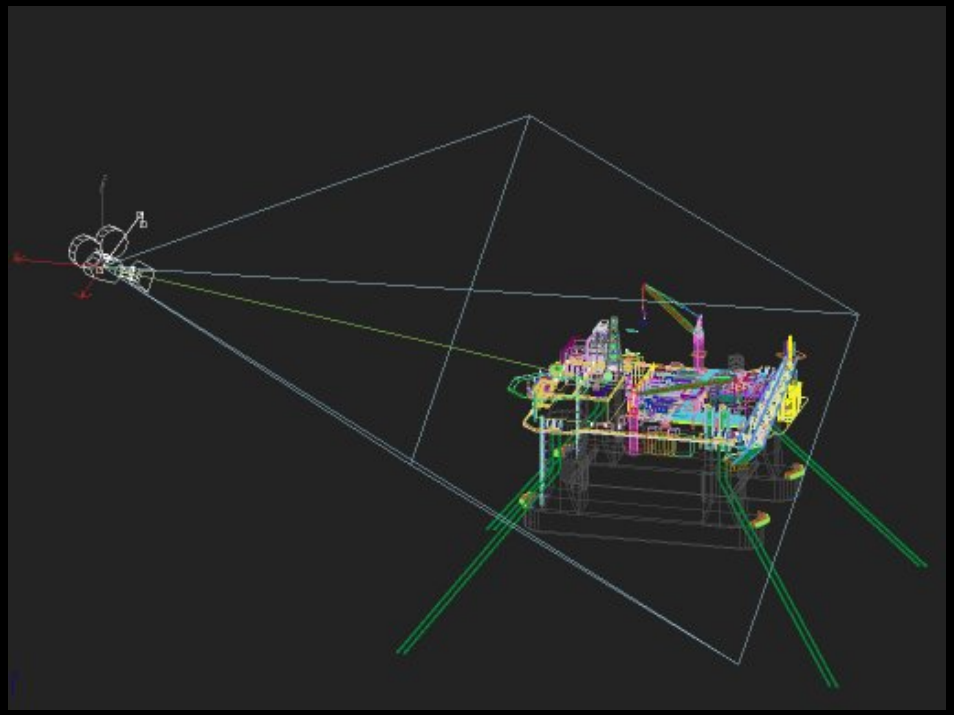
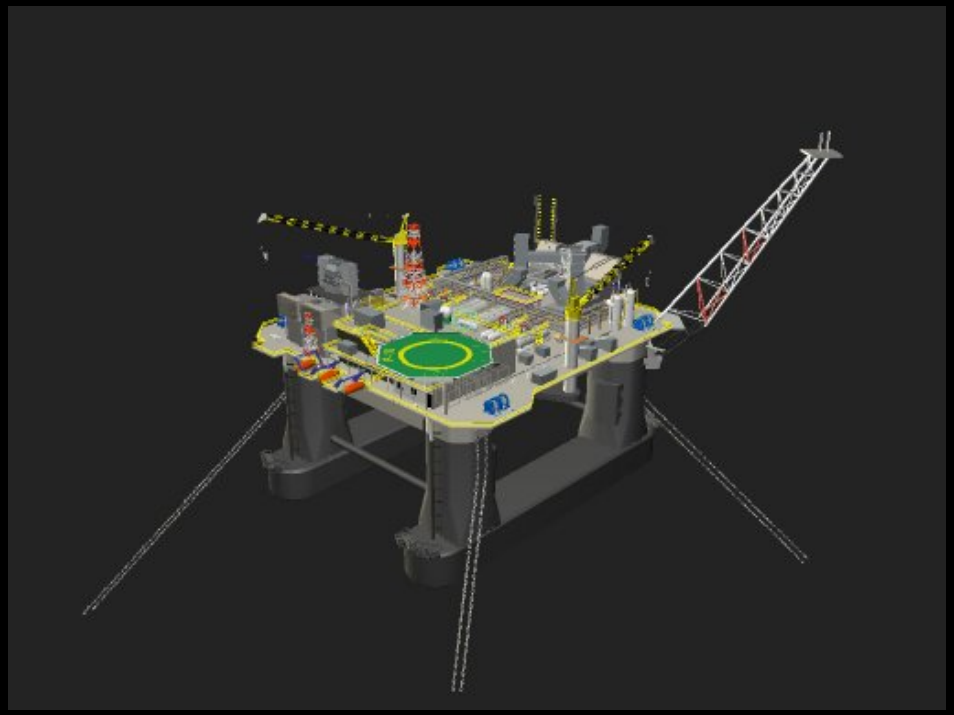
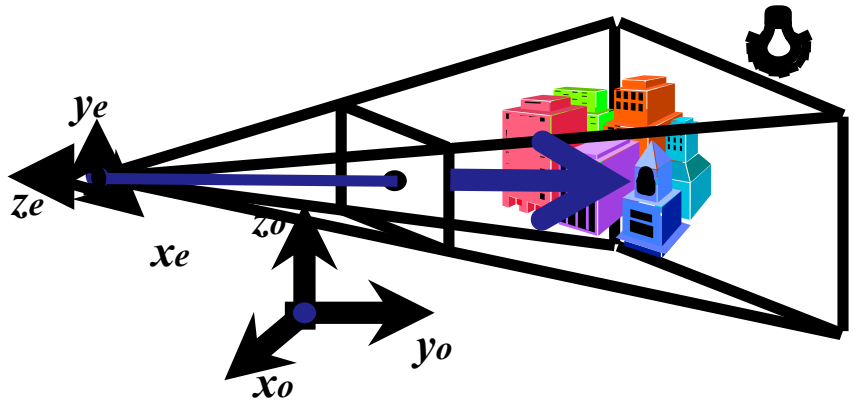
ZBuffer
Rendering com Mapa de Profundidade
(*OpenGL Rendering*)





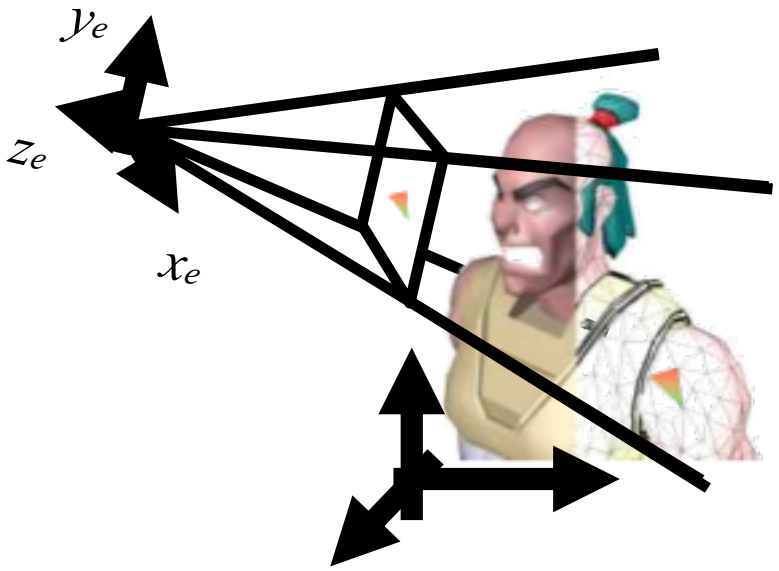
Complexidade do Algoritmo de Rastreamento de Raios

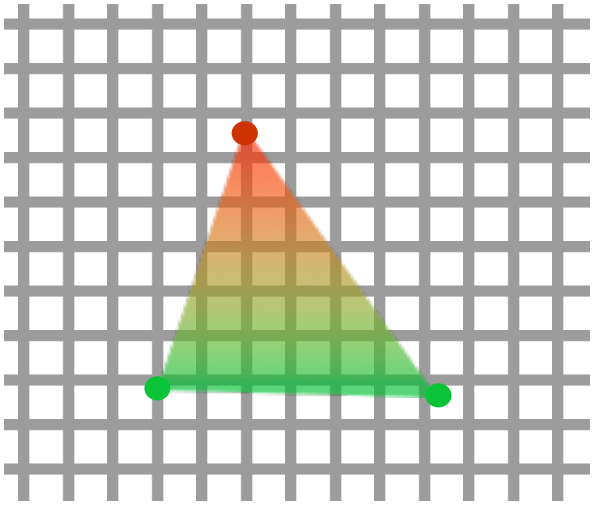


Para cada pixel da tela
Defina um raio
Para cada objeto da cena
Calcule o objeto visível
Para cada luz da cena
Lance um raio
Para cada objeto da cena
Teste se o objeto faz sombra
Calcule sua iluminação

Complexidade maior que $\boldsymbol{O}(\text{num. de pixels} \times \text{num. de objetos}^2)$

Mapa de Profundidade ou ZBuffer





Calcule a cor de cada vértice através da equação de iluminação

Projete cada vértice

Transforme o triângulo em fragmentos correspondentes a cada pixel encoberto

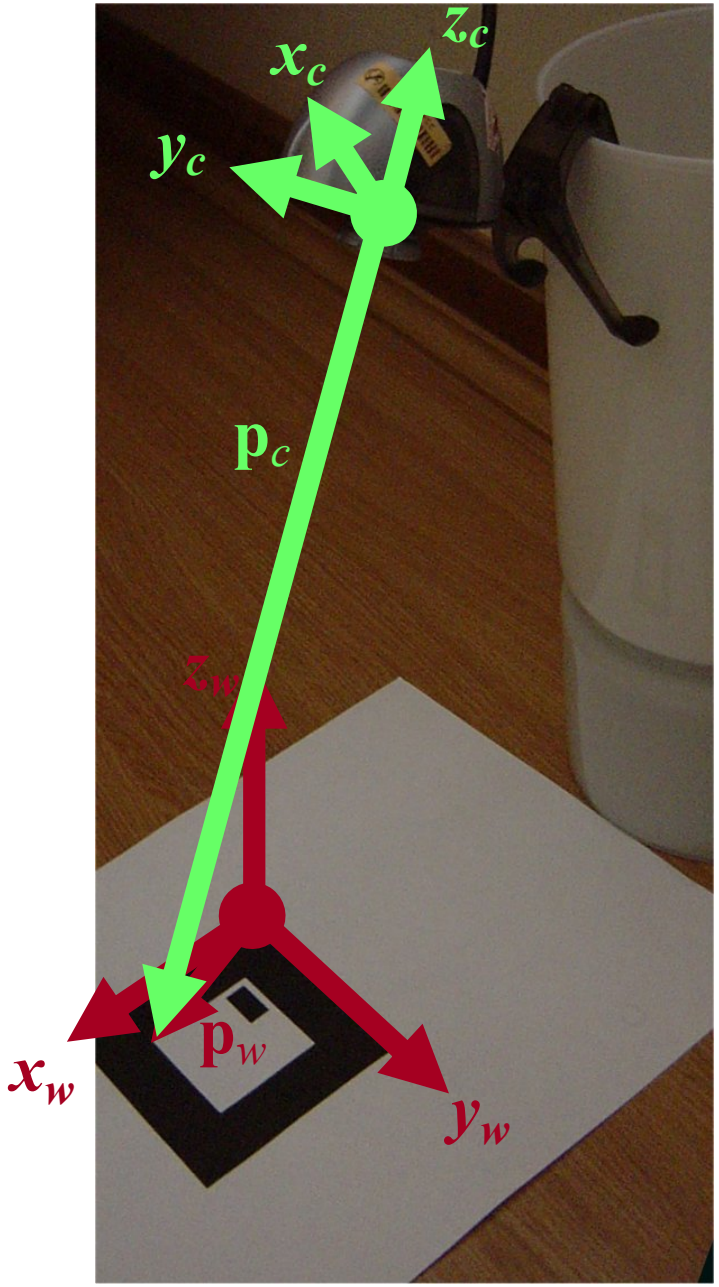
Para cada fragmento

Se a profundidade do fragmento é menor que a atual no pixel

Pinte o pixel com a cor interpolada dos vértices

View Matrix

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

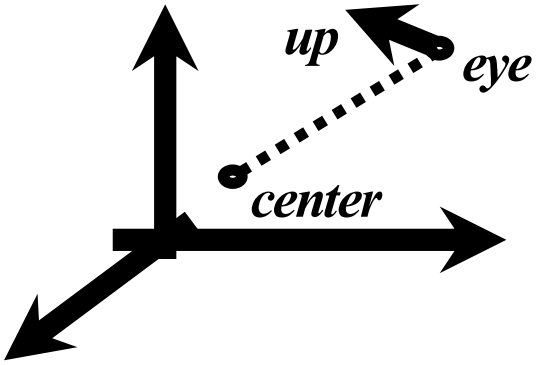


Glu Look At

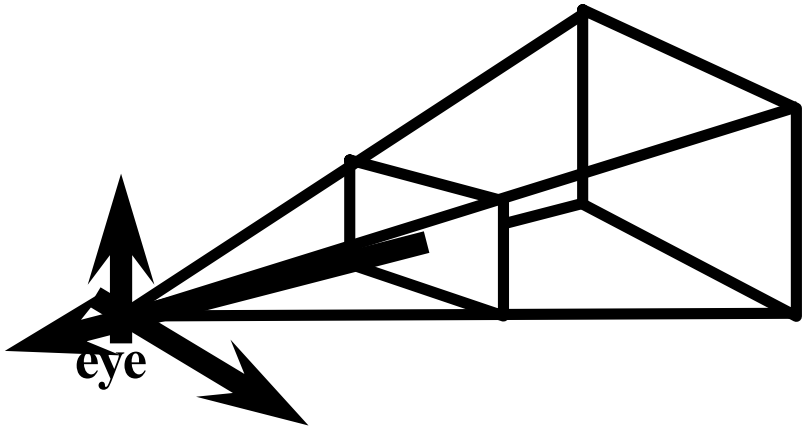
```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble centerx, GLdouble centery, GLdouble centerz,
               GLdouble upx, GLdouble upy, GLdouble upz);
```

Dados: ***eye***, ***center***, ***up*** *(definem o sistema de coordenadas do olho)*

Determine a matriz que leva do sistema de Coordenadas dos Objetos para o sistema de Coordenadas do Olho



