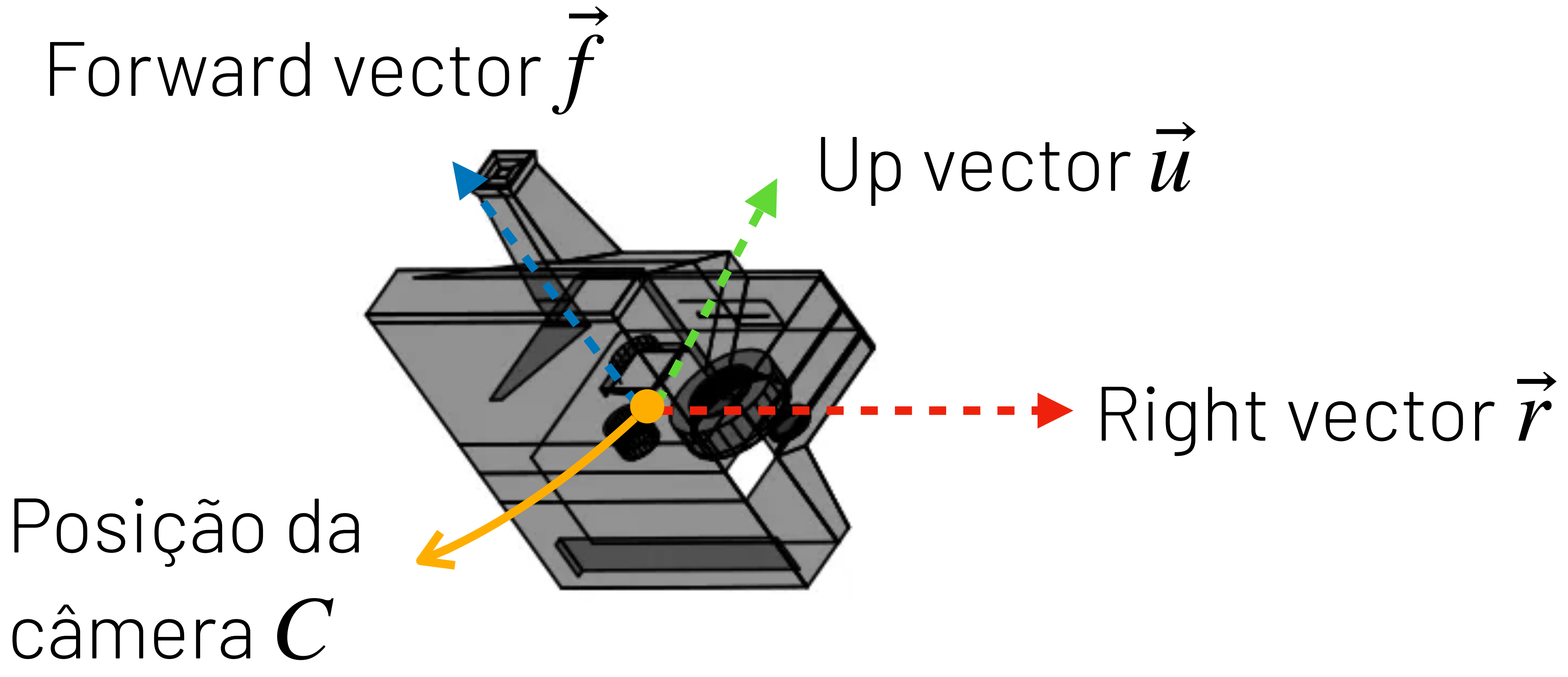
$$V = \underbrace{\begin{bmatrix} b_x^1 & b_y^1 & b_z^1 & 0 \\ b_x^2 & b_y^2 & b_z^2 & 0 \\ b_x^3 & b_y^3 & b_z^3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T}$$

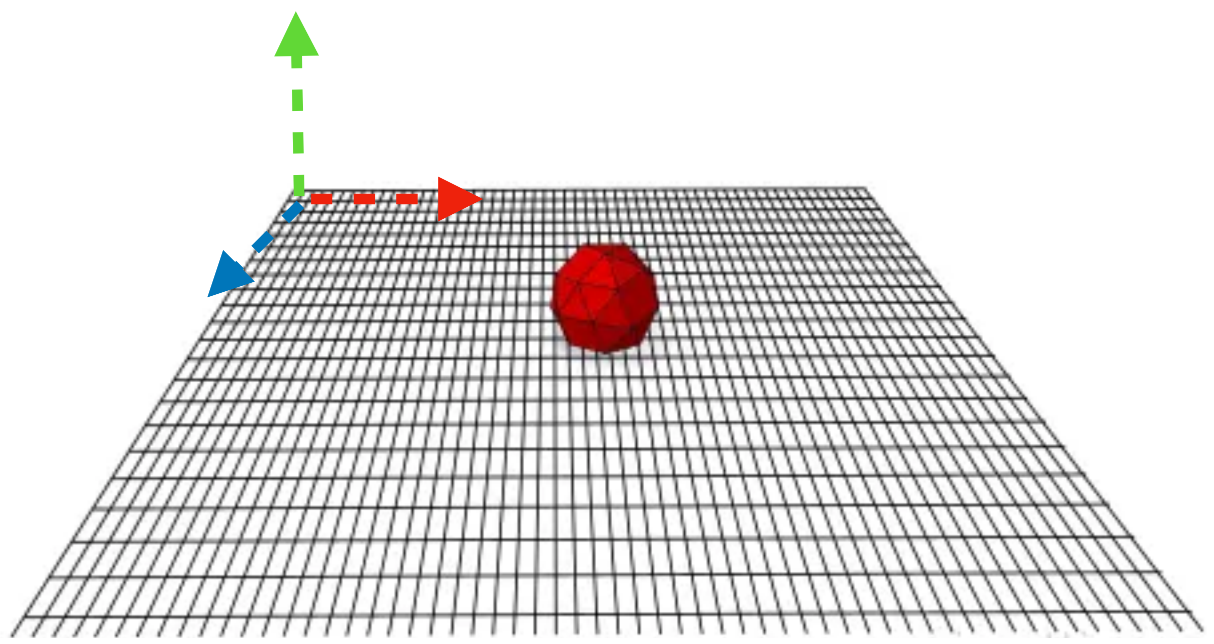


https://www.ime.unicamp.br/~marcia/AlgebraLinear/mudanca_base.html

Matriz da Câmera (View Matrix V)

No caso da View Matrix V, a nova base é formada pelos vetores U (up), R (right) e F (forward), que apontam para cima, direita e frente da câmera, respectivamente:



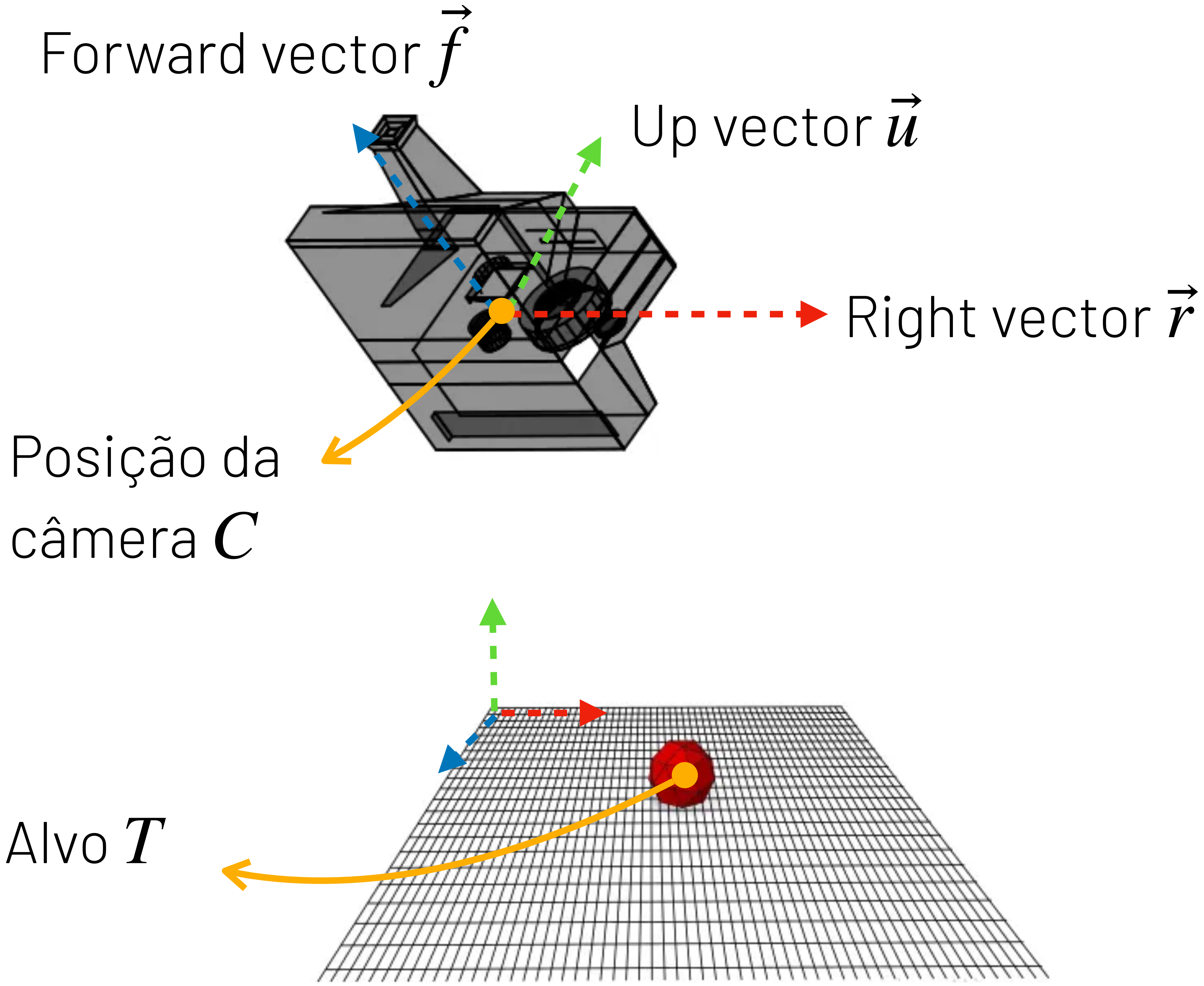


$$V = \begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ f_x & f_y & f_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{V = \begin{bmatrix} r_x & r_y & r_z & -r \cdot C \\ u_x & u_y & u_z & -u \cdot C \\ f_x & f_y & f_z & -f \cdot C \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

Matriz da Câmera (View Matrix V)

Tradicionalmente, a view matrix é criada por uma função chamada `CreateLookAt`, que calcula a base da câmera formada pelos vetores U (up), R (right) e F (forward):