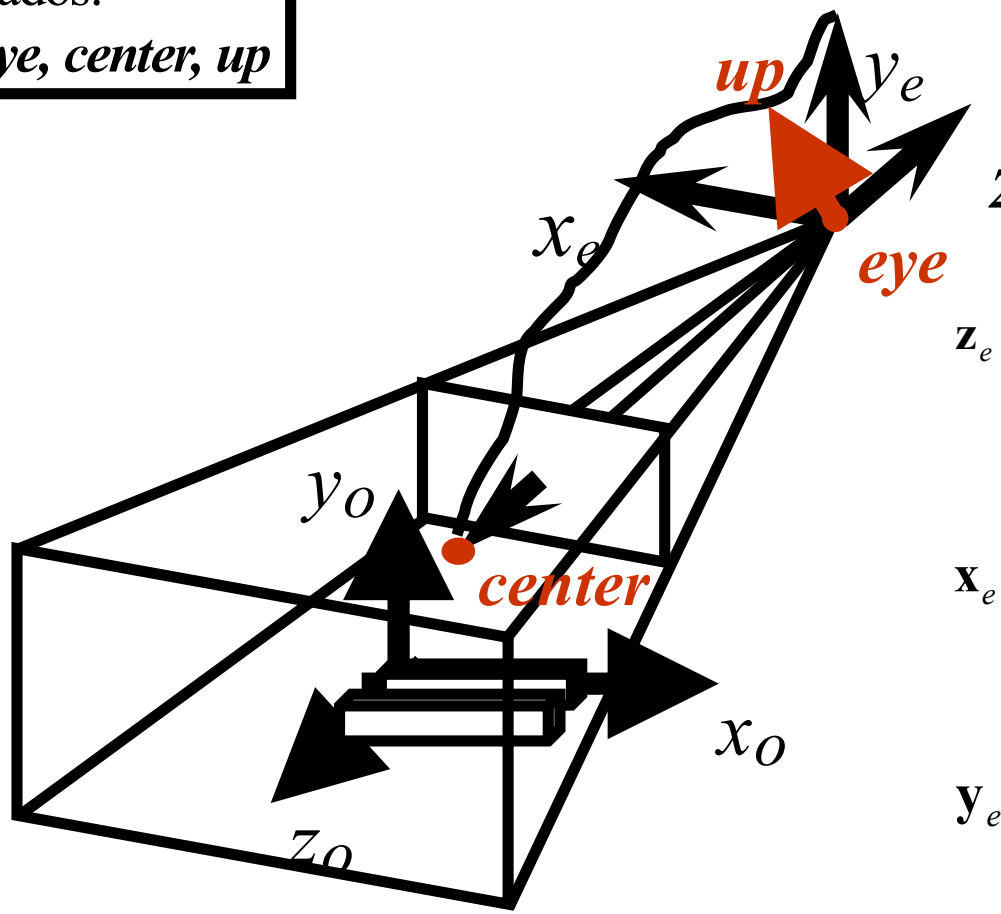
Coordenadas dos objetos



Coordenadas do olho

Calculo do sistema - $x_e\ y_e\ z_e$

dados:
eye, center, up

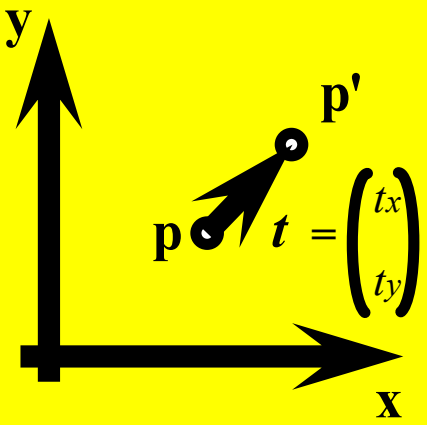


$$\mathbf{z}_e = \frac{1}{\|\mathbf{eye} - \mathbf{center}\|}(\mathbf{eye} - \mathbf{center})$$

$$\mathbf{x}_e = \frac{1}{\|\mathbf{up} \times \mathbf{z}_e\|}(\mathbf{up} \times \mathbf{z}_e)$$

$$\mathbf{y}_e = \mathbf{z}_e \times \mathbf{x}_e$$

Transformações Geométricas:
Translação



$$\mathbf{p'} = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

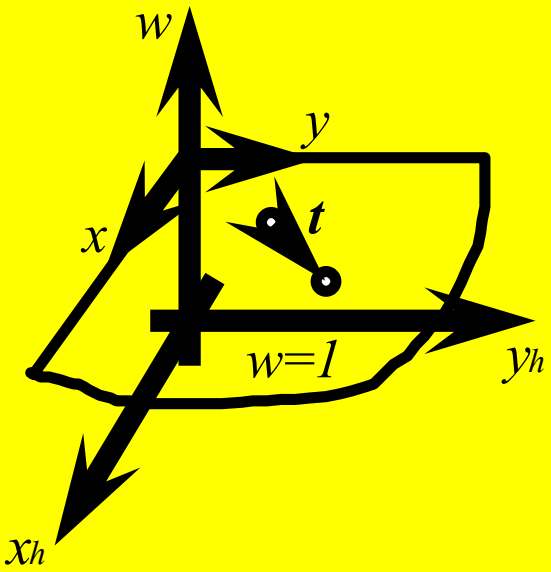
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

⇐ *Não pode ser escrito na forma* ☒

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} tx \\ ty \end{pmatrix}$$

⇐ *Ruim para implementação* ☒

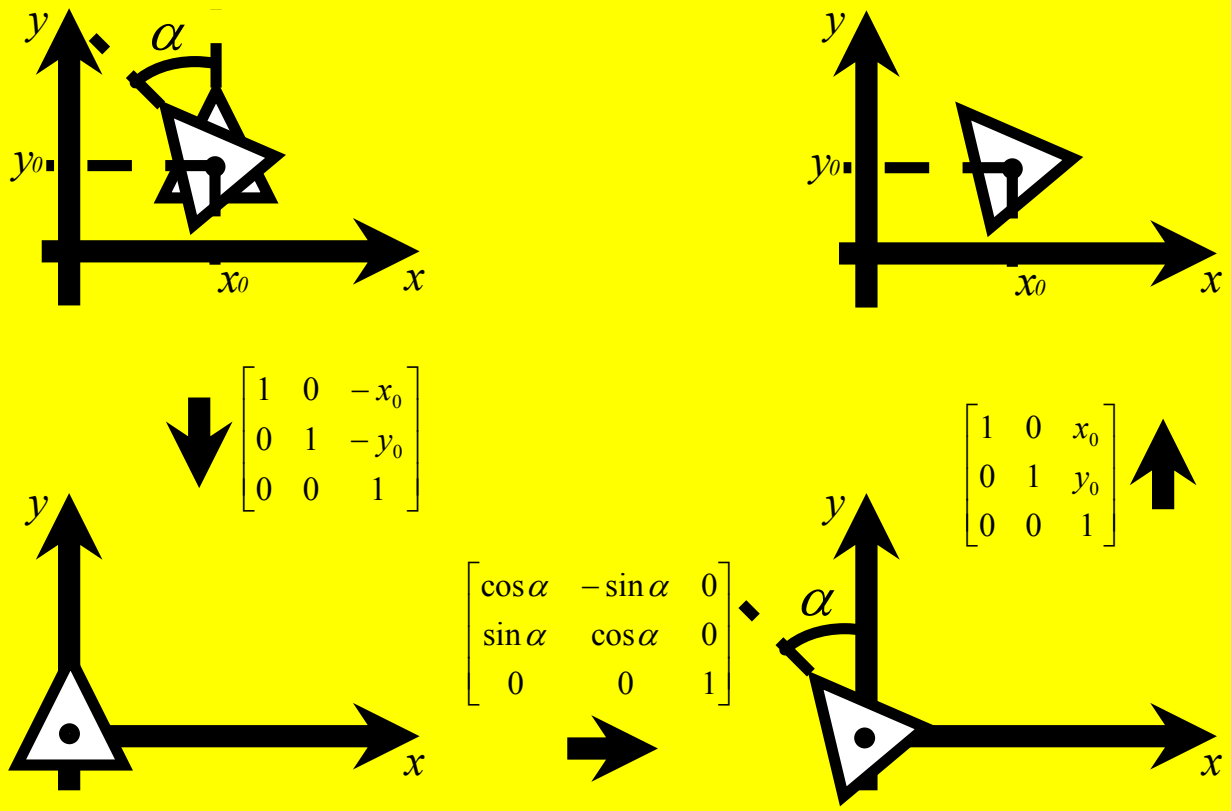
Translação num plano do R^3



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

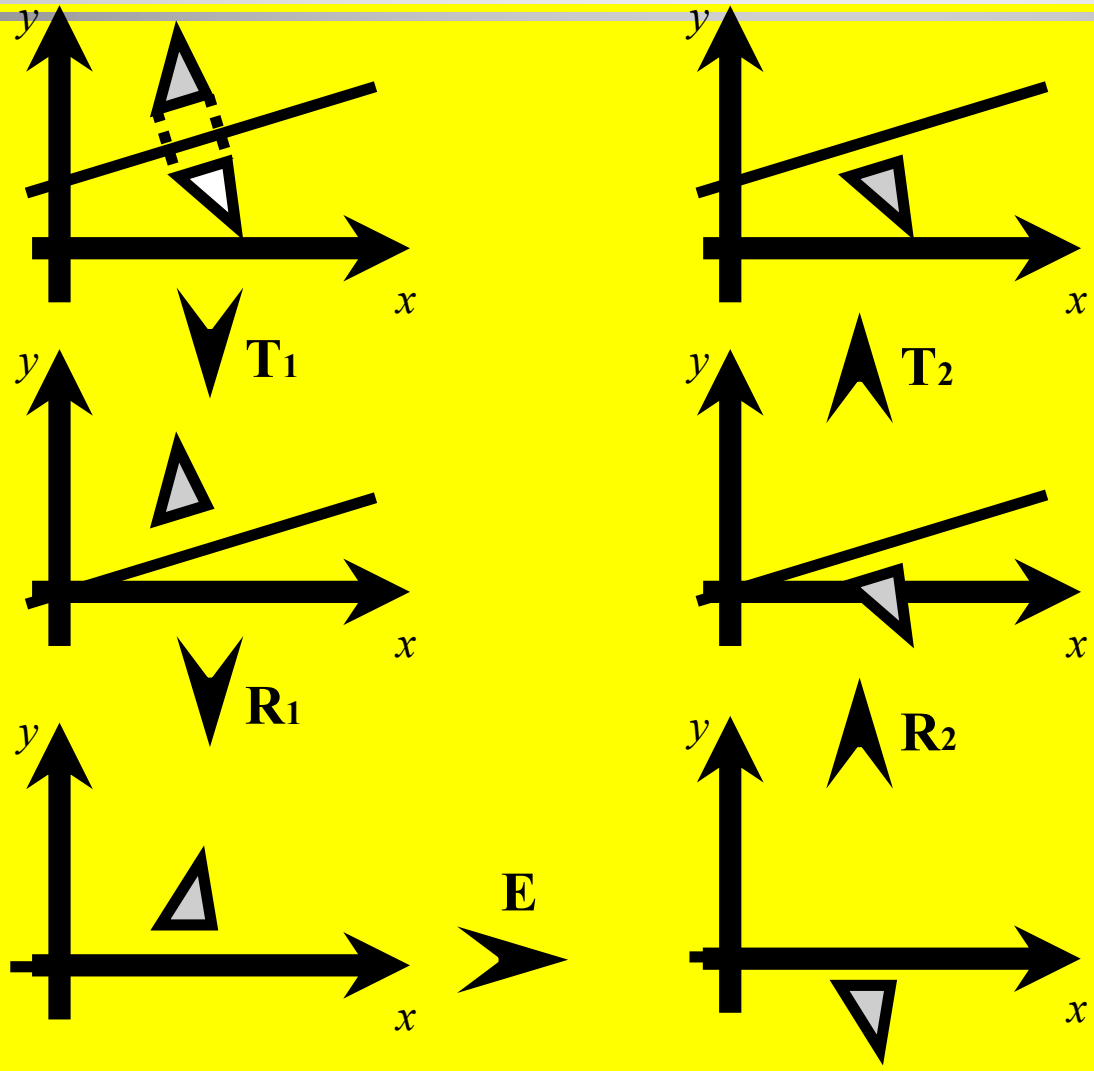
matriz de translação

Concatenação



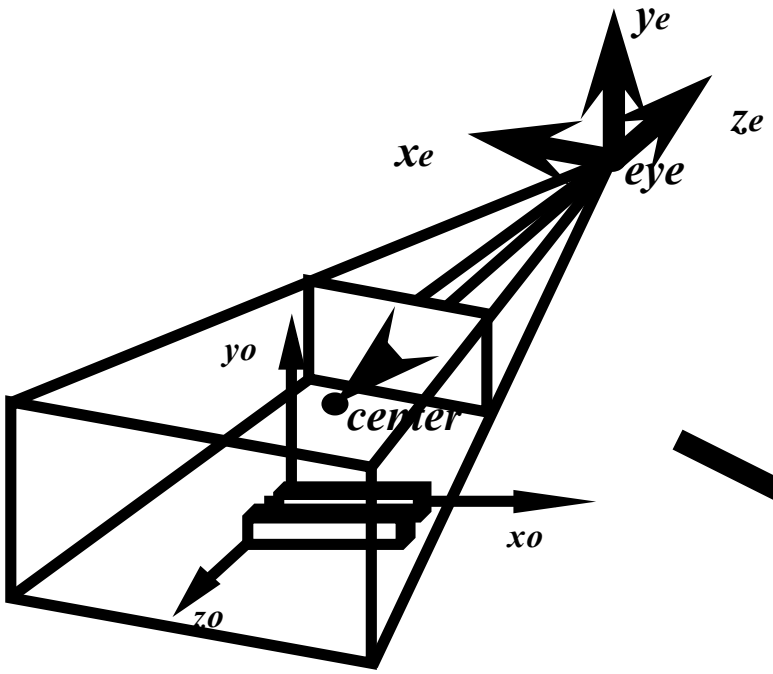
$$\begin{Bmatrix} x' \\ y' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix}$$

Concatenação

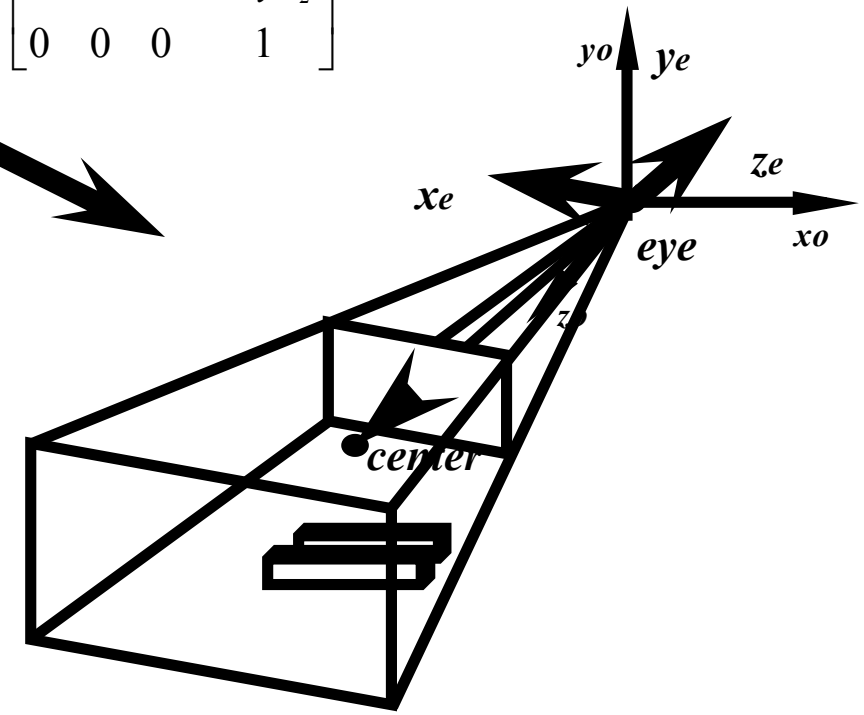


$P' = T_2\ R_2\ E\ R_1\ T_1\ P$

Translada o *eye* para origem

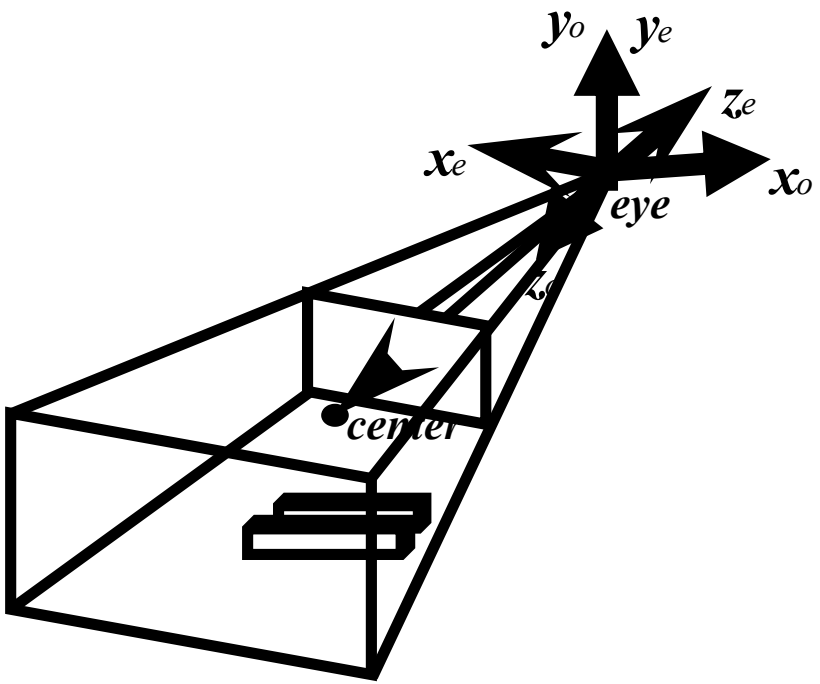


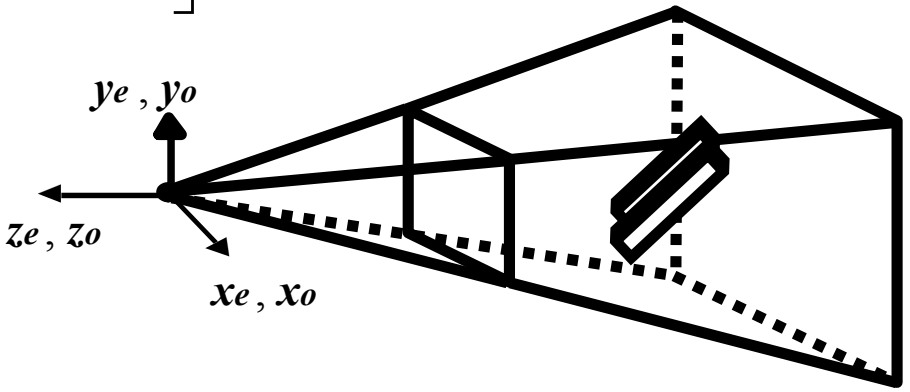
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Roda $x_e\ y_e\ z_e$ para $x_w\ y_w\ z_w$

$$\mathbf{R} = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ez} & 0 \\ z_{ex} & z_{ey} & z_{ez} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





Matriz `LookAt` do OpenGL

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

$$\mathbf{z}_e = \frac{1}{\|\mathbf{eye} - \mathbf{center}\|}(\mathbf{eye} - \mathbf{center})$$

$$\mathbf{x}_e = \frac{1}{\|\mathbf{up} \times \mathbf{z}_e\|}(\mathbf{up} \times \mathbf{z}_e)$$

$$\mathbf{y}_e = \mathbf{z}_e \times \mathbf{x}_e$$

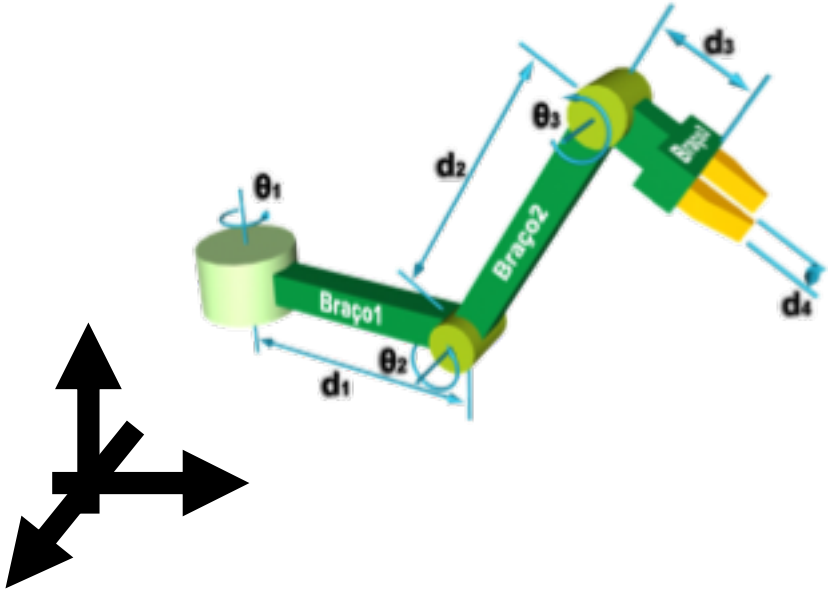
$$\mathbf{R} = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ey} & 0 \\ z_{ex} & z_{ex} & z_{ex} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

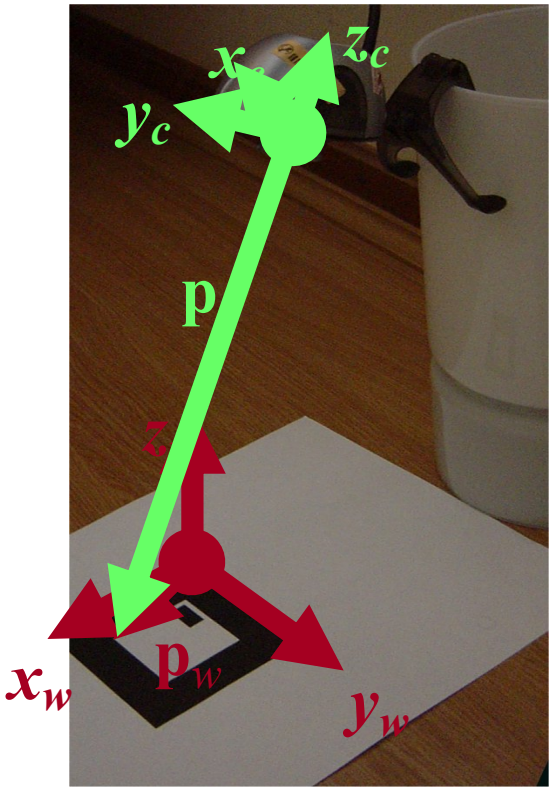
$$\mathbf{L}_{at} = \mathbf{RT} = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ey} & 0 \\ z_{ex} & z_{ex} & z_{ex} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{eye} \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{R} & -[\mathbf{R}](\mathbf{eye}) \\ \mathbf{0} & 1 \end{bmatrix}$$

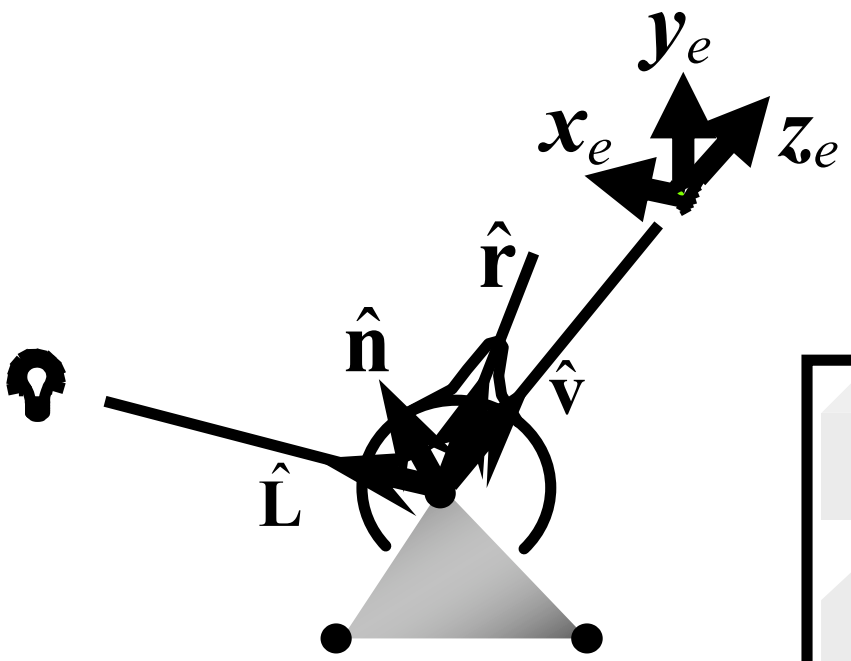
Model View

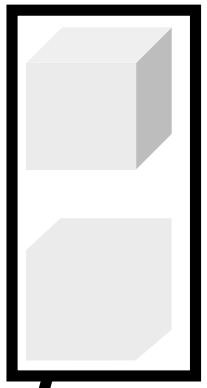


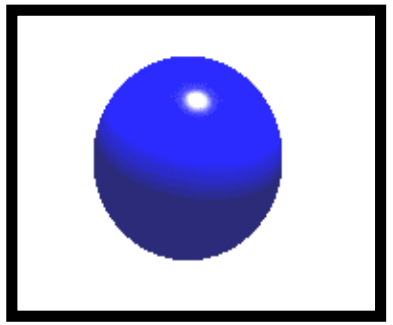


$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ey} & 0 \\ z_{ex} & z_{ex} & z_{ex} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

Cor dos vértices





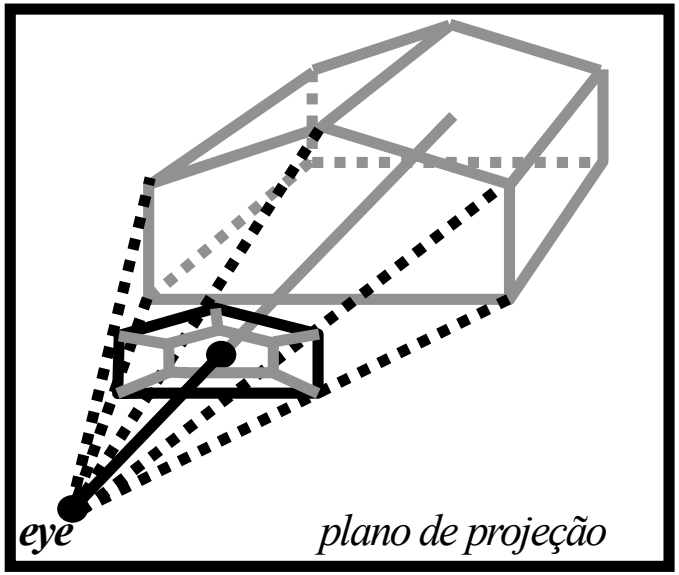


$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \sum_{luzes} \left(\begin{pmatrix} l_{ar} \\ l_{ag} \\ l_{ab} \end{pmatrix} \otimes \begin{pmatrix} k_{ar} \\ k_{ag} \\ k_{ab} \end{pmatrix} + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} \left(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}} \right) + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} \left(\hat{\mathbf{r}} \cdot \hat{\mathbf{v}} \right)^n \right)$$