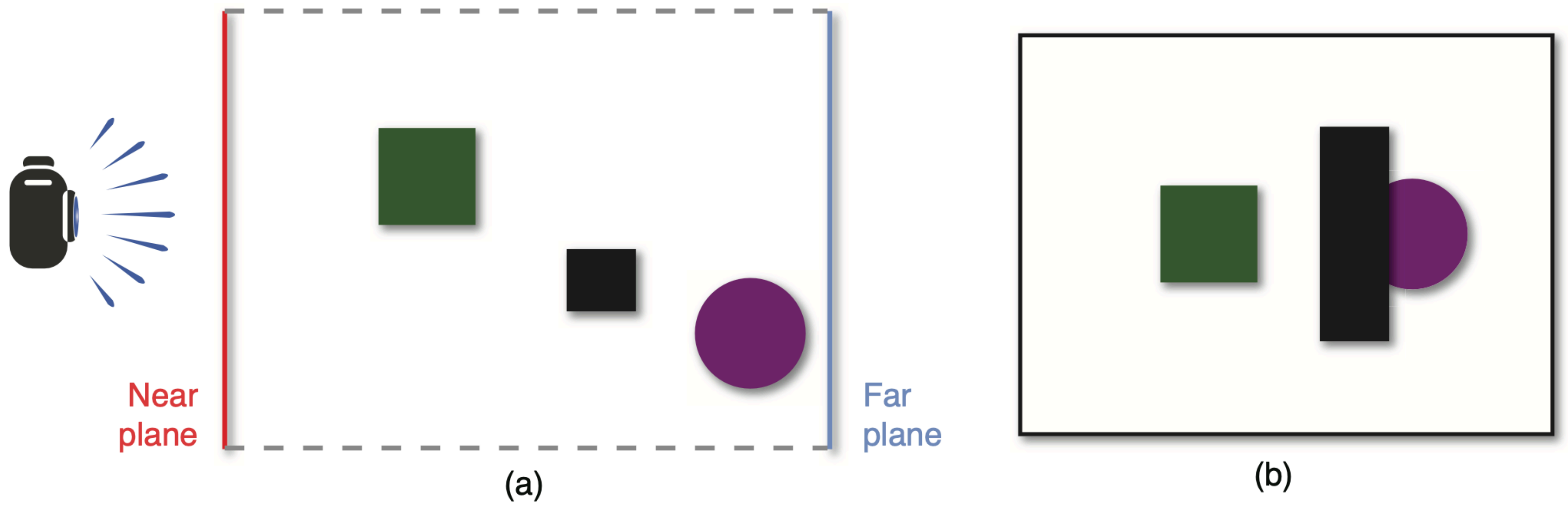



```
Matrix4 CreateLookAt(const Vector3& c, const Vector3& t
                     const Vector3& u)
{

   Vector3 f = Vector3::Normalize(c - t);
   Vector3 r = Vector3::Normalize(Vector3::Cross(u, f));
   u = Vector3::Normalize(Vector3::Cross(f, r));

   Vector3 d;
   d.x = Vector3::Dot(r, eye);
   d.y = Vector3::Dot(u, eye);
   d.z = Vector3::Dot(f, eye);

   float view[4][4] = {{r.x, r.x, r.x, -d.x},
                       {u.y, u.y, u.y, -d.y},
                       {f.z, f.z, f.z, -d.z},
                       {0.0f, 0.0f, 0.0f, 1.0f}};

    return Matrix4(view);
}
```

Projeções

Agora que que os vértices $v \in \mathbb{R}^4$ dos objetos estão no sistema de coordenadas da câmera, nos resta apenas projetá-los em pontos $p \in \mathbb{R}^2$ superfície 2D da tela:

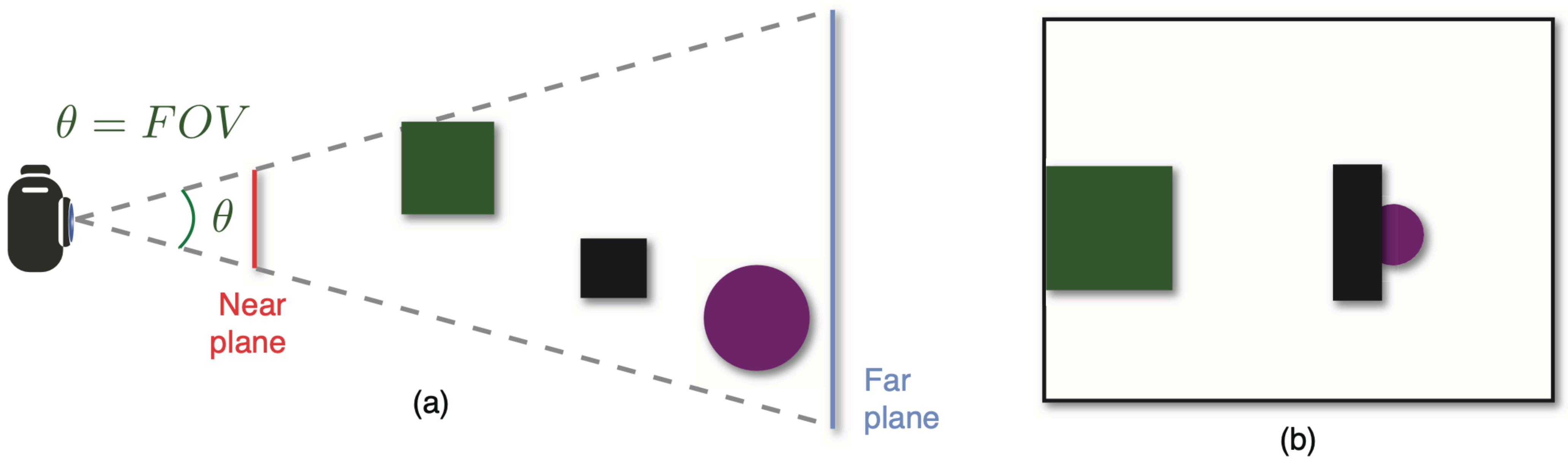
Projeção Ortográfica



- Tamanho dos objetos projetados **não** depende da distância da câmera

  Objetos mais distantes da câmera têm o mesmo tamanho do que os mais próximos

Projeção Perspectiva

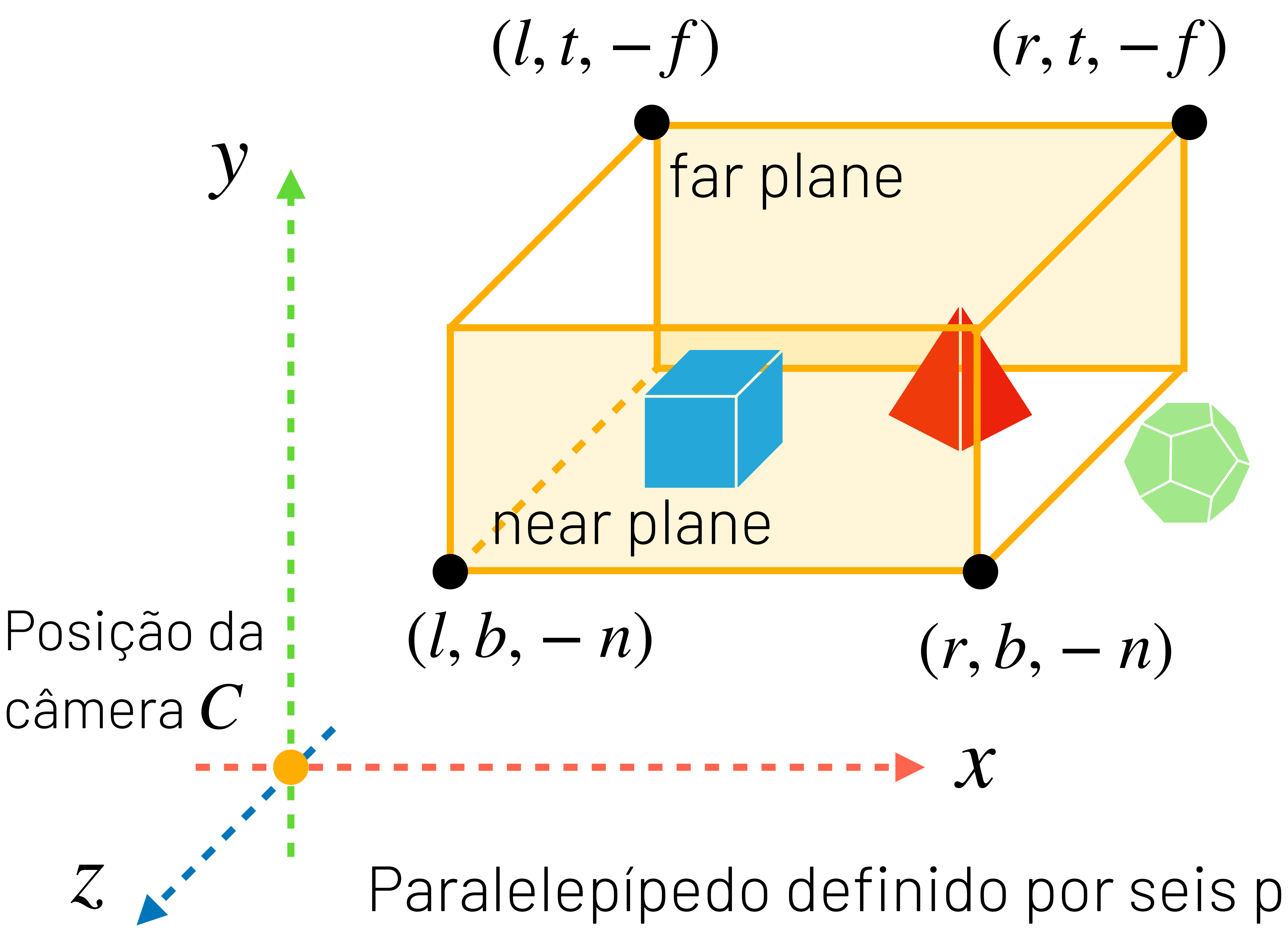


- Tamanho dos objetos projetados depende da distância da câmera

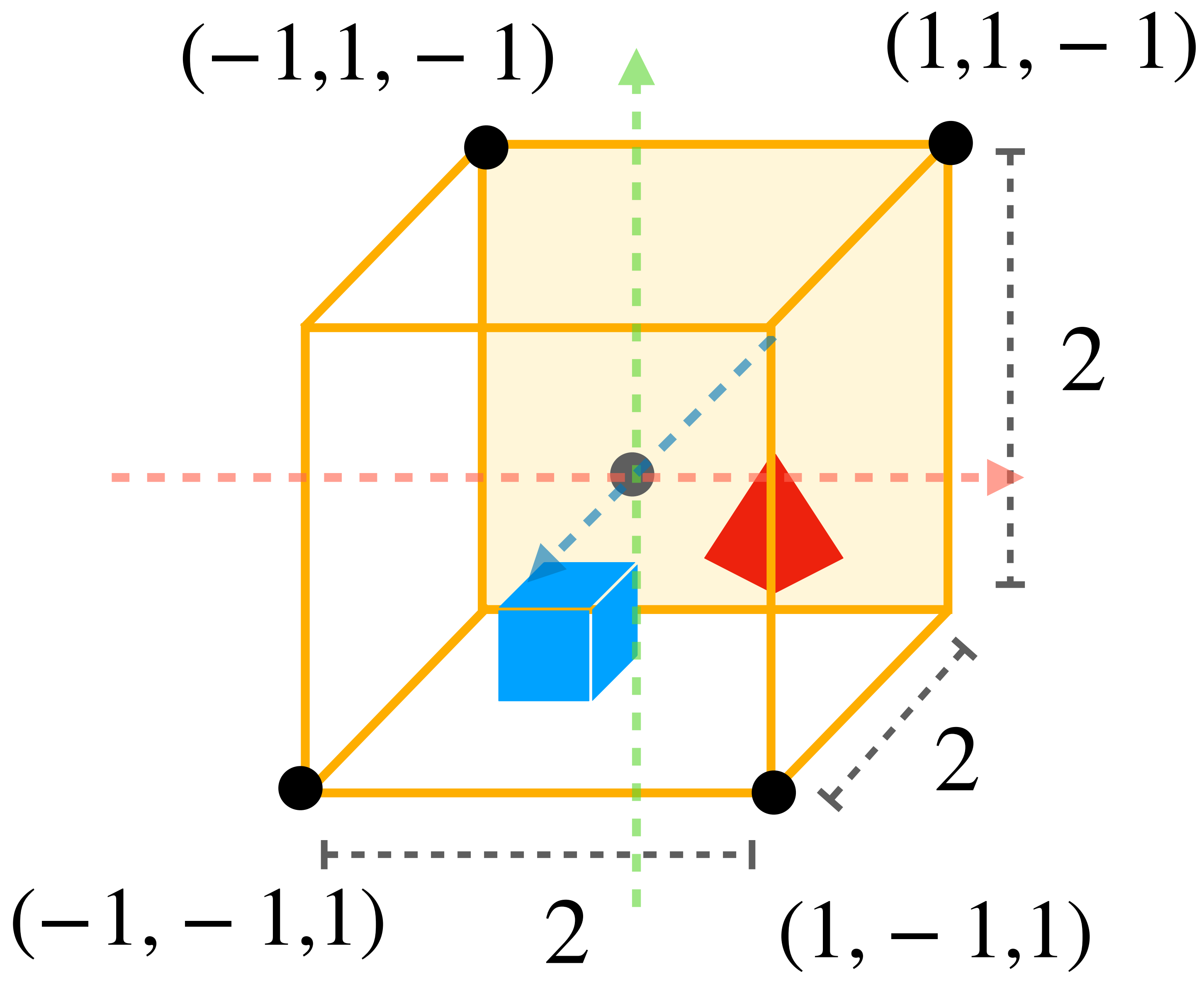
  Objetos mais distantes da câmera ficam menores do que os mais próximos.