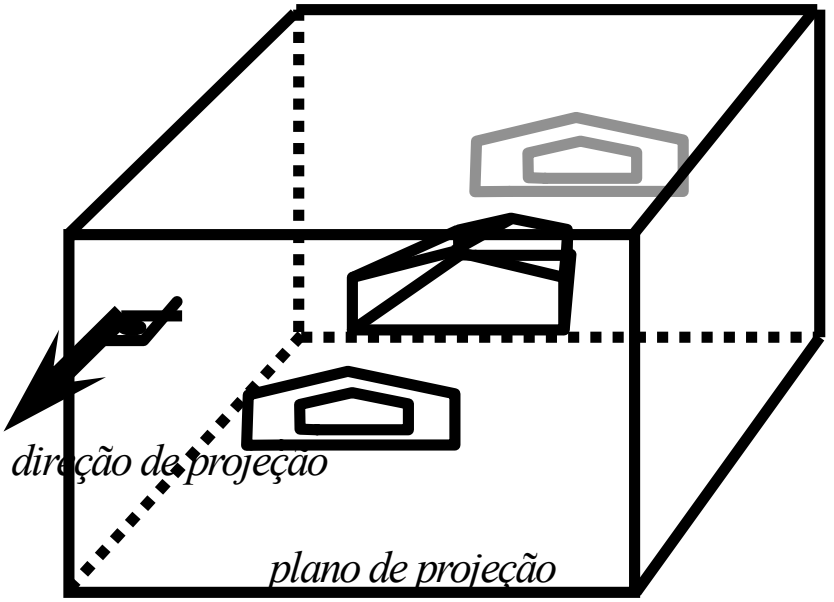
Projeção

Simplificação da projeção cônica

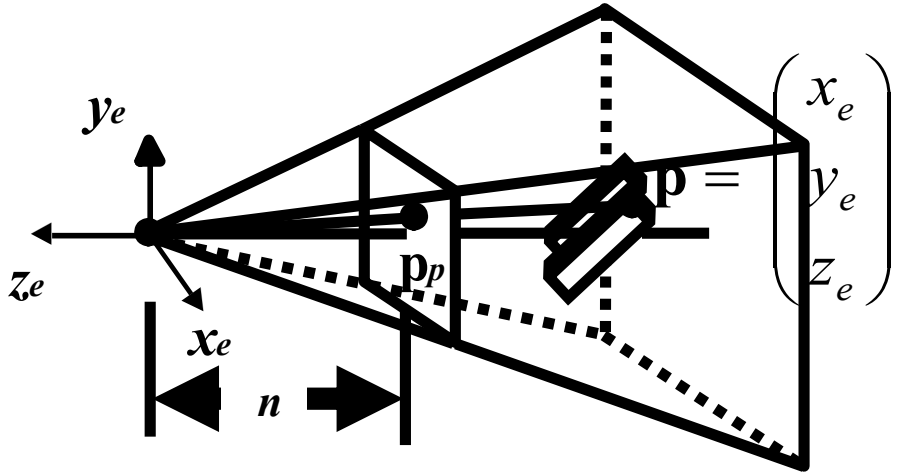
Projeção cônica



Projeção ortográfica



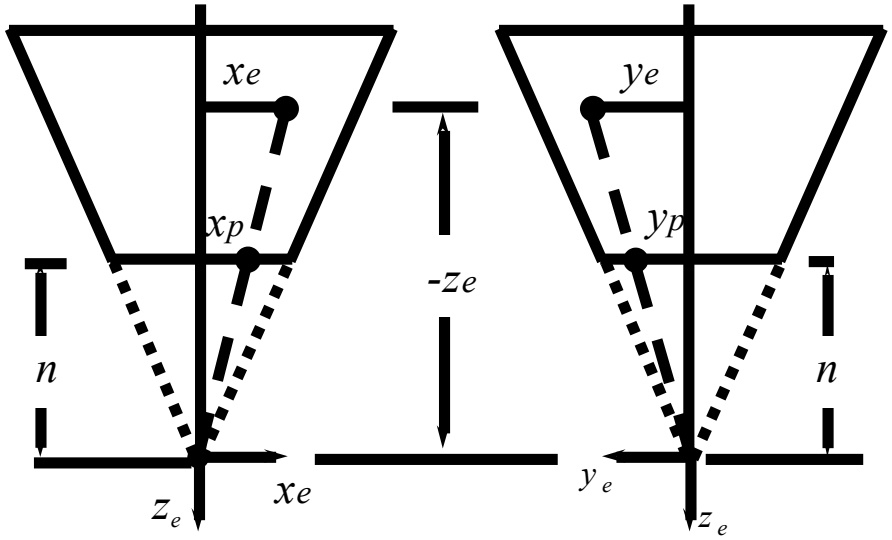
Projeção cônica simples



$$\mathbf{p}_p = \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \frac{n}{-z_e} \begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix}$$

$$\frac{xp}{xe} = \frac{n}{-ze}$$

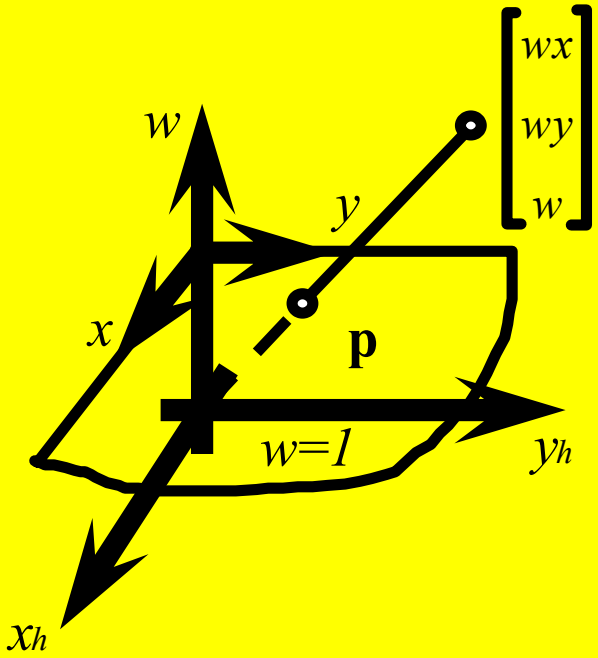
$$xp = \frac{n}{-ze}\, xe$$



$$\frac{yp}{ye} = \frac{n}{-ze}$$

$$yp = \frac{n}{-ze}\, ye$$

Coordenadas projetivas (ou homogêneas)



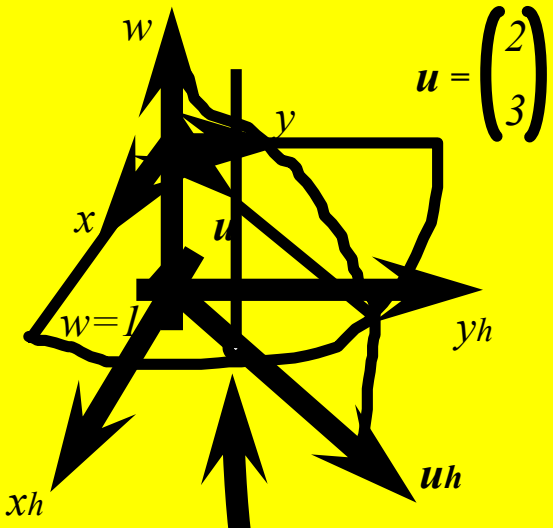
$$\mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix} \triangleq \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \triangleq \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ w \end{bmatrix}$$

$$x = x_h / w \qquad y = y_h / w \qquad w>0$$

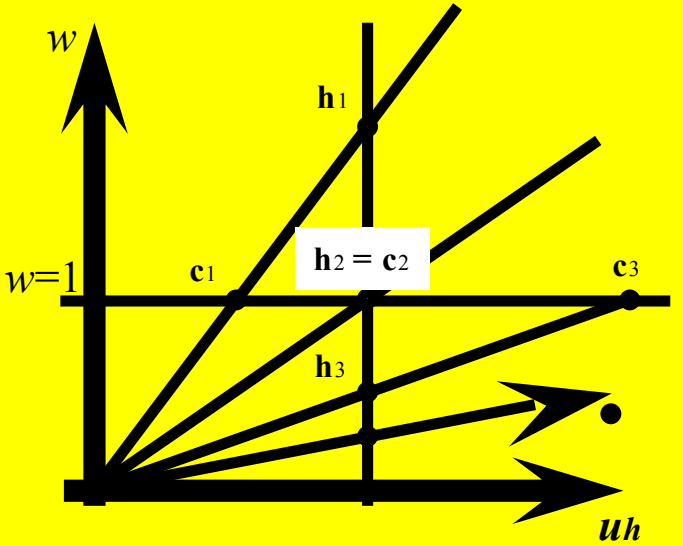
Ex.:

$$\begin{pmatrix} 3 \\ 2 \end{pmatrix} \triangleq \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \triangleq \begin{bmatrix} 6 \\ 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 9 \\ 6 \\ 3 \end{bmatrix}$$

Vantagens das coordenadas homogêneas (pontos no infinito)



$$\begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix} = \begin{pmatrix} ? \\ ? \end{pmatrix}$$

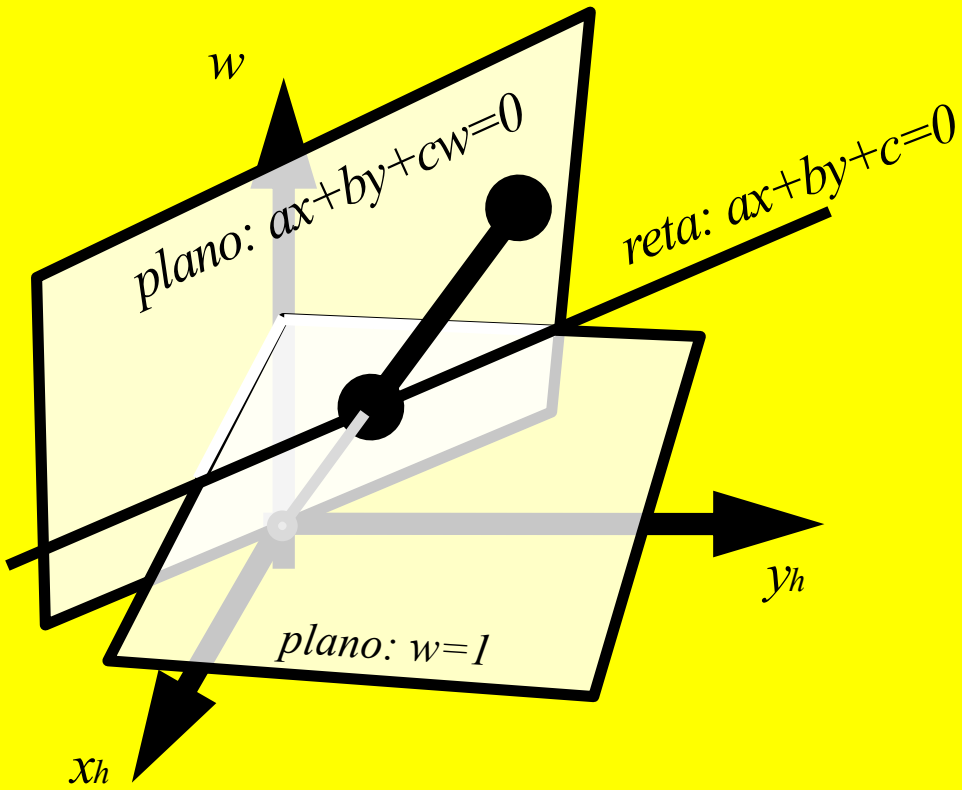


$$\underset{\mathbf{h}_1}{\begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix}} \quad \underset{\mathbf{h}_2}{\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}} \quad \underset{\mathbf{h}_3}{\begin{bmatrix} 2 \\ 3 \\ 1/2 \end{bmatrix}} \quad \underset{\mathbf{h}_4}{\begin{bmatrix} 2 \\ 3 \\ 1/4 \end{bmatrix}} \cdots \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix}$$

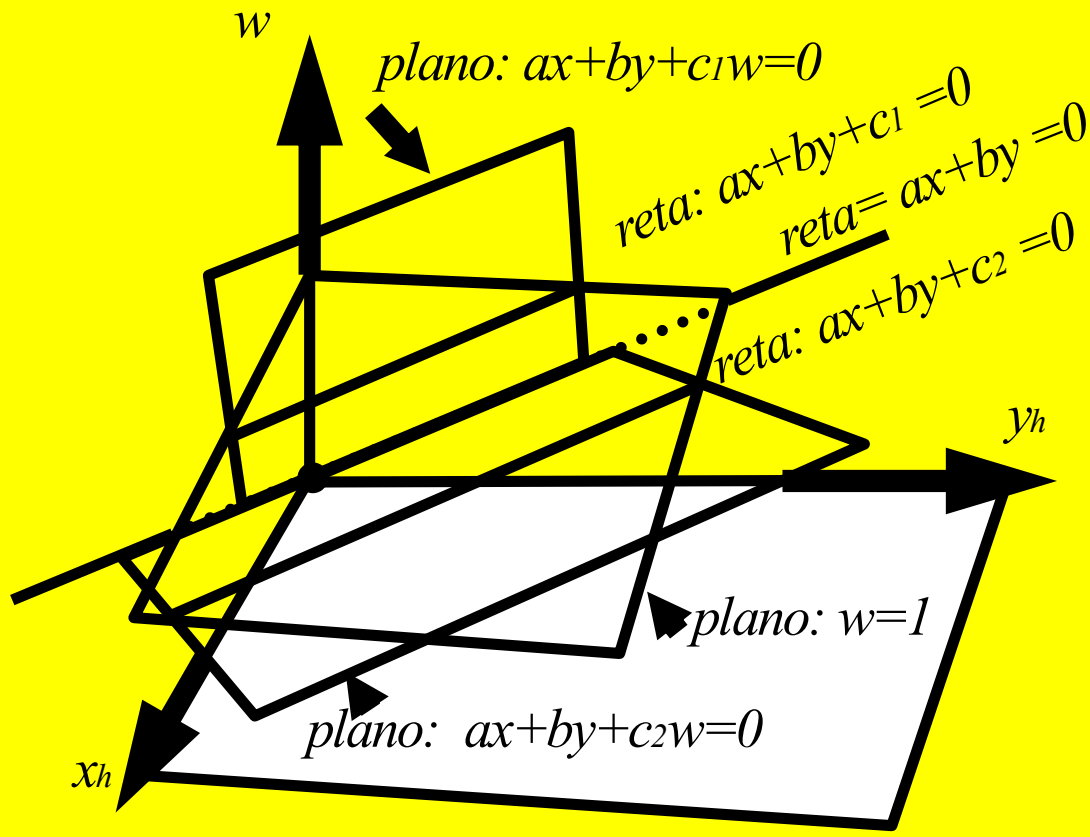
$$\underset{\mathbf{c}_1}{\begin{pmatrix} 1 \\ 1.5 \end{pmatrix}} \quad \underset{\mathbf{c}_2}{\begin{pmatrix} 2 \\ 3 \end{pmatrix}} \quad \underset{\mathbf{c}_3}{\begin{pmatrix} 4 \\ 6 \end{pmatrix}} \quad \underset{\mathbf{c}_4}{\begin{pmatrix} 8 \\ 12 \end{pmatrix}}$$

infinito na direção (2,3)