Matriz de Projeção P: Ortográfica

A **Projeção Ortográfica** restringe a cena por um paralelepipedo na frente da câmera (origem) e mapeia esse volume para o *normalized device coordinates* (NDC) da OpenGL $[-1,1]^3$:



$$\begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix} \longrightarrow$$



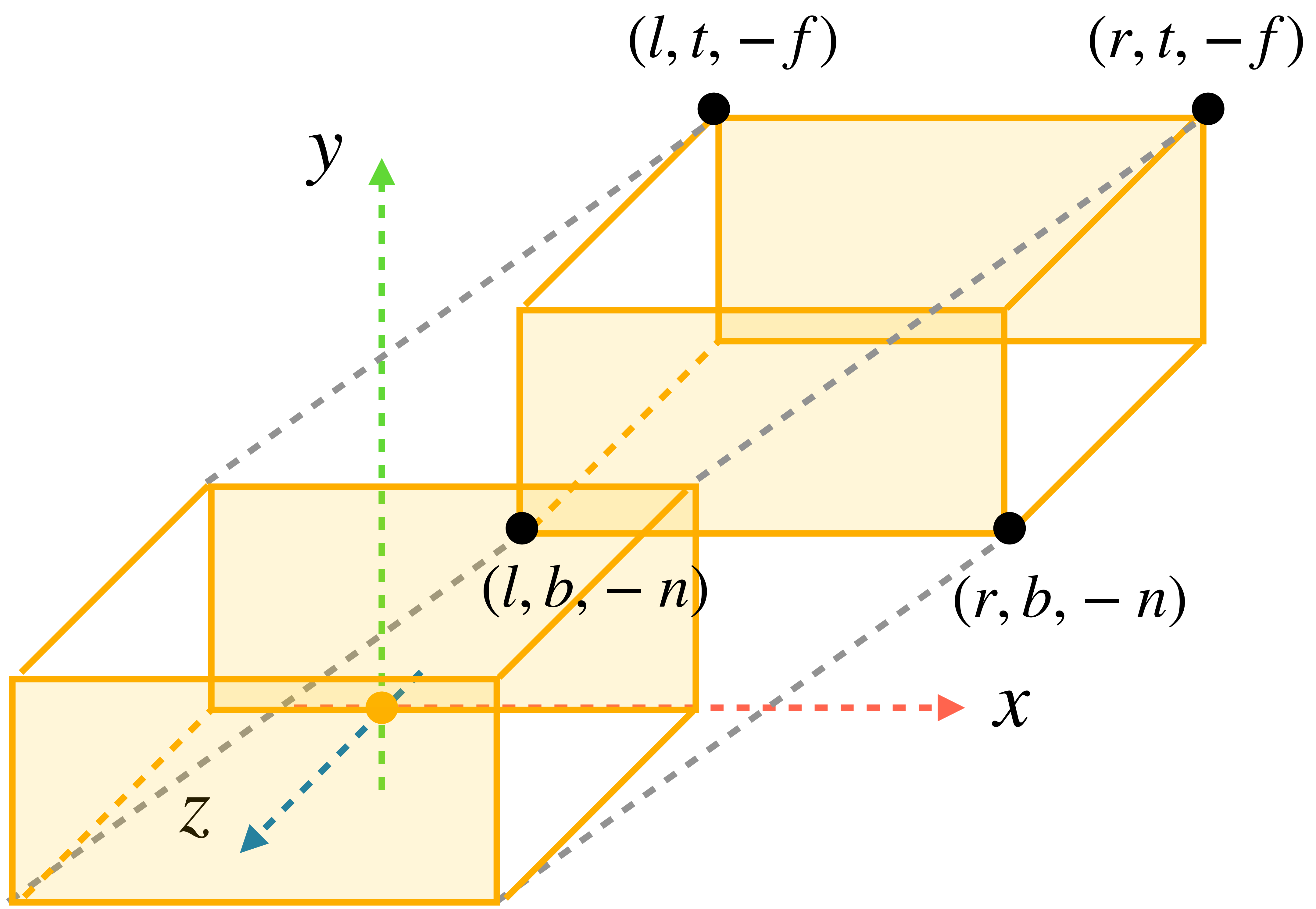
Paralelepípedo definido por seis pontos:

- left l, right r, bottom b e top t near n e far f

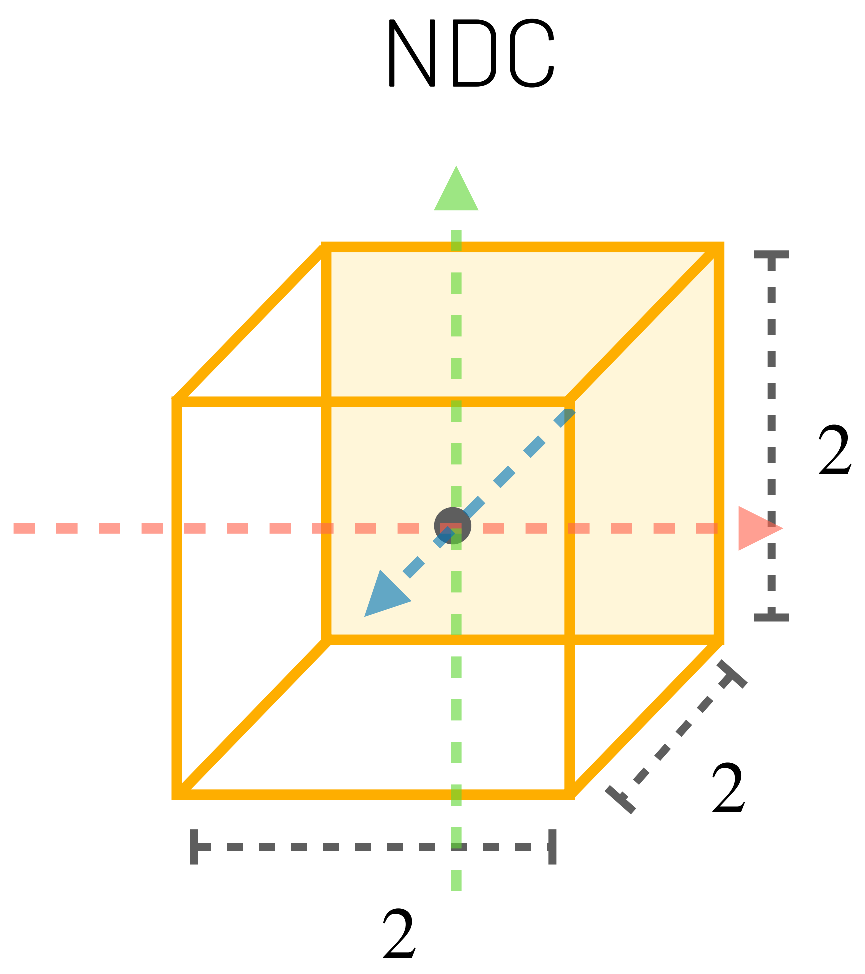
Normalized Device Coordinates (NDC)

Matriz de Projeção P: Ortográfica

A primeira parte da matriz de projeção ortográfica é uma transformação de translação $\mathbf{T}$ que move o centro do parlelepídedo para o centro do sistema de coordenadas da câmera:



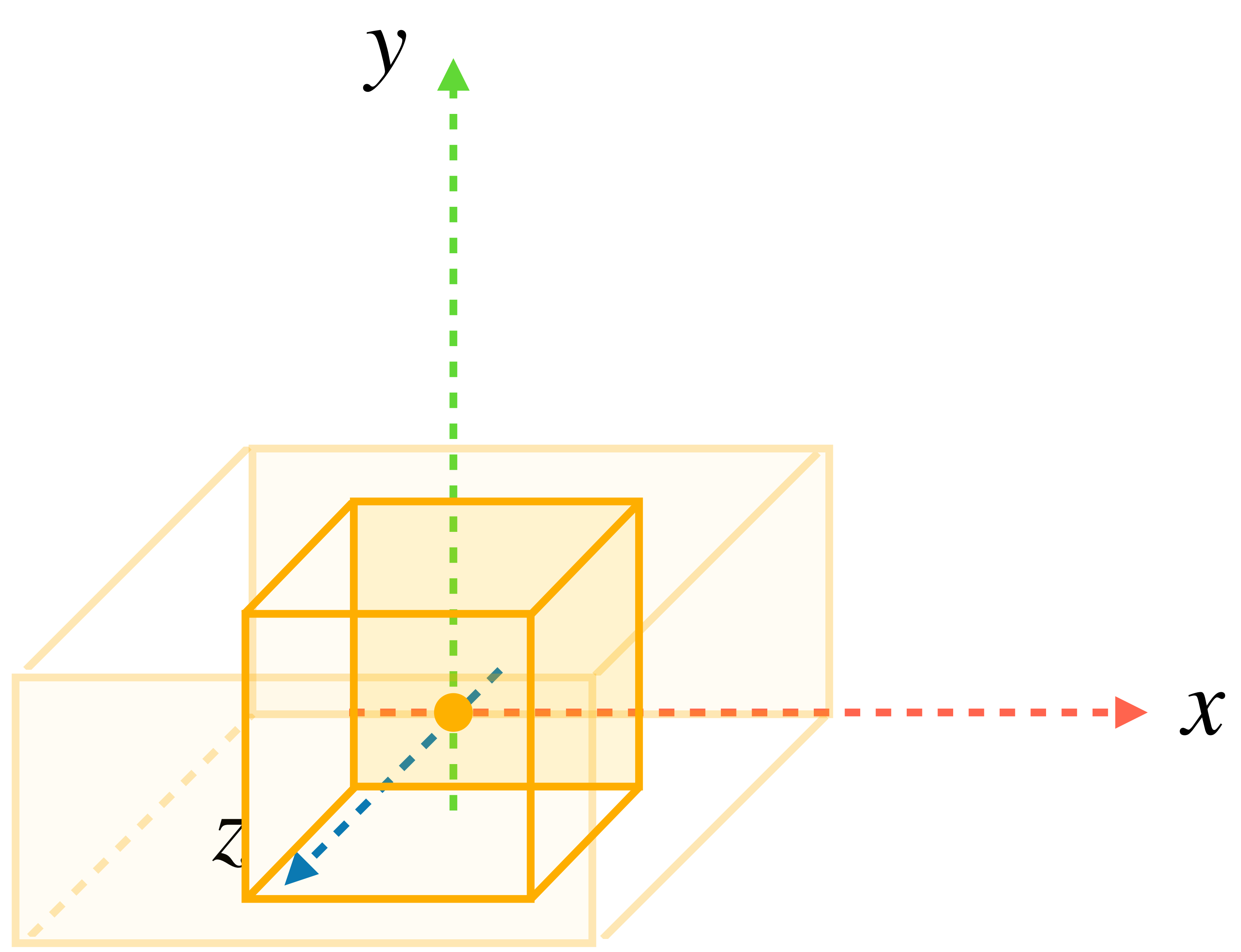
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- Translação $\mathbf{T}_x$: $-(l+r)/2$
- Translação $\mathbf{T}_y$: $-(b+t)/2$
- Translação $\mathbf{T}_z$: $-(f+n)/2$