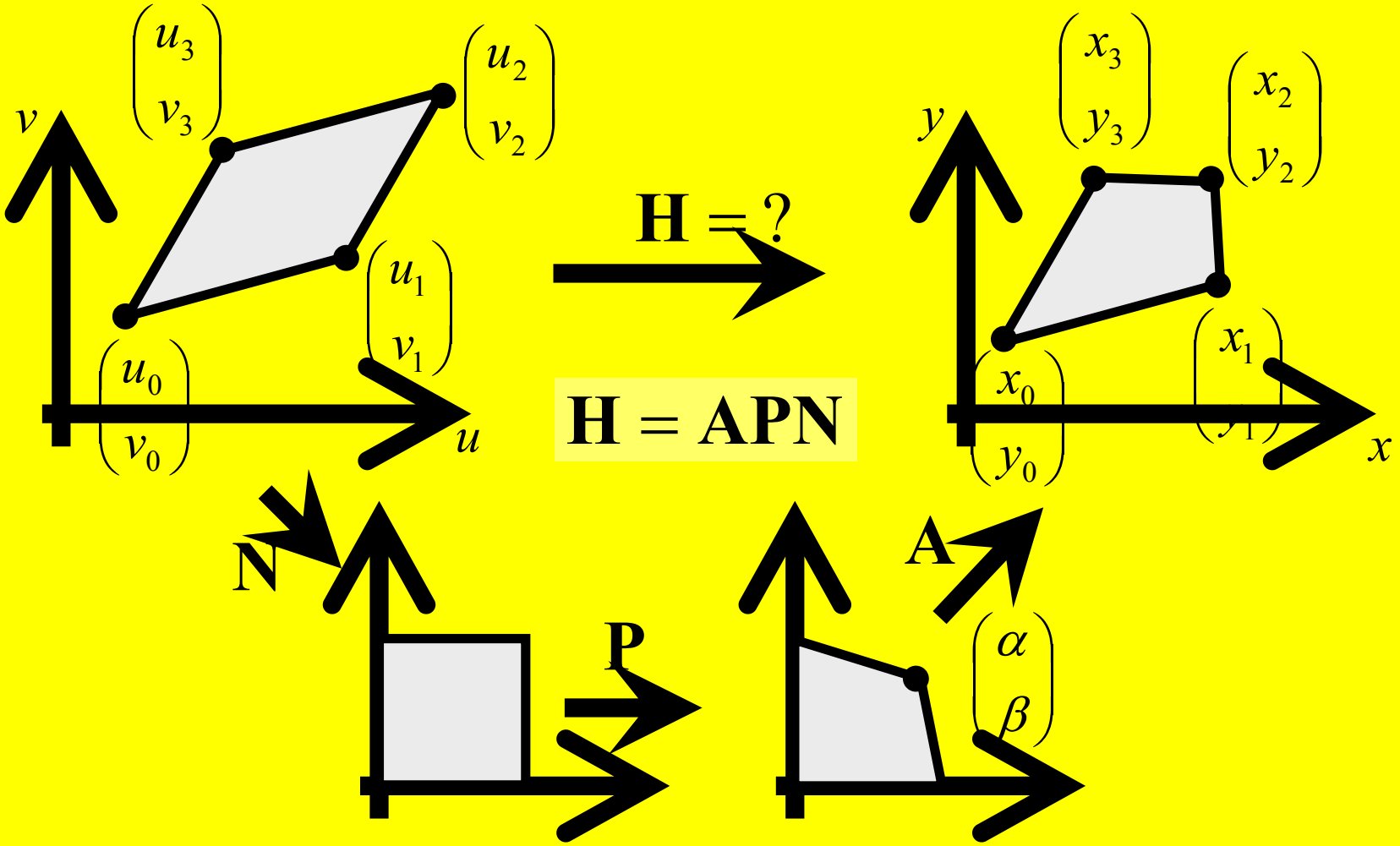
Reta no espaço projetivo



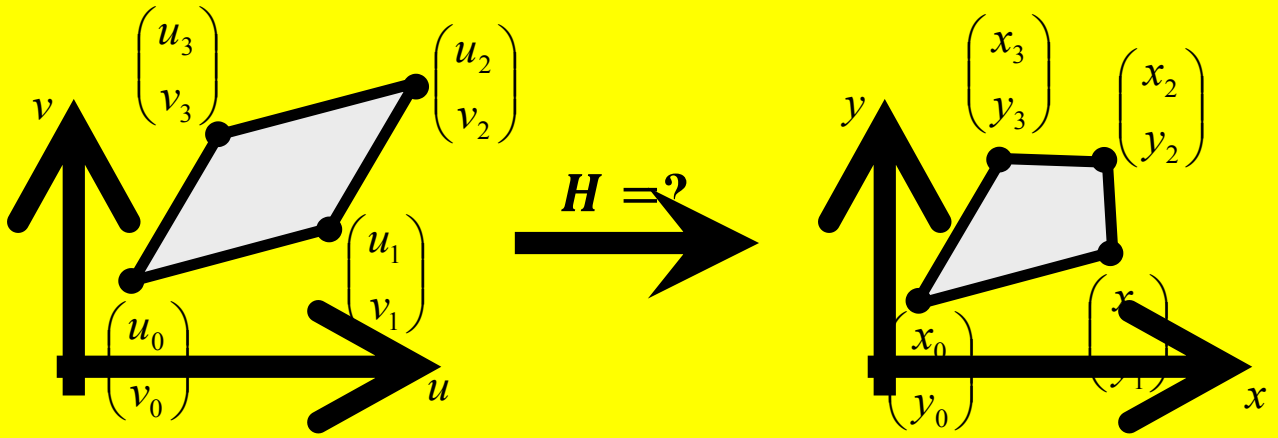
Reta paralelas no espaço projetivo



Matriz da Homografia



Matriz da Homografia



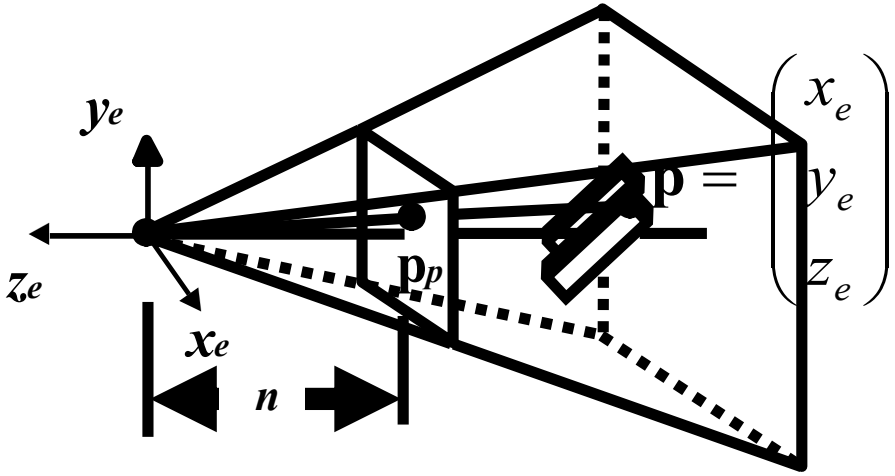
$$\boldsymbol{H} = \begin{bmatrix} h_0 & h_3 & h_6 \\ h_1 & h_4 & h_7 \\ h_2 & h_5 & h_8 \end{bmatrix}$$

$$x_0 = \frac{h_0\, u_0 + h_3 v_0 + h_6}{h_2\, u_0 + h_5 v_0 + h_8} \qquad y_0 = \frac{h_1\, u_0 + h_4 v_0 + h_7}{h_2\, u_0 + h_5 v_0 + h_8}$$

$$(h_2\, u_0 + h_5 v_0 + h_8)\, x_0 = h_0\, u_0 + h_3 v_0 + h_6$$

$$(h_2\, u_0 + h_5 v_0 + h_8)\, y_0 = h_1 u_0 + h_4 v_0 + h_7$$

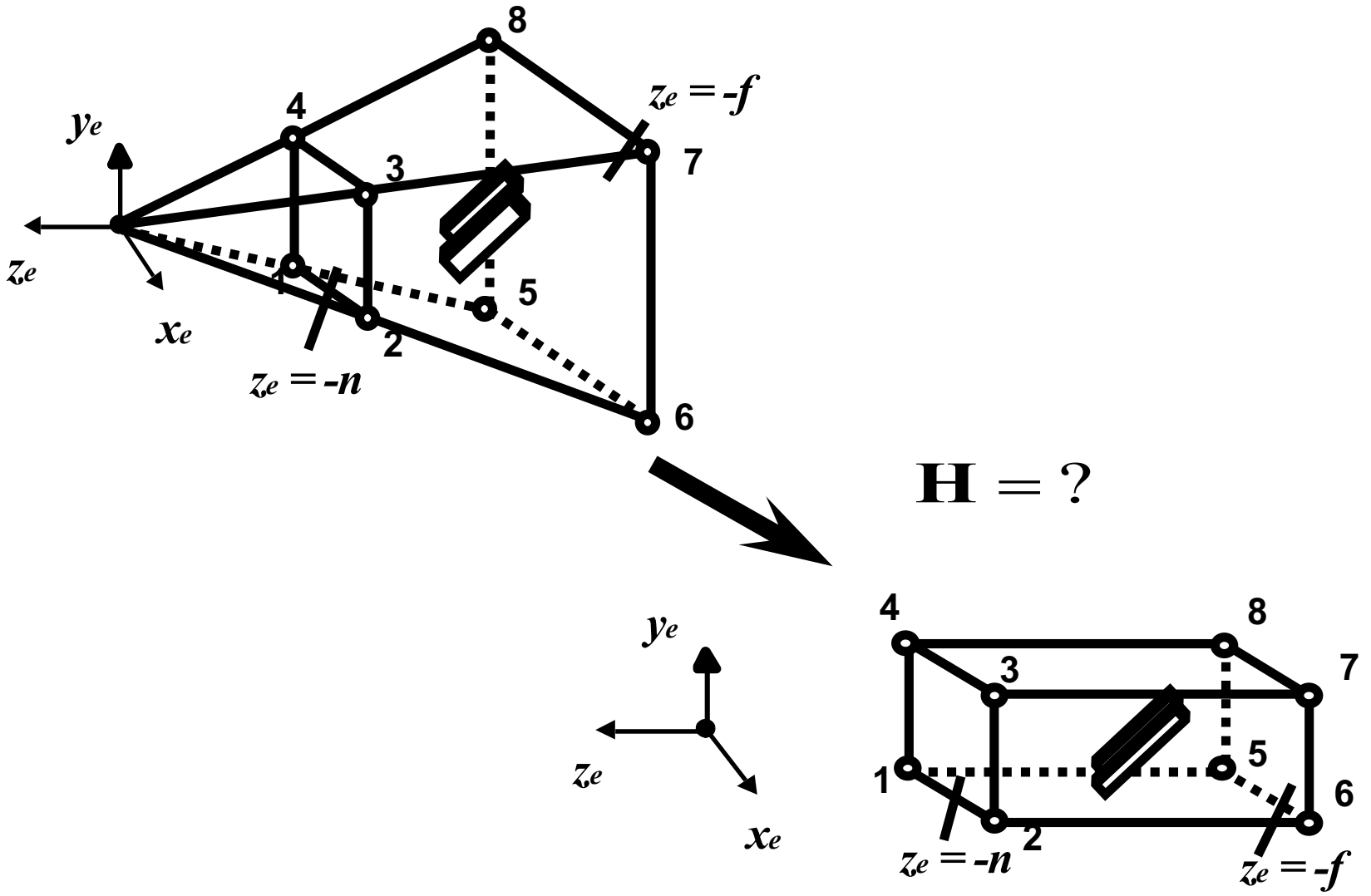
Projeção cônica simples



$$\mathbf{p}_p = \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \frac{n}{-z_e} \begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix}$$

$$\begin{bmatrix} wx_p \\ wy_p \\ wz_p \\ w \end{bmatrix} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} nx_e \\ ny_e \\ nz_e \\ -z_e \end{bmatrix} \quad \rightarrow \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \frac{1}{-z_e} \begin{pmatrix} nx_e \\ ny_e \\ nz_e \end{pmatrix}$$

Distorce o frustum de visão para o espaço da tela



Transformação projetiva
- a origem vai para o infinito em z^+ -

$$\begin{bmatrix} w_i x'_i \\ w_i y'_i \\ w_i z'_i \\ w_i \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ \alpha \\ 0 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} 0 \\ 0 \\ \alpha \\ 0 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{03} \\ m_{13} \\ m_{23} \\ m_{33} \end{bmatrix}$$

$$\mathbf{P} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & \alpha \\ m_{30} & m_{31} & m_{32} & 0 \end{bmatrix}$$