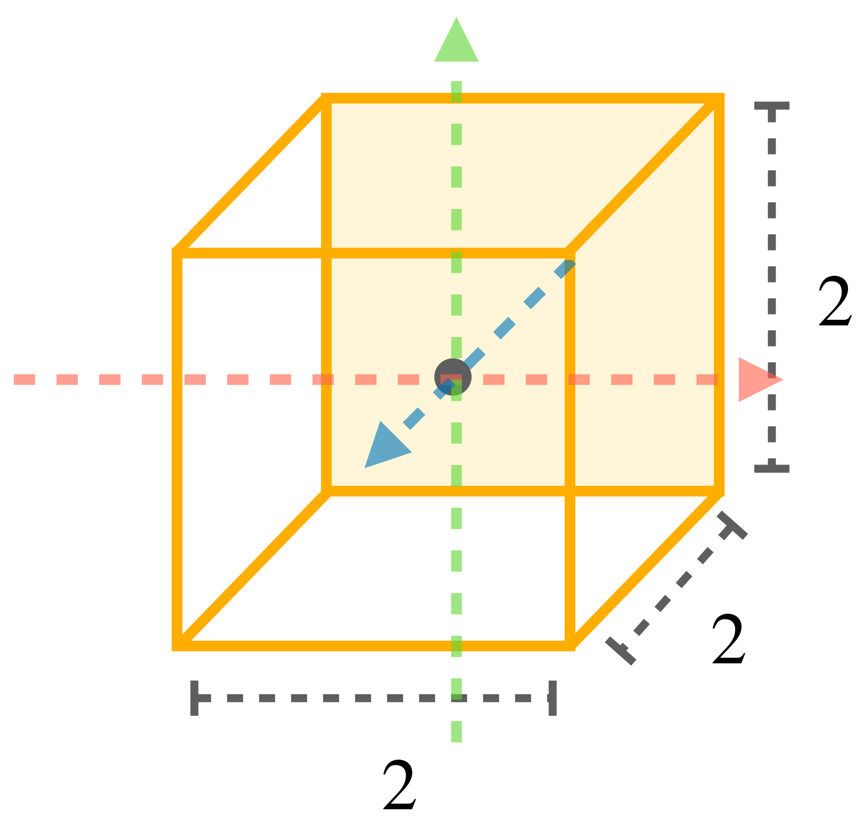
Matriz de Projeção P: Ortográfica

A segunda parte da matriz de projeção ortográfica é uma transformação de escala S para ajustar as dimensões do parapelípepo para as dimensões do cubo NDC $[-1,1]^3$



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

NDC



- Escala em x: $(r-l) \cdot \mathbf{S}_x = 2 \rightarrow \mathbf{S}_x = \dfrac{2}{r-l}$
- Escala em y: $(r-l) \cdot \mathbf{S}_y = 2 \rightarrow \mathbf{S}_y = \dfrac{2}{r-l}$
- Escala em z: $(r-l) \cdot \mathbf{S}_z = 2 \rightarrow \mathbf{S}_z = \dfrac{-2}{r-l}$

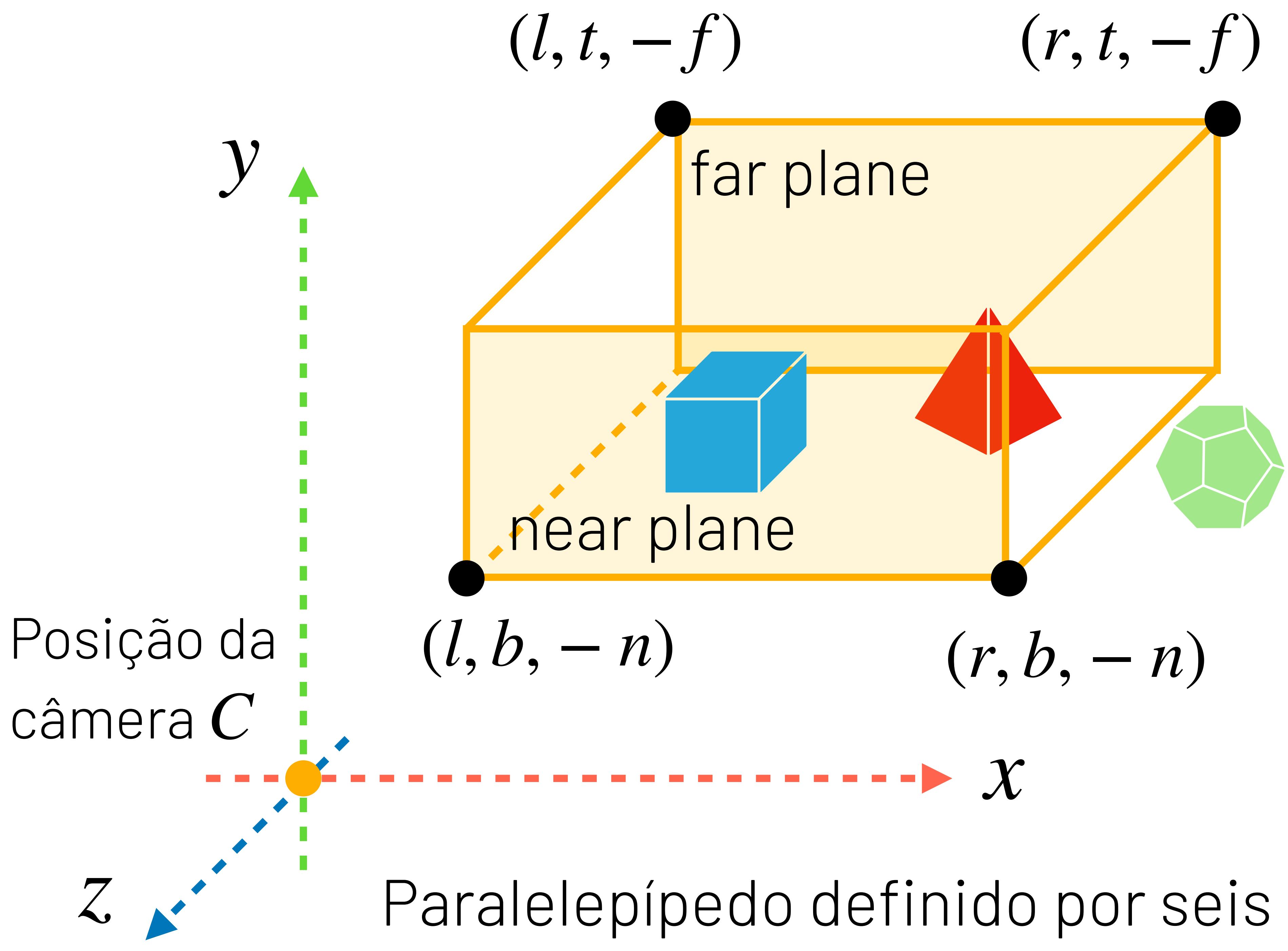
Matriz de Projeção P: Ortográfica

Combinando as matrizes de translação $\mathbf{T}$ e escala $\mathbf{S}$ com uma multiplicação, obtemos a seguinte matriz de projeção ortográfica $\mathbf{P}_{\text{ortho}} = \mathbf{S} \cdot \mathbf{T}$:

$$\underbrace{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{S} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T} = \underbrace{\boxed{\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{b+t}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{n+f}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}}}_{\mathbf{P}_{\text{ortho}}}$$

Matriz de Projeção P: Ortográfica

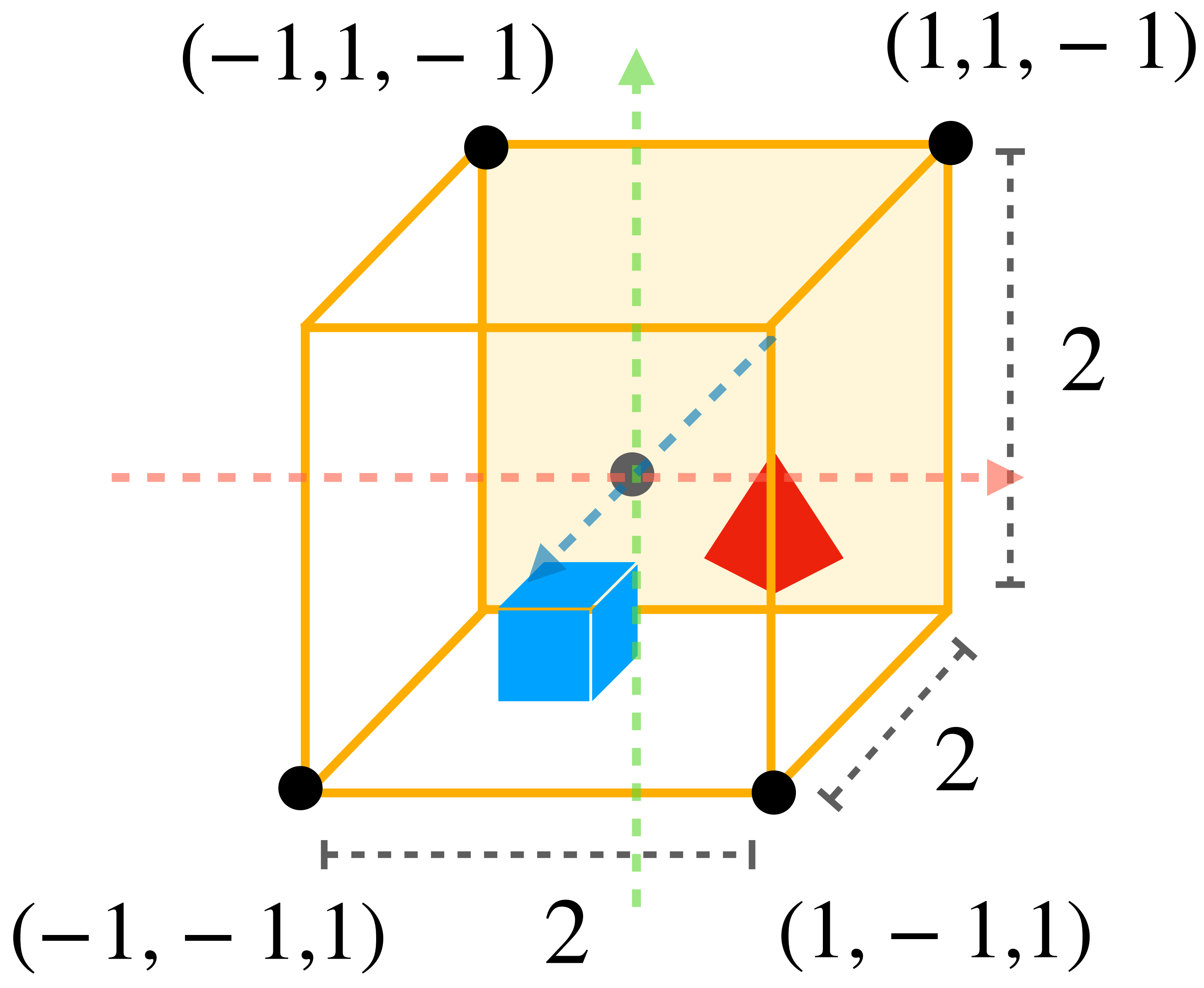
A **Projeção Ortográfica** restringe a cena por um paralelepipedo na frente da câmera (origem) e mapeia esse volume para o *normalized device coordinates* (NDC) da OpenGL $[-1,1]^3$:



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

$$\xrightarrow{\qquad\qquad}$$

$\mathbf{P}_{\text{ortho}}$



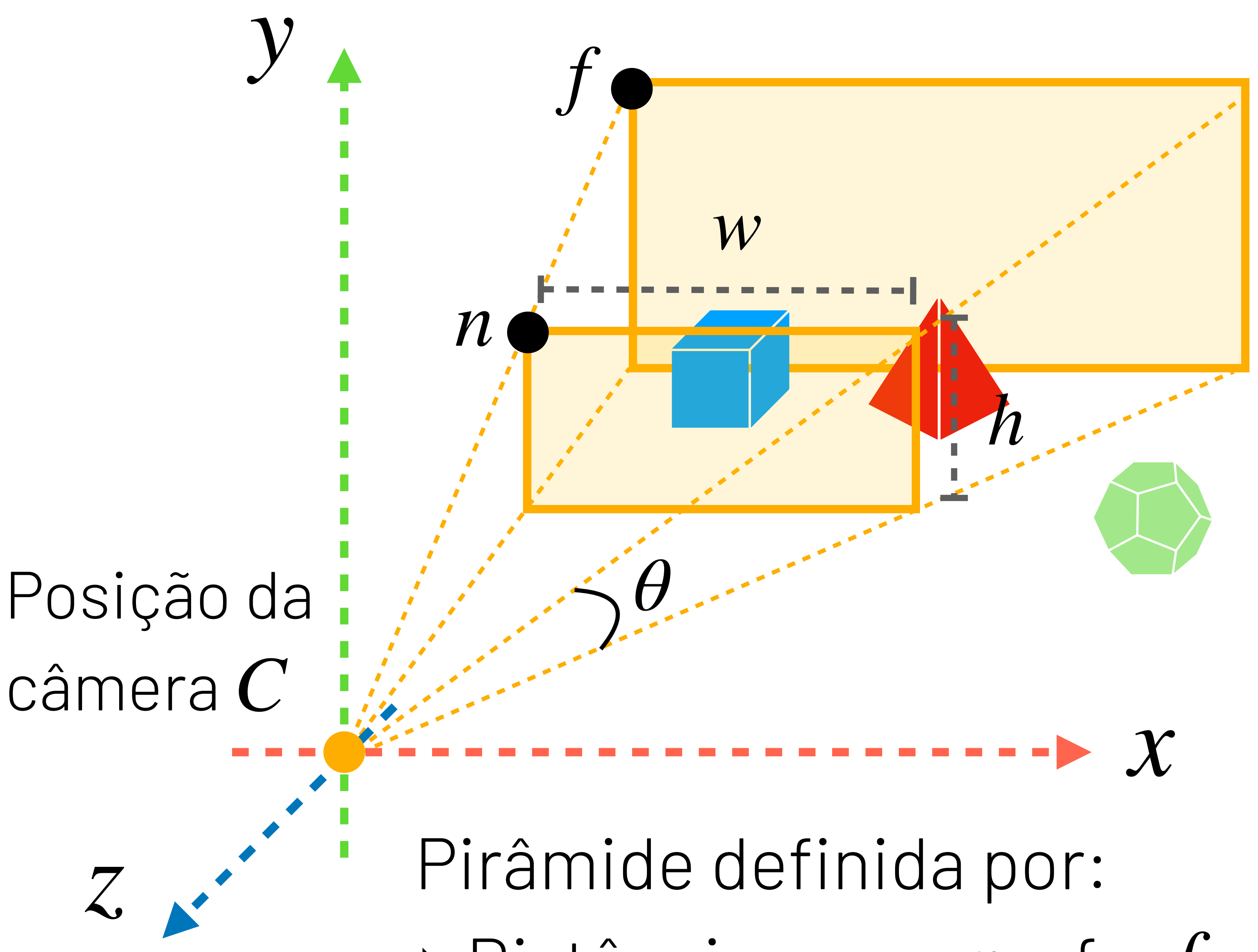
NDC

Paralelepípedo definido por seis pontos:

- left l, right r, bottom b e top t near n e far f

Matriz de Projeção P: Perspectiva

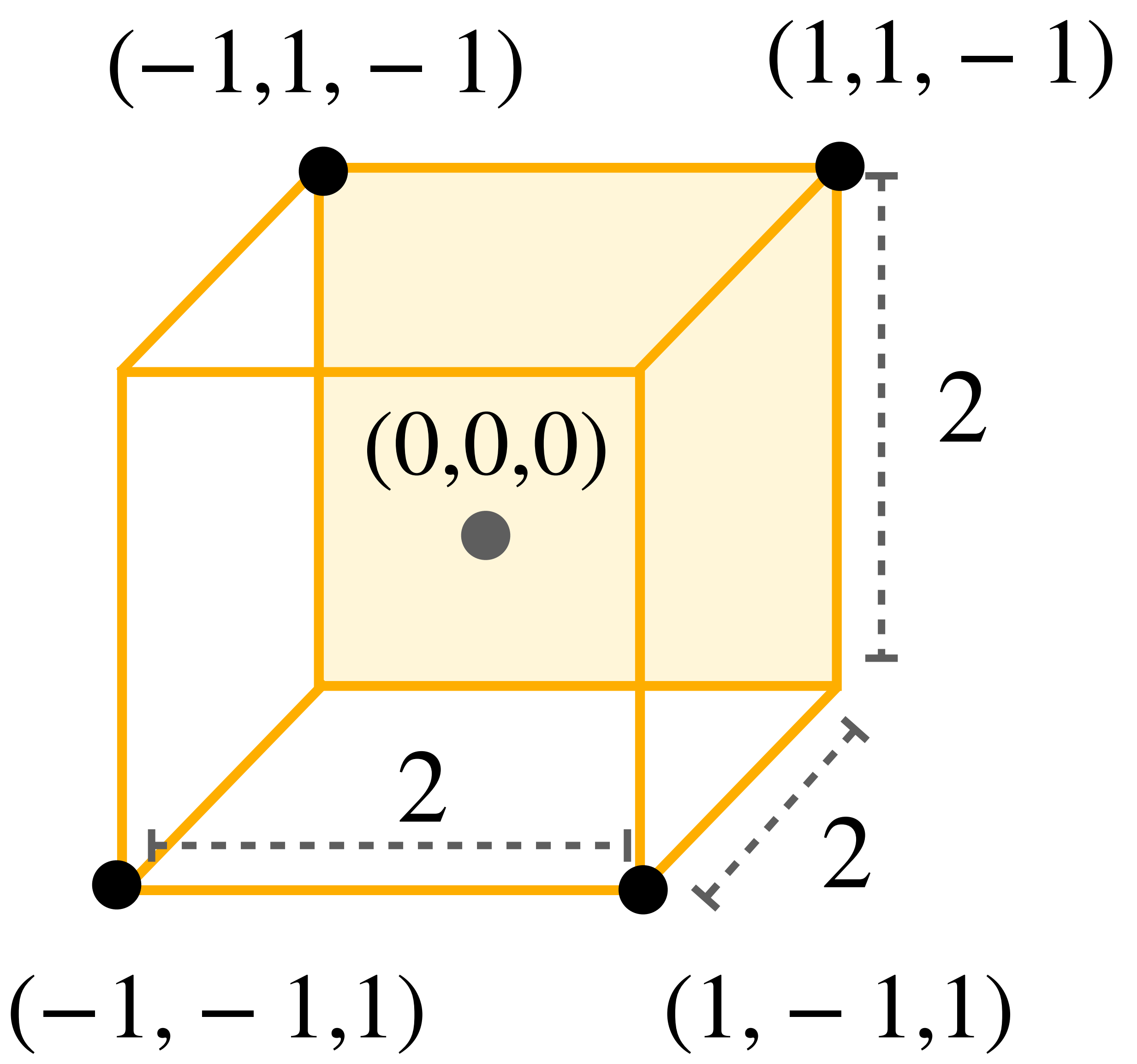
A **Projeção Perspectiva** restringe a cena por um tronco de pirâmide na frente da câmera e mapeia esse volume para o *normalized device coordinates* (NDC) da OpenGL $[-1,1]^3$:



Pirâmide definida por:

- Distâncias near n e far f
- Altura h e largura w
- Campo de visão θ no eixo y (field of view - fovY)