Transformação projetiva

- os pontos do plano $z = -n$ ficam imóveis -

$$\begin{bmatrix} x \\ y \\ -n \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} \beta x \\ \beta y \\ -\beta n \\ \beta \end{bmatrix} \quad \forall x, y \in R$$

$$\begin{bmatrix} \beta x \\ \beta y \\ -\beta n \\ \beta \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & \alpha \\ m_{30} & m_{31} & m_{32} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00}x + m_{01}y - m_{02}n \\ m_{10}x + m_{11}y - m_{12}n \\ m_{20}x + m_{21}y - m_{22}n + \alpha \\ m_{30}x + m_{31}y - m_{32}n \end{bmatrix} \forall x, y \in R$$

$$\beta x = m_{00}x + m_{01}y - m_{02}n \qquad \Rightarrow m_{00} = \beta, \quad m_{01} = m_{02} = 0$$

$$\beta y = m_{10}x + m_{11}y - m_{12}n \quad \Rightarrow m_{11} = \beta, \quad m_{10} = m_{12} = 0$$

$$-\beta n = m_{20}x + m_{21}y - m_{22}n + \alpha \quad \Rightarrow m_{22} = \beta + \frac{\alpha}{n}, \quad m_{20} = m_{21} = 0$$

$$\beta = m_{30}x + m_{31}y - m_{32}n \quad \Rightarrow m_{32} = -\frac{\beta}{n}, \quad m_{30} = m_{31} = 0$$

Transformação projetiva

- matriz P com estas condições -

$$\mathbf{H} = \begin{bmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \beta + \dfrac{\alpha}{n} & \alpha \\ 0 & 0 & -\dfrac{\beta}{n} & 0 \end{bmatrix}$$

Transformação projetiva

- condição sobre os pontos do plano $z = -f$ -

$$\begin{bmatrix} x \\ y \\ -f \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} \gamma \frac{n}{f} x \\ \gamma \frac{n}{f} y \\ -\gamma f \\ \gamma \end{bmatrix} \quad \forall x, y \in R$$

$$\begin{bmatrix} \gamma \frac{n}{f} x \\ \gamma \frac{n}{f} y \\ -\gamma f \\ \gamma \end{bmatrix} = \begin{bmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \beta + \frac{\alpha}{n} & \alpha \\ 0 & 0 & -\frac{\beta}{n} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ -f \\ 1 \end{bmatrix} = \begin{bmatrix} \beta x \\ \beta y \\ -\beta f - \frac{\alpha f}{n} + \alpha \\ \beta \frac{f}{n} \end{bmatrix} \quad \forall x, y \in R$$

$$\beta = \gamma \frac{n}{f}$$

$$-\gamma f = \alpha \left(1 - \frac{f}{n} \right) - \beta f \quad \Rightarrow \alpha = \gamma n$$

Transformação projetiva

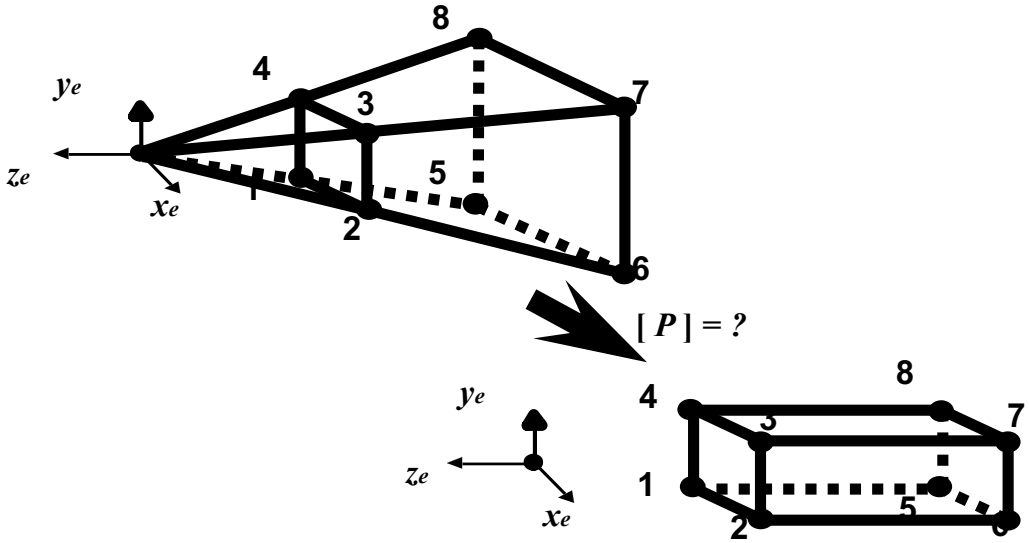
- condição sobre os pontos do plano $z = -f$ -

$$\mathbf{P} = \begin{bmatrix} \gamma\frac{n}{f} & 0 & 0 & 0 \\ 0 & \gamma\frac{n}{f} & 0 & 0 \\ 0 & 0 & \gamma\frac{n}{f}+\gamma & \gamma n \\ 0 & 0 & -\gamma\frac{1}{f} & 0 \end{bmatrix}$$

$$\gamma = f$$

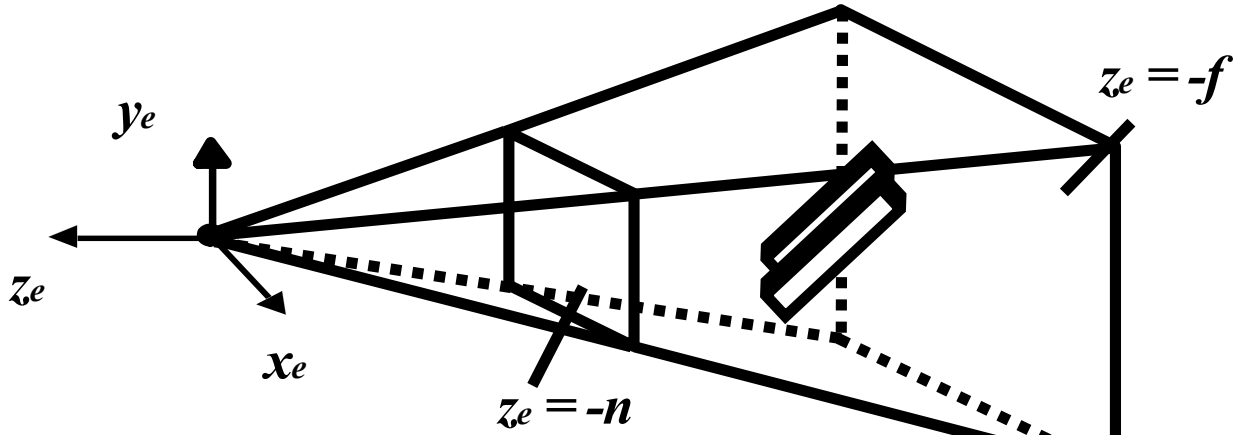
$$\mathbf{H} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Resumindo



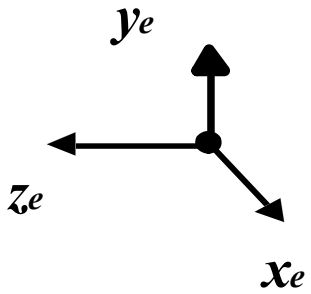
$$\mathbf{H} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

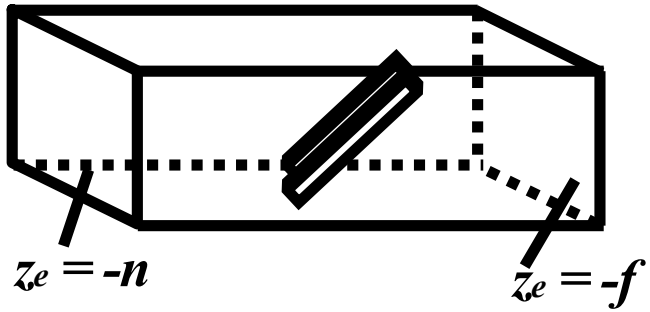
Outro enfoque: distorce o frustum de visão para o espaço da tela



$$\mathbf{H} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$d = n$





Uma equação para profundidade

$$z' = -a - \frac{b}{z}$$

Ptos no near ($z=-n$):

$$-n = -a + \frac{b}{n}$$

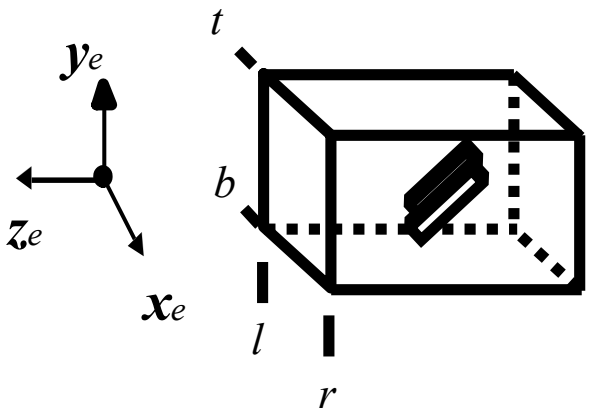
Ptos no far ($z=-f$):

$$-f = -a + \frac{b}{f}$$

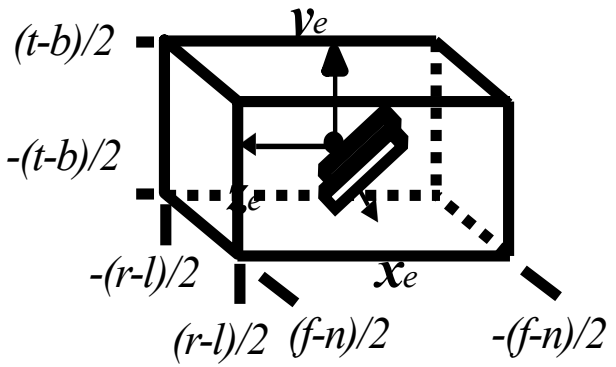
$$\Rightarrow \quad \begin{cases} a = f + n \\ b = f \cdot n \end{cases}$$

$$[\,\mathbf{H}\,] = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

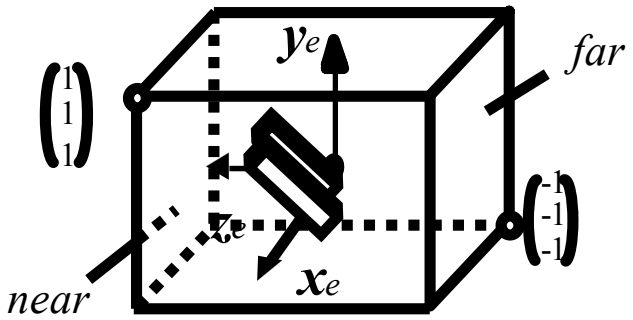
Transforma o prisma de visão
cubo normalizado [-1,1]×[-1,1] ×[-1,1]



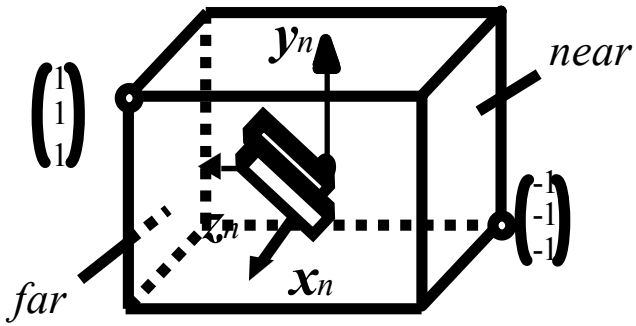
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & +(f+n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



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

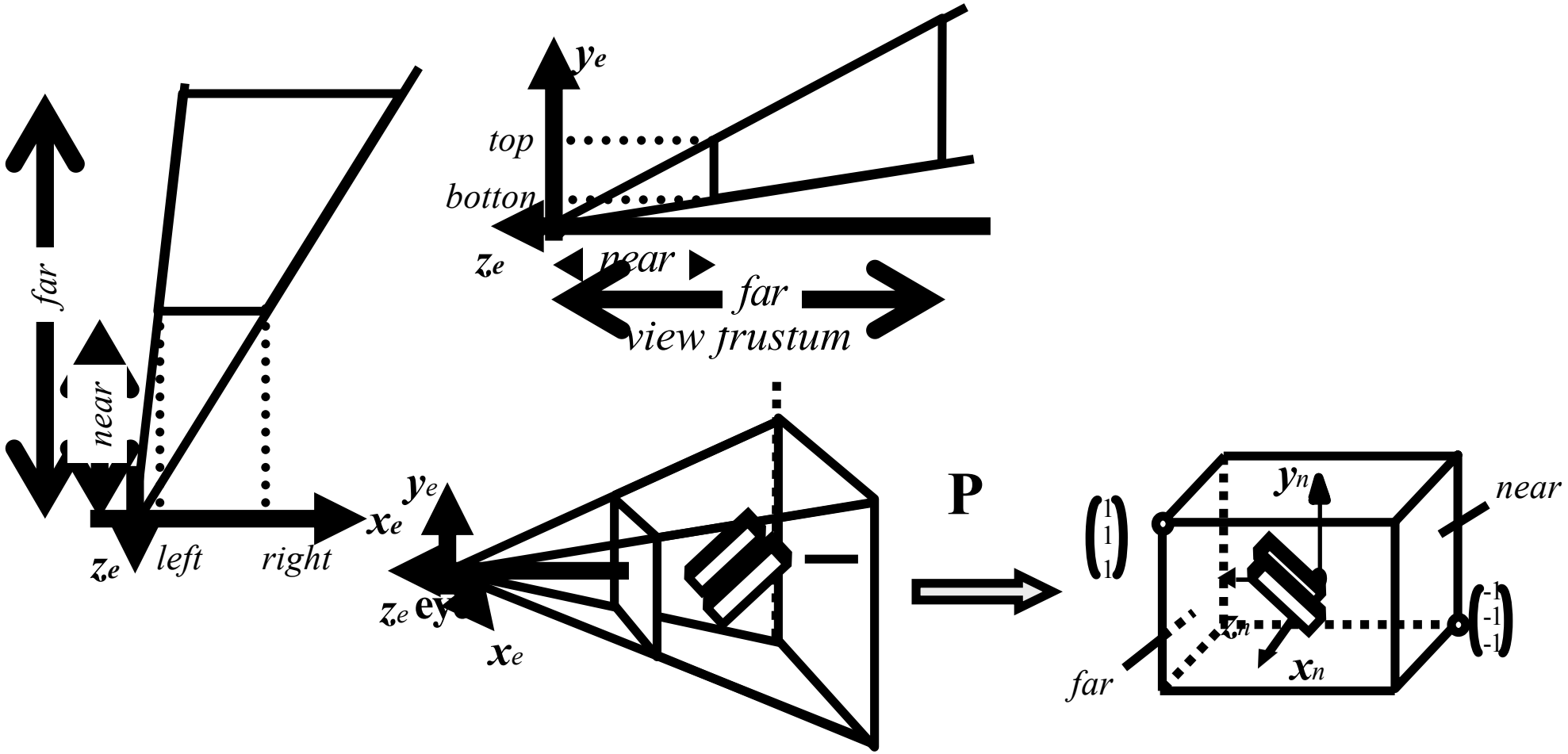


$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Projeção Cônica (*Frustum*)

```
void glFrustum(GLdouble left,GLdouble right,GLdouble bottom, GLdouble top,
               GLdouble near_, GLdouble far_ );
```