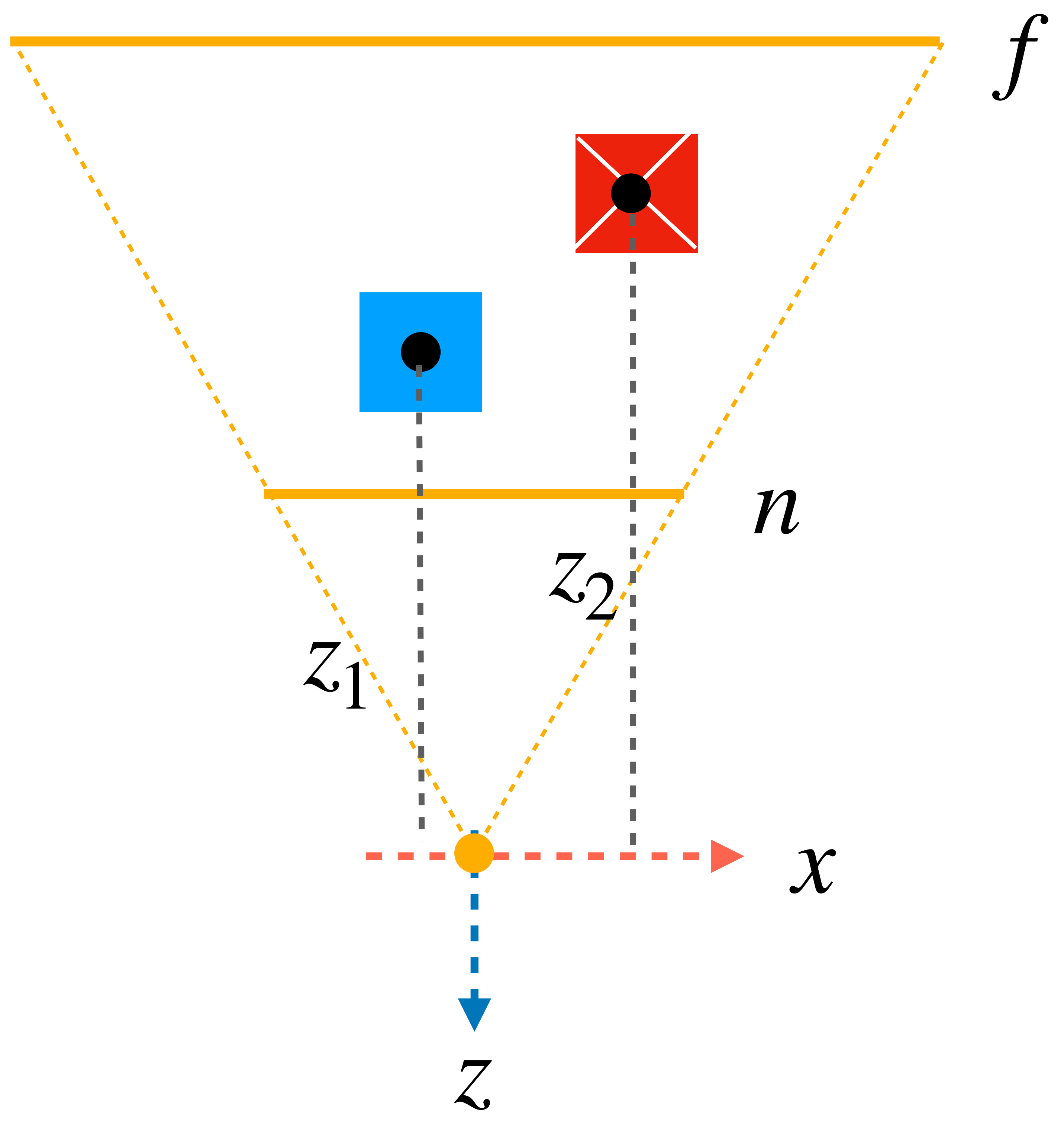
$$\begin{bmatrix} ? & 0 & 0 & ? \\ 0 & ? & 0 & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & 0 & ? \end{bmatrix} \longrightarrow$$



NDC

Matriz de Projeção P: Perspectiva

A primeira parte da matriz de projeção é fazer com que objetos mais distantes da câmera, ou seja, com valores maiores de z, pareçam menores.



Como a OpenGL irá realizar a divisão de perspectiva automaticamente, basta alterar a última linha da matriz de projeção para $\mathbf{P}_{\text{pers}}[3] = [0 \quad 0 \quad -1 \quad 0]$

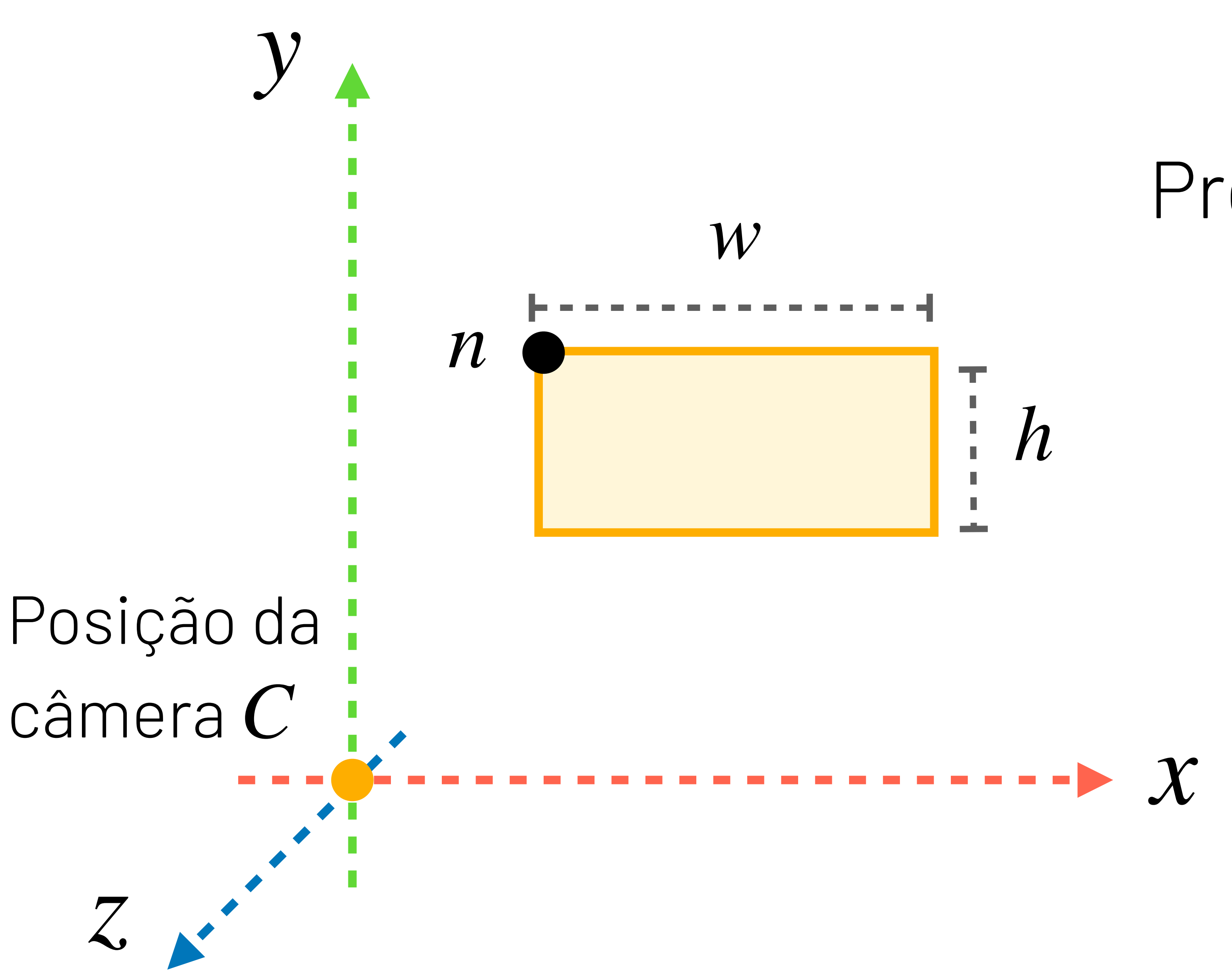
$$\begin{array}{r} \\ \\ \\ \mathbf{P}_{\text{pers}}[3] \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z \end{bmatrix}$$

Divisão Perspectiva irá dividir as coordenadas por $-z$:

$$\frac{x}{-z}, \frac{y}{-z}, \frac{z}{-z}$$

Matriz de Projeção P: Perspectiva

A segunda parte da matriz de projeção é encontrar os fatores de escala horizontais S_x para acomodar a proporção da tela $a = h/w$:



Proporção da tela:

$$a = \frac{h}{w}$$

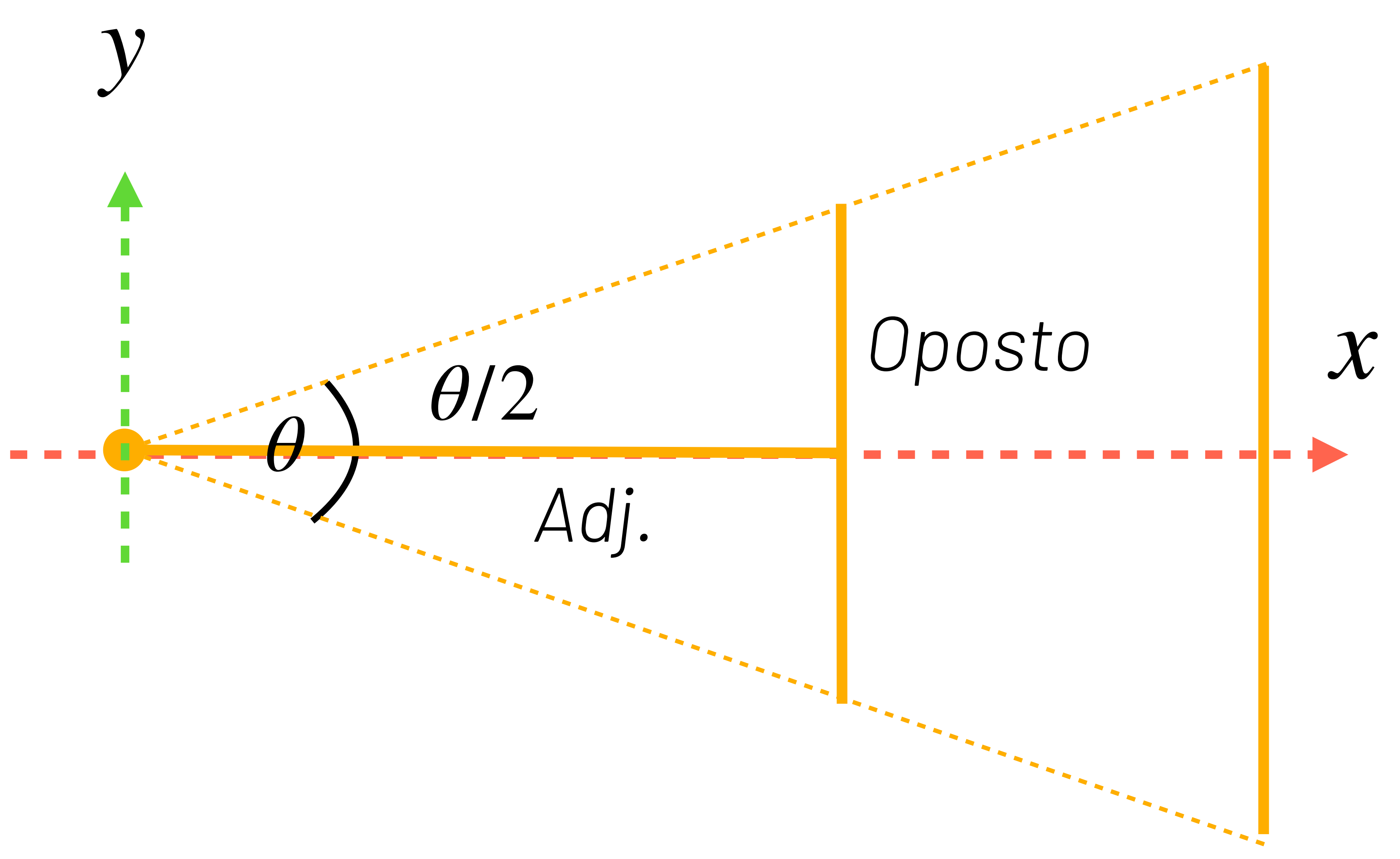
$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Escala em x: $\mathbf{S}_x = a$

Como a proporção da tela a foi definida com a altura h no numerador, só precisamos "corrigir" o eixo x

Matriz de Projeção P: Perspectiva

A terceira parte da matriz de projeção é encontrar os fatores de escala horizontais S_x e verticais S_y em função do ângulo θ, de tal forma que, **quanto maior θ, menor os objetos**:



Fator inversamente proporcional a θ :

$$\frac{1}{\text{op./adj.}} = \frac{1}{tan(\theta/2)}$$

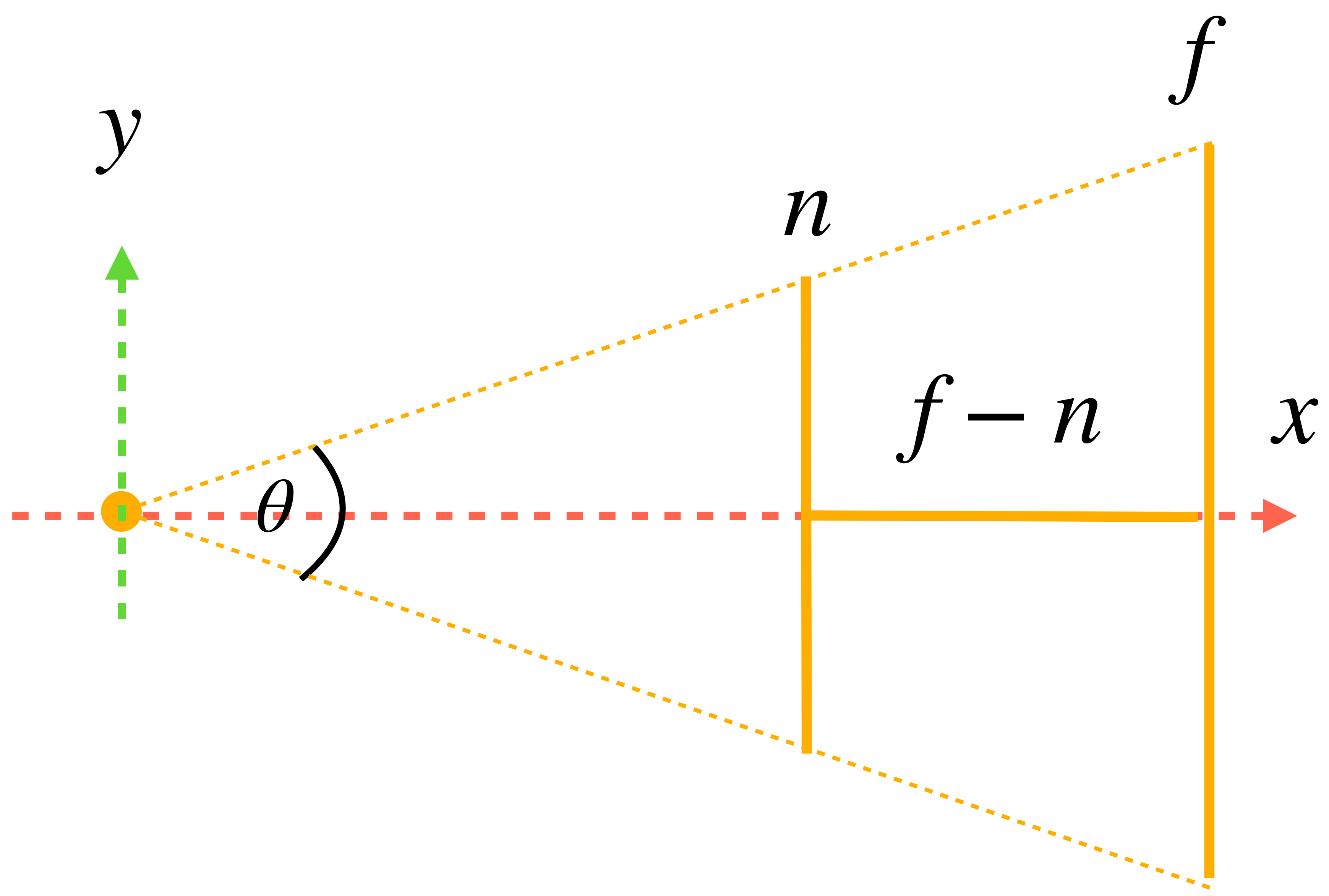
$$\begin{bmatrix} \frac{1}{tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Escala em x: $\mathbf{S}_x = 1/tan(\theta/2)$
- Escala em y: $\mathbf{S}_y = 1/tan(\theta/2)$

Para isso, podemos simplesmente escalar x e y pelo inverso da razão entre oposto/adjacente

Matriz de Projeção P: Perspectiva

A última parte é **normalizar** os valores de z no intervalo $[-1,1]$ com uma escala S_z e uma translação $\mathbf{T}_{\mathbf{z}'}$, assim como fizemos na projeção ortográfica:



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Escala em z: $\mathbf{S}_z = \dfrac{f}{f-n}$

- Translação em z: $\mathbf{T}_z = -n$

Matriz de Projeção P: Perspectiva

Combinando as matrizes de translação $\mathbf{T}$ e escala $\mathbf{S}$ com uma multiplicação e alterando a última linha para divisão de perspectiva, obtemos a seguinte matriz de projeção perspectiva:

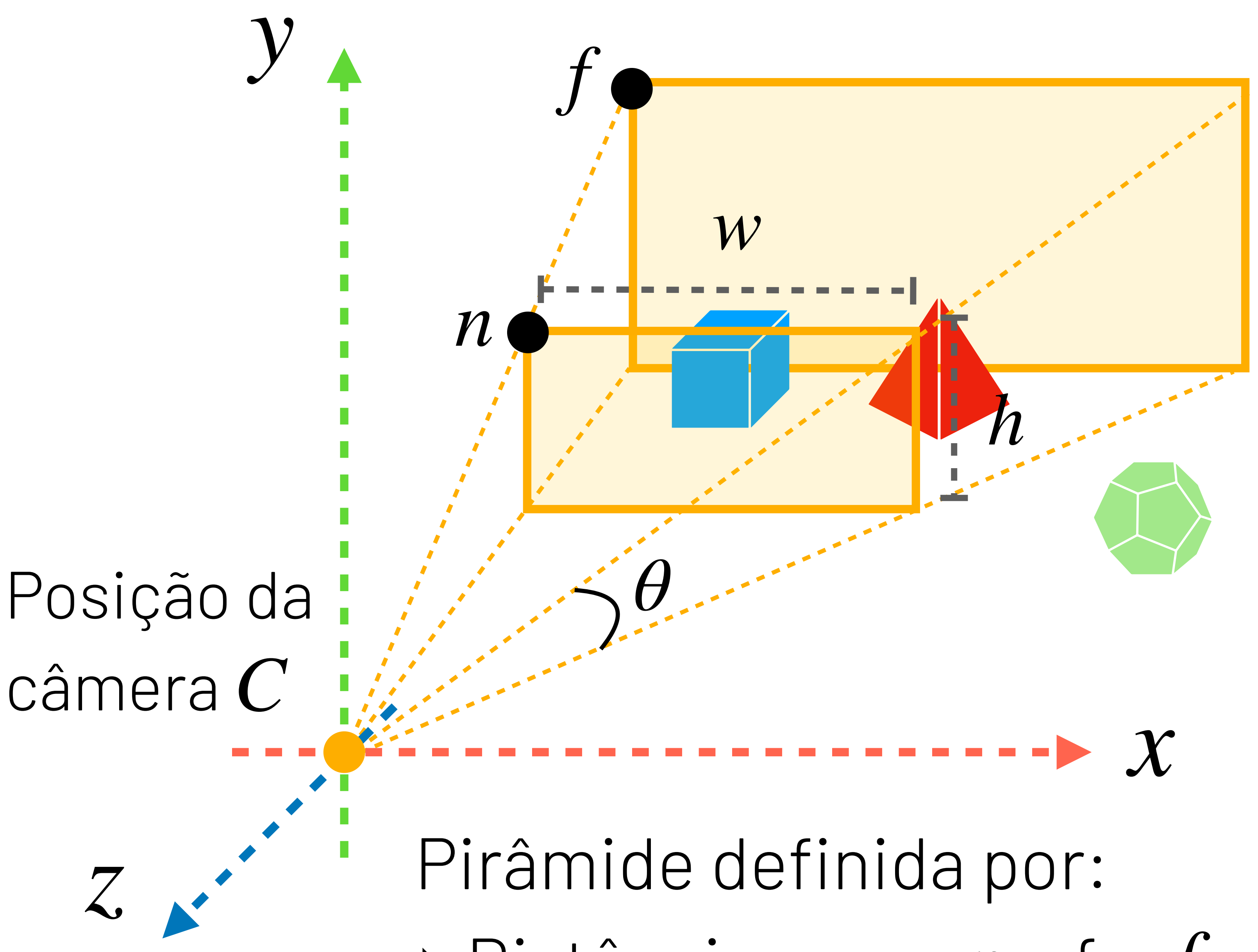
$$\begin{bmatrix} \frac{1}{a} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -n \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \frac{1}{a \cdot tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-nf}{f-n} \\ \boxed{0 \quad 0 \quad 0 \quad 1} \end{bmatrix} = \boxed{\begin{bmatrix} \frac{1}{a \cdot tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}} \quad \mathbf{P}_{\text{persp}}$$

Alterar última linha para divisão de perspectiva

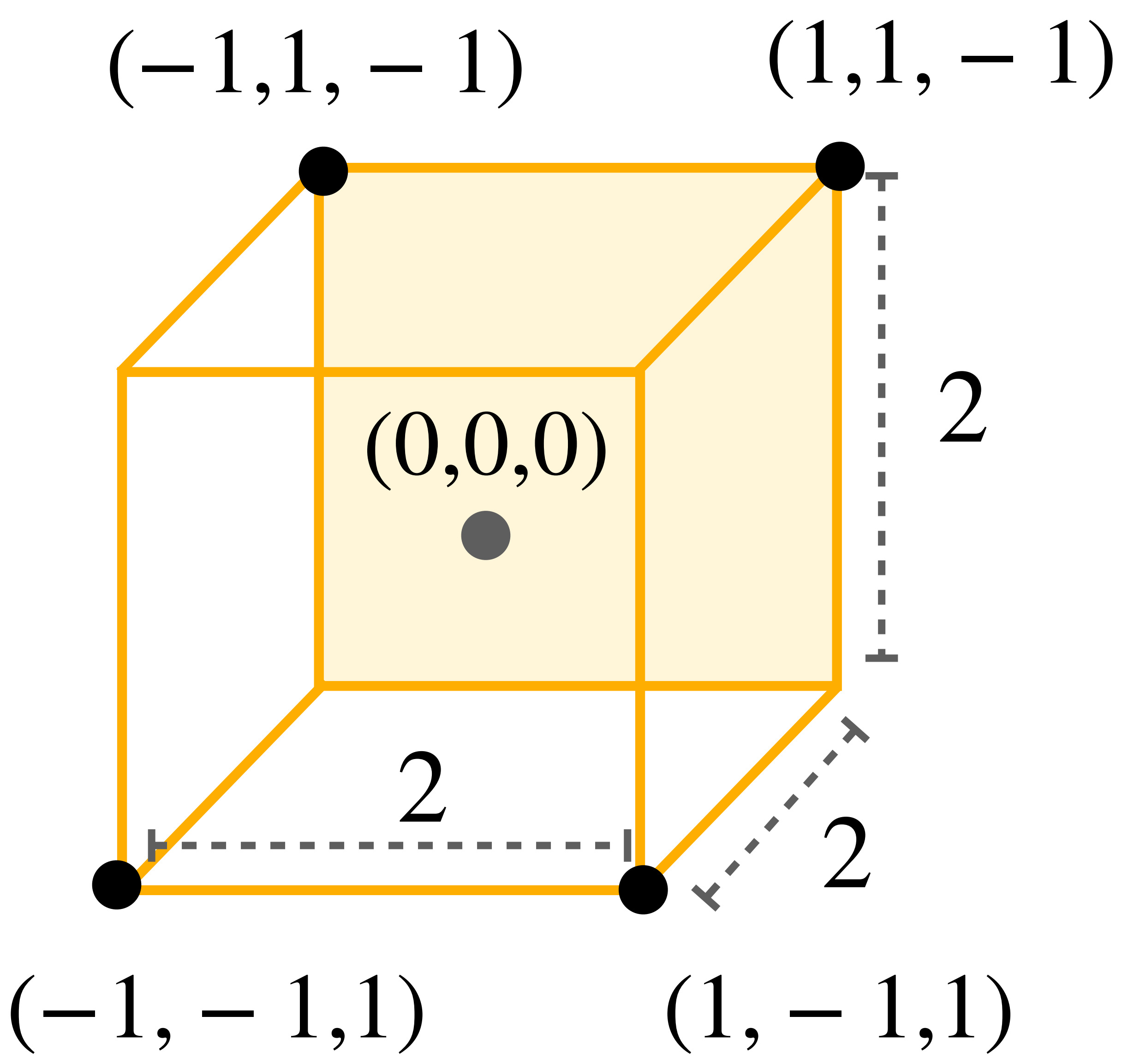
Matriz de Projeção P: Perspectiva

A **Projeção Perspectiva** restringe a cena por um tronco de pirâmide na frente da câmera e mapeia esse volume para o *normalized device coordinates* (NDC) da OpenGL $[-1,1]^3$:



$$\begin{bmatrix} \frac{1}{a \cdot tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta/2)} & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\xrightarrow[\mathbf{P}_{\text{persp}}]{}$$



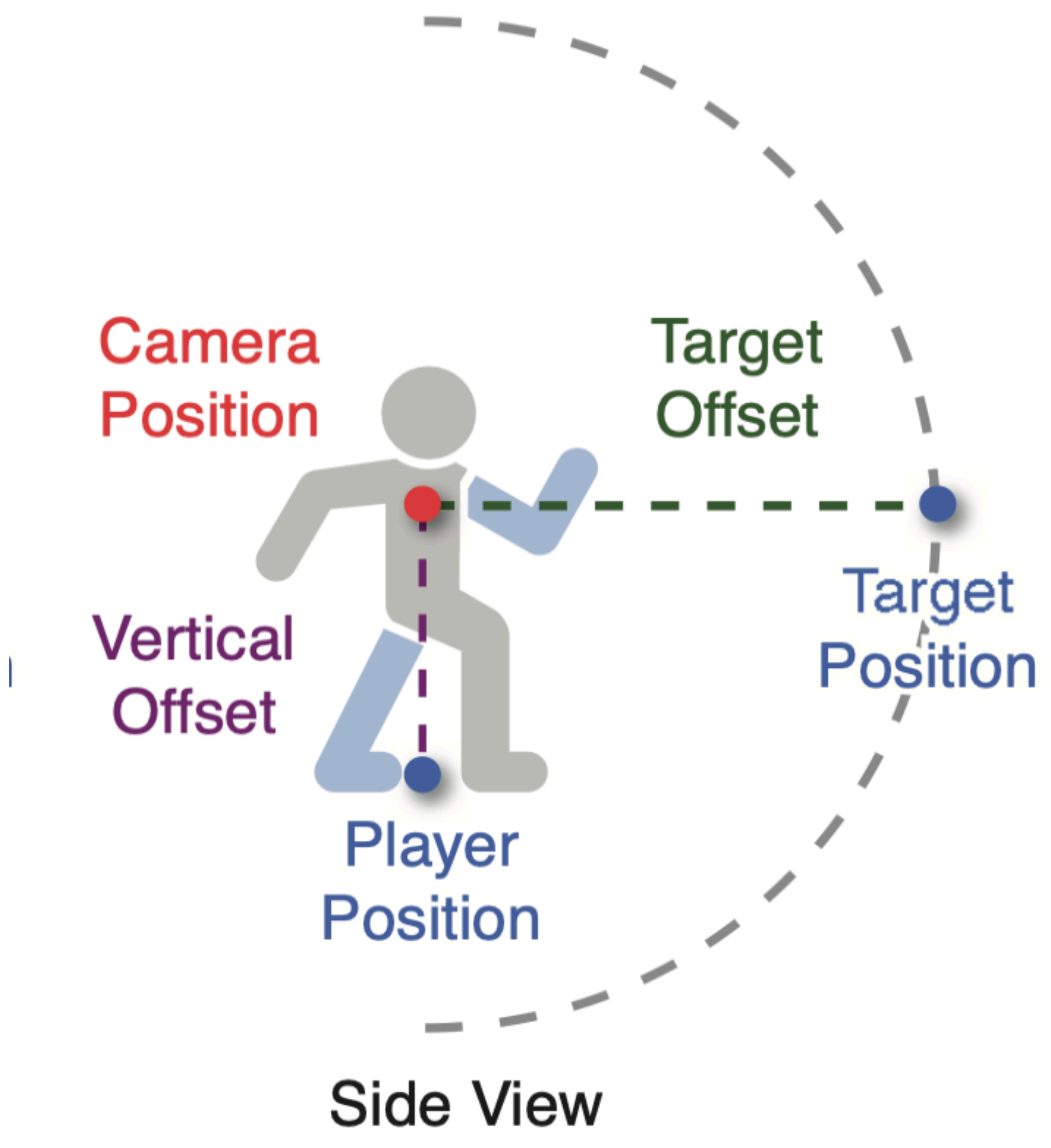
NDC

Pirâmide definida por:

- Distâncias near n e far f
- Altura h e largura w
- Campo de visão θ no eixo y (field of view - fovY)

Câmera em Primeira Pessoa

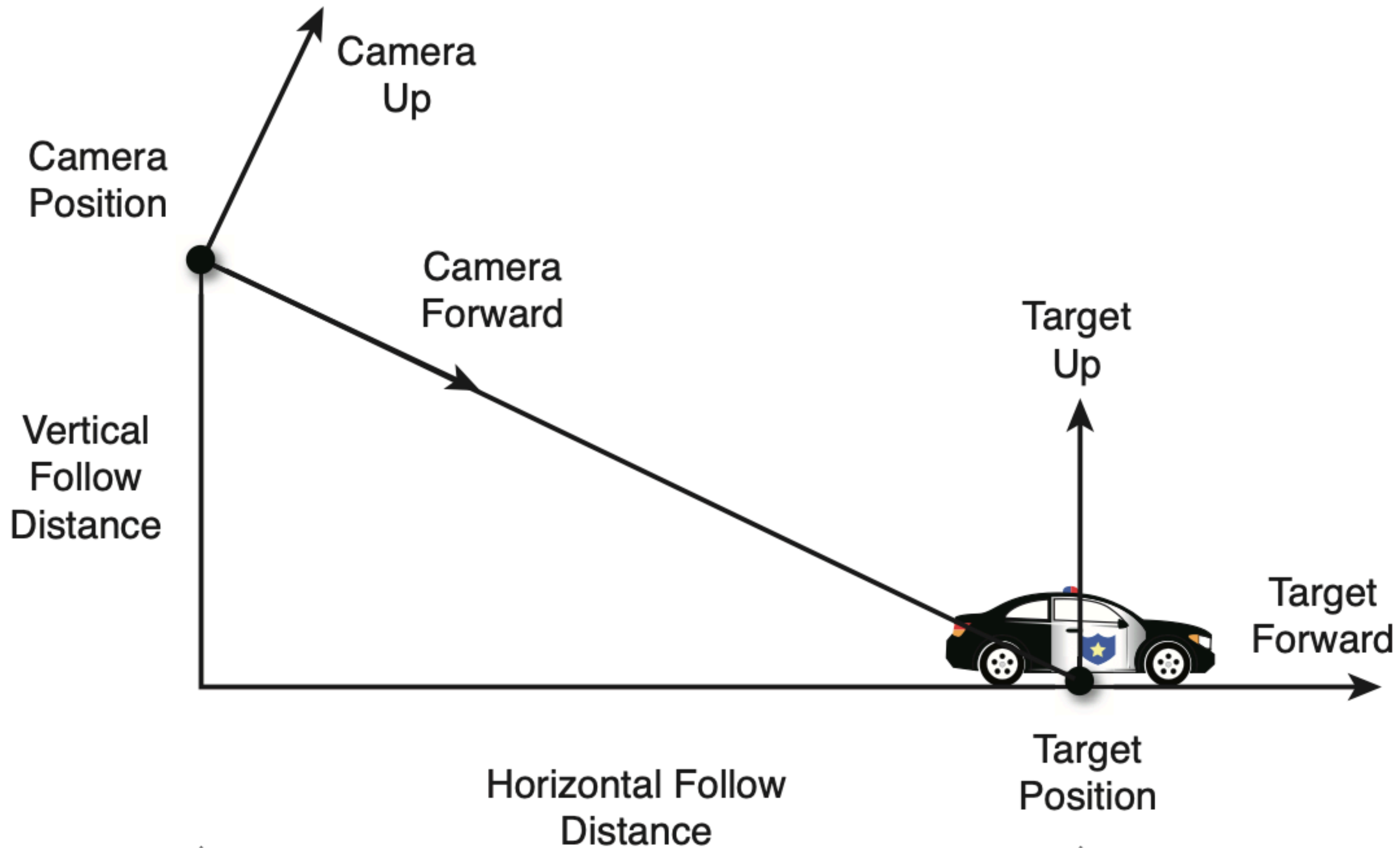
Em uma **Câmera em primeira pessoa**, a posição da câmera é sempre um deslocamento vertical definido da posição do jogador:




```
void FPSCamera::Update(Actor *player, float tOffset)
{
   // A posição da câmera (c) é a posição do jogador
   Vector3 c = player->GetPosition();

   // Posição alvo (t) na frente do jogador
   Vector3 t = c + player->GetForward() * tOffset;

   // Up é um vetor unitário no eixo Y
   Vector3 up = Vector3::UnitY;

   // Cria view matrix com os parâmetros acima
   Matrix4 view = Matrix4::CreateLookAt(c, t, up);
   SetViewMatrix(view);
}
```

Câmera em Terceira Pessoa

A **Câmera em terceira pessoa** segue um ponto à frente (tDist) de um objeto alvo (`tPos`) com uma distância de acompanhamento vertical (`vDist`) e horizontal (`hDist`) predefinida.




```
void FollowCamera::Update(Actor *player, float hDist,
float vDist, float tDist)
{
   // A posição da câmera (c) atrás do jogador
   Vector3 c = player->GetPosition();
   c -= player->GetForward() * hDist;
   c -= Vector3::UnityY * hVist;

  // A posição do alvo (t) a frente do jogador
  Vector3 t = player->GetPosition() +
              player->GetForward() * tDist;

   // Up é um vetor unitário no eixo Y
  Vector3 up = Vector3::UnityY;

   // (Up is just UnitZ since we don't flip the camera)
  Matrix4 view = Matrix4::CreateLookAt(c, t, up);
  SetViewMatrix(view);
}
```

Próxima aula

A19: Iluminação

- Iluminação 2D
  - Light Mapping
- Flat Shading
- Gouraud Shading
- Phong Shading