Matriz `Frustum` do OpenGL

$$\mathbf{H} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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

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

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

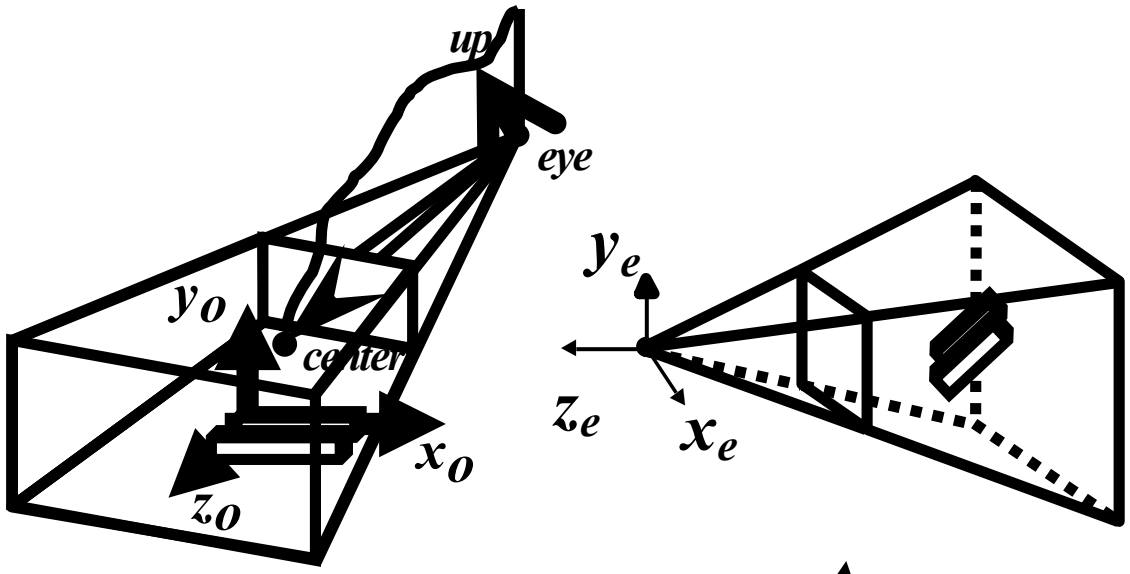
$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(f-n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & +(f+n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

OpenGL Spec

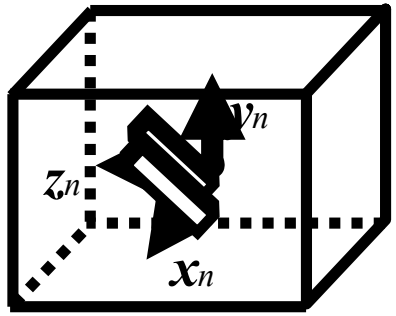
$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Resumo das transformações
(até o momento)



III.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix}$$



$$\begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix}$$

entrada

I.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = [\mathbf{M}_{view}] \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

II.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix} = [\mathbf{P}] \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

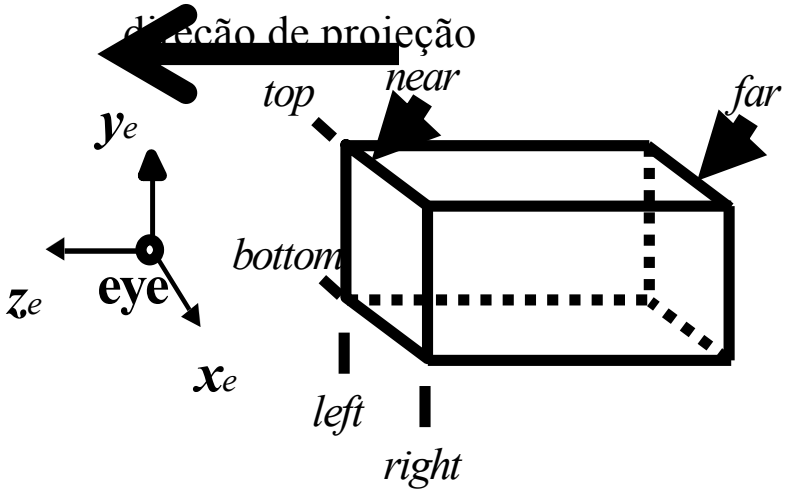
$$\frac{1}{w} \begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix}$$

IV.

$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}$$

saída

Projeção Paralela
(Ortho)




```
void glOrtho( GLdouble left, GLdouble right,
       GLdouble bottom, GLdouble top,
       GLdouble near_, GLdouble far_ );
```

```
void gluOrtho2D( GLdouble left, GLdouble right,
       GLdouble bottom, GLdouble top );
```

Matriz Ortho do OpenGL

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

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

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

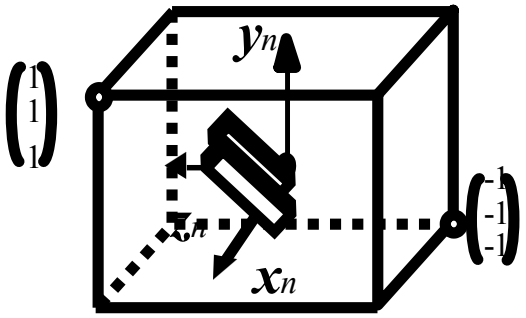
$$\mathbf{MST} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OpenGL Spec

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

Transformação para o `viewport`

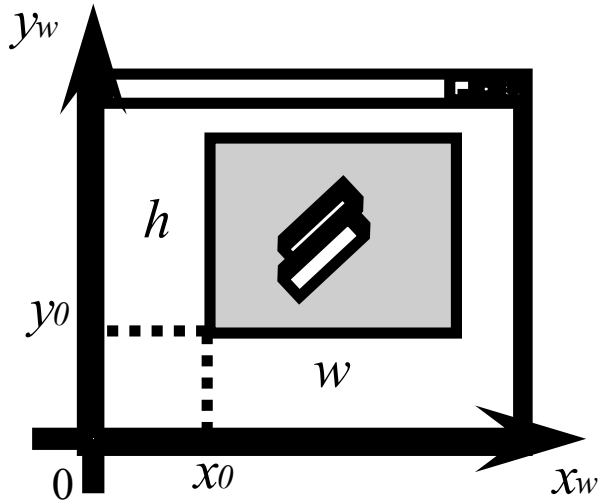
```
void glViewport(int x0, int y0, int w, int h );
```



$$x_w = x_0 + w\left(\frac{x_n + 1}{2}\right)$$

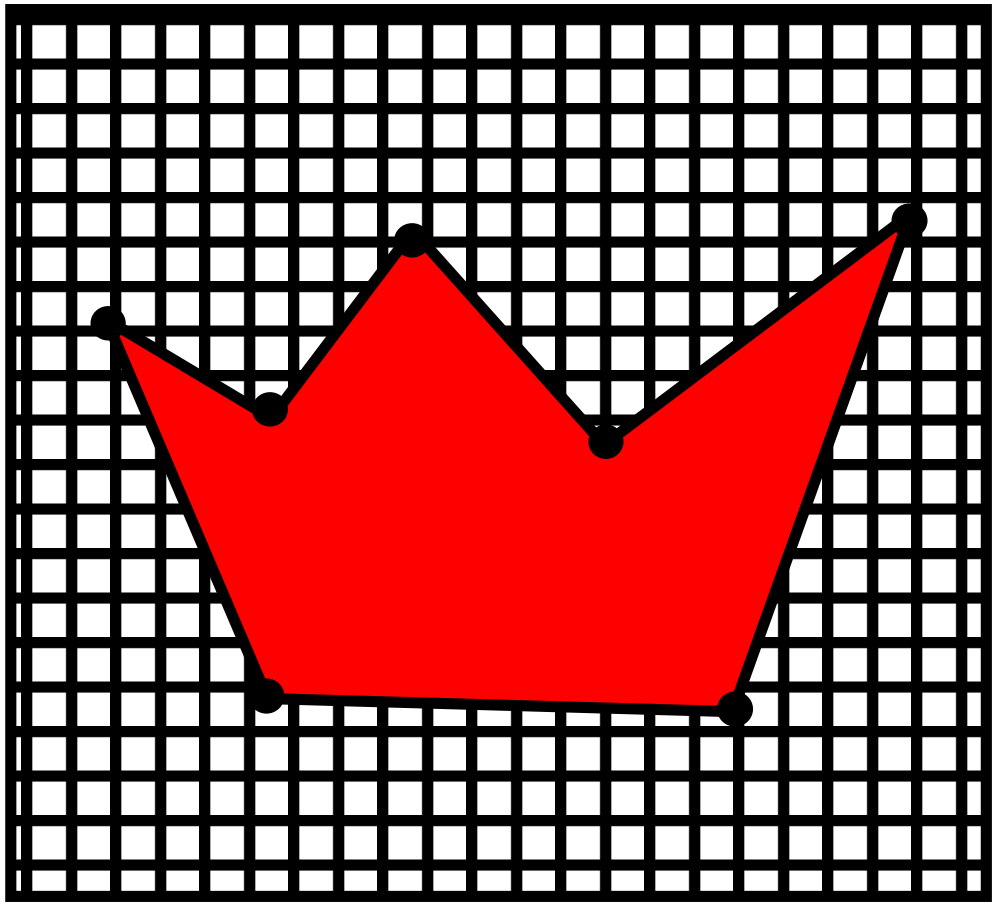
$$y_w = y_0 + h\left(\frac{y_n + 1}{2}\right)$$

$$z_w = z_{\max}\left(\frac{z_n + 1}{2}\right)$$

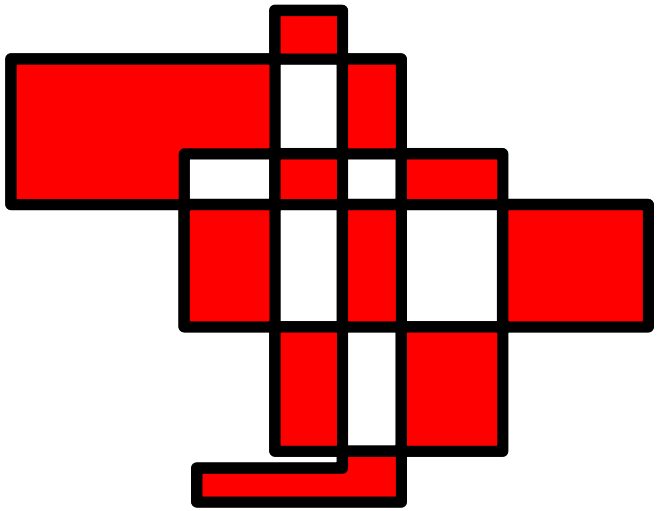


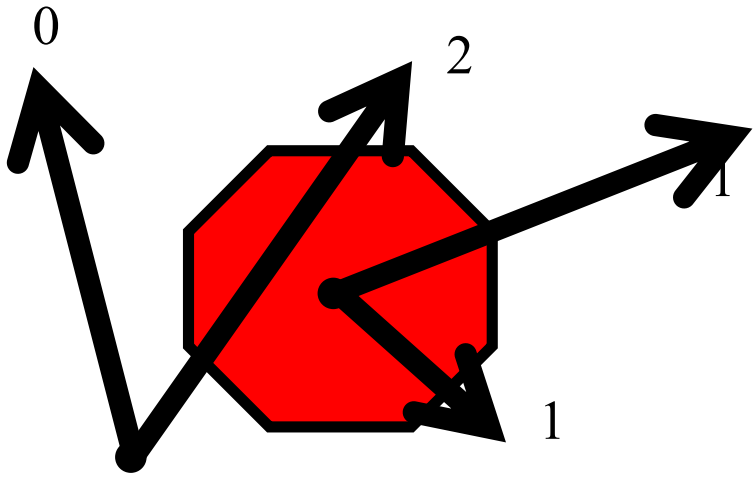
$z_w \in [0..\ z_{\max}], \quad z_{\max} = 2^n\text{-}1$ *geralmente* 65535

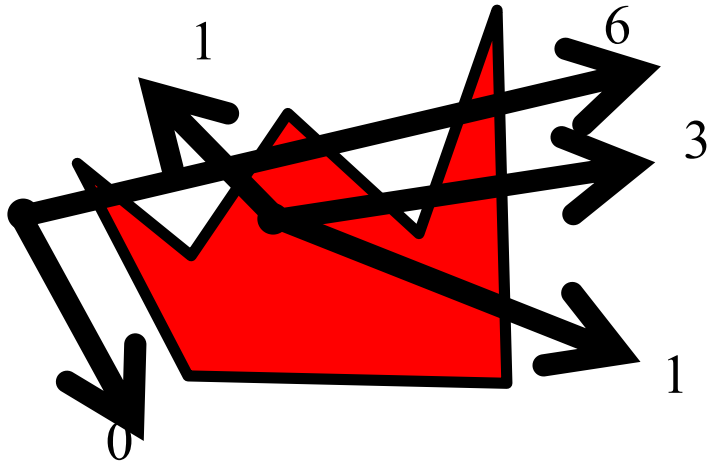
Rastreio de polígonos



Conceito de interior num polígono qualquer (regra par-ímpar)

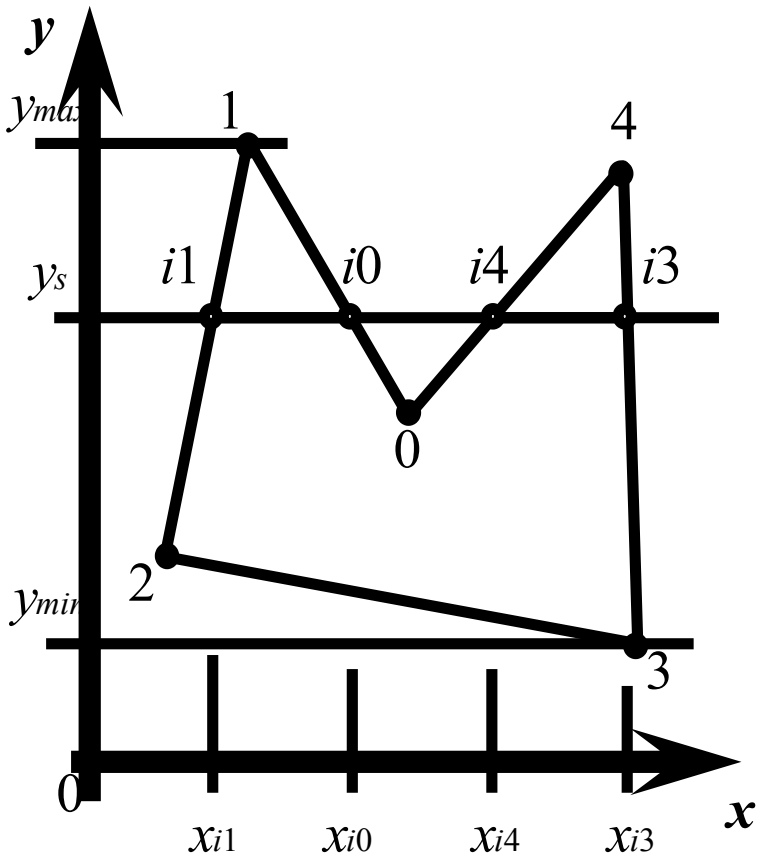






Preenchimento de polígonos

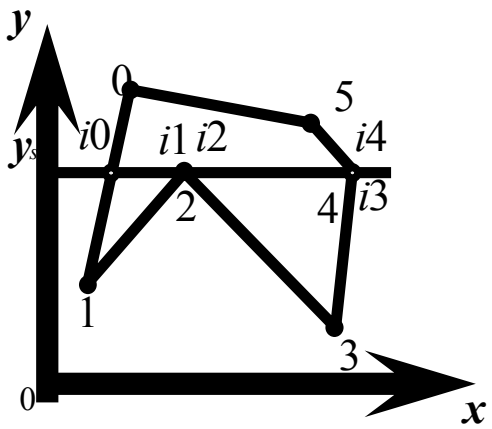
```
dados: {(x0,y0), (x1,y1), (x2,y2) (x3,y3), (x4,y4)}
```

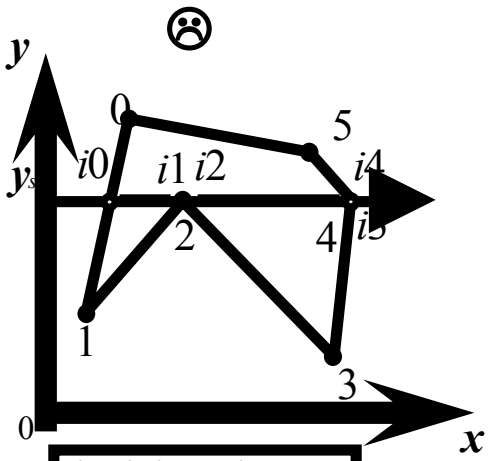


$vx = \{x_{i1}, x_{i0}, x_{i4}, x_{i3}\}$

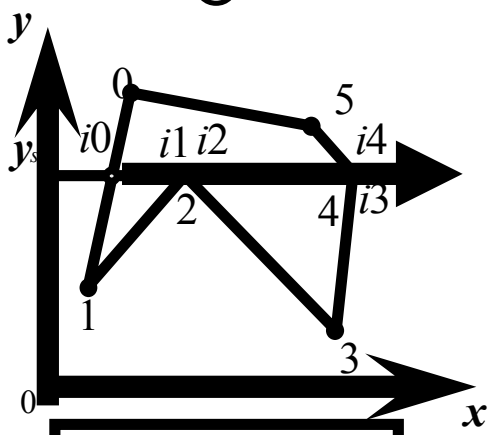
```
acha ymax e ymin
Para cada y ∈[ymax,ymin]
   Para cada aresta
       calcula as interseções
   ordena interseções
desenha linhas horizontais
```

Interseção nos vértices



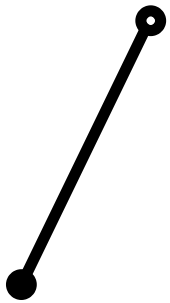


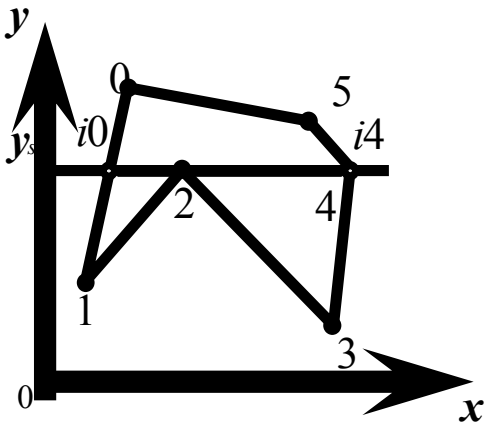
inclui vértices:
*i*0-*i*1, *i*2-i3, *i*4-?



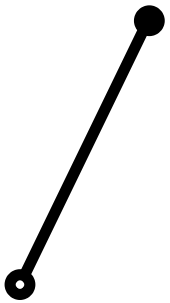
não inclui vértices:
*i*0-?

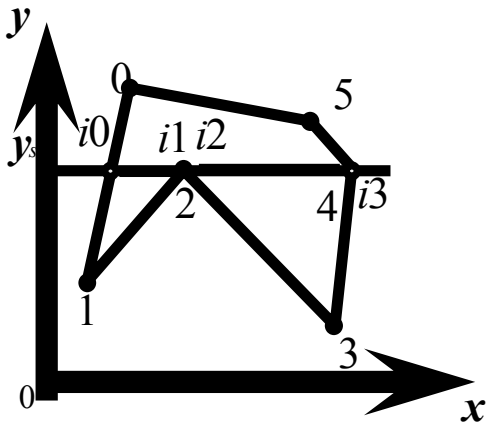
Solução:





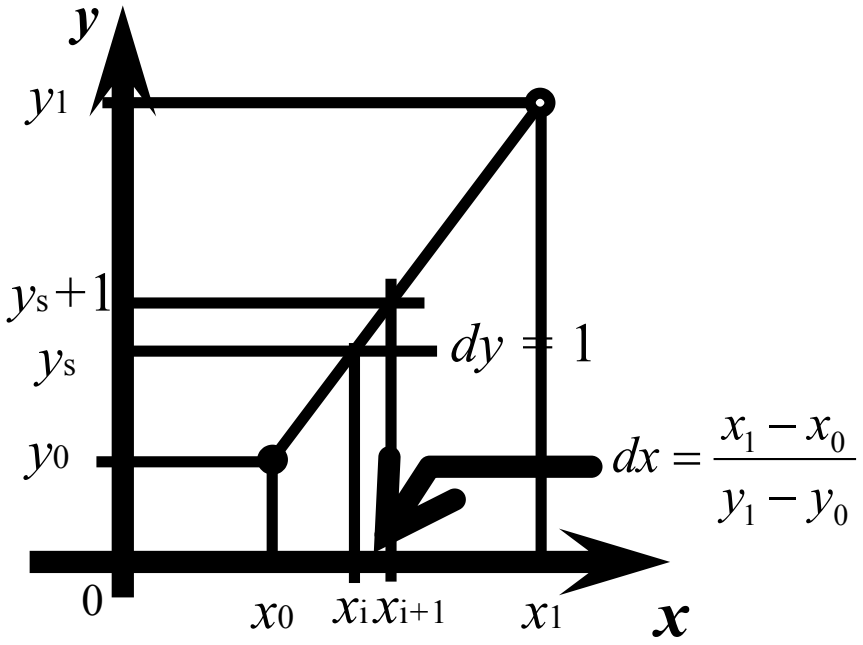
ou





Otimização do algoritmo de preenchimento: interpolação linear

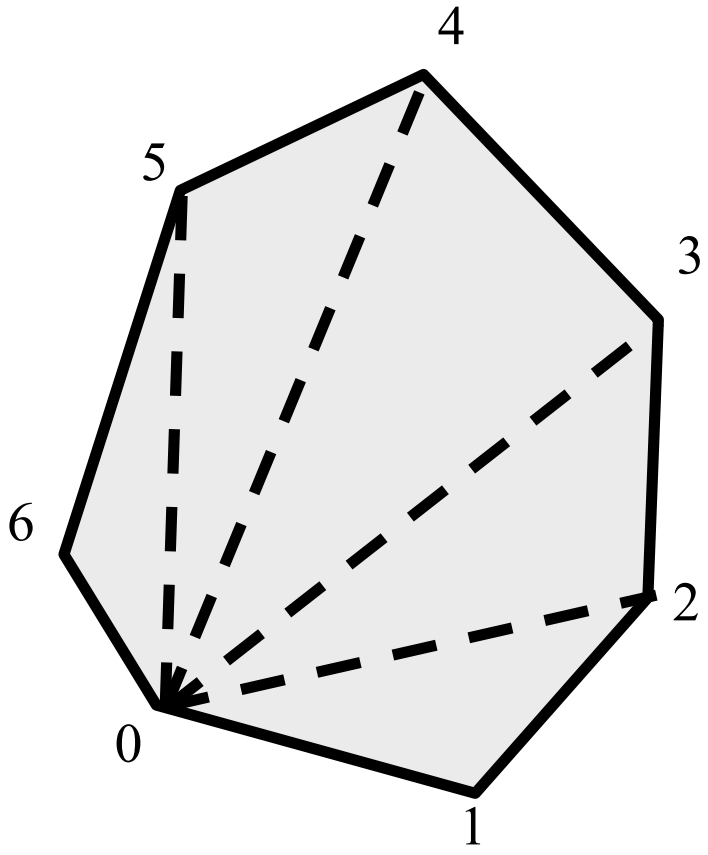
Interpolação linear na aresta

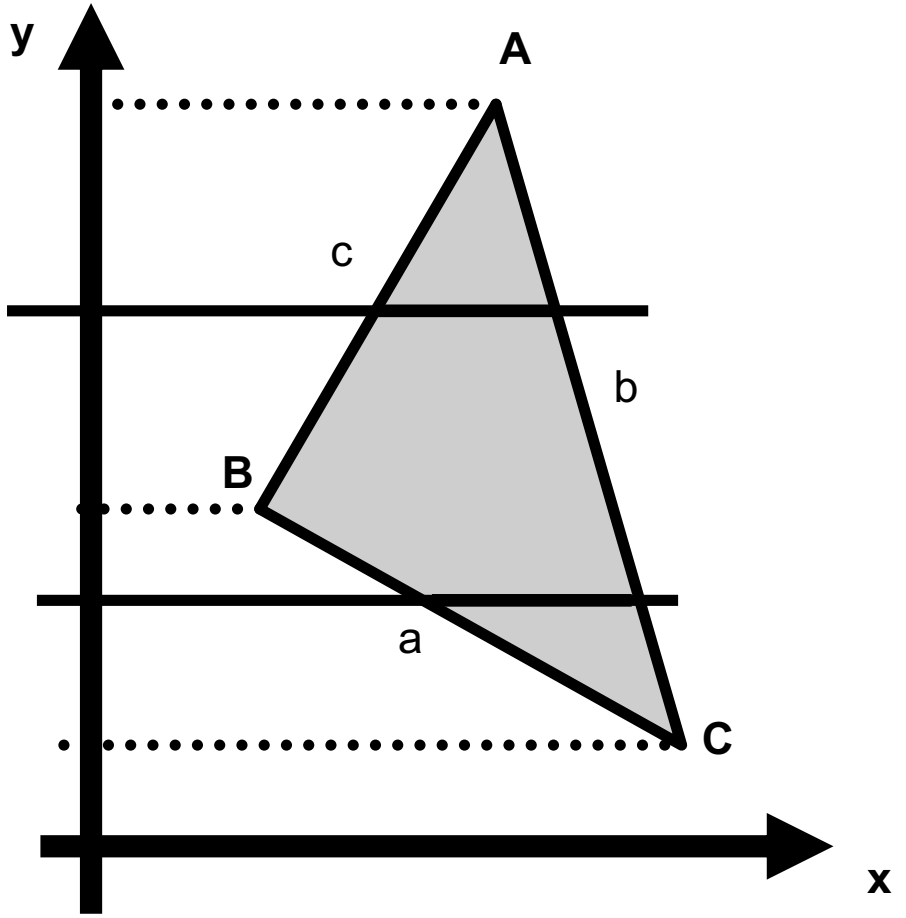


$$x_0, x_0 + dx,\ x_0 + 2dx, \ldots$$

$$c_0, c_0 + dc,\ c_0 + 2dc, \ldots$$

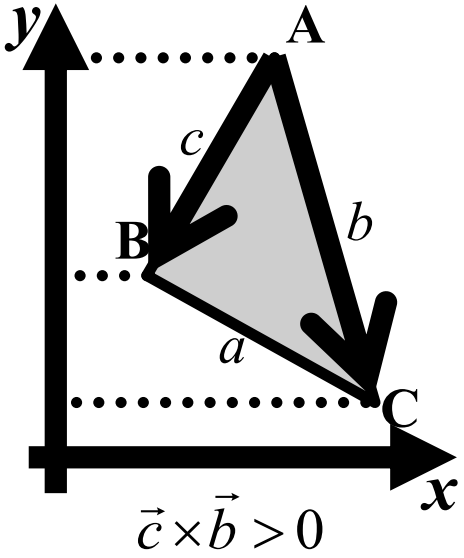
Otimizações do algoritmo de preenchimento: triângulos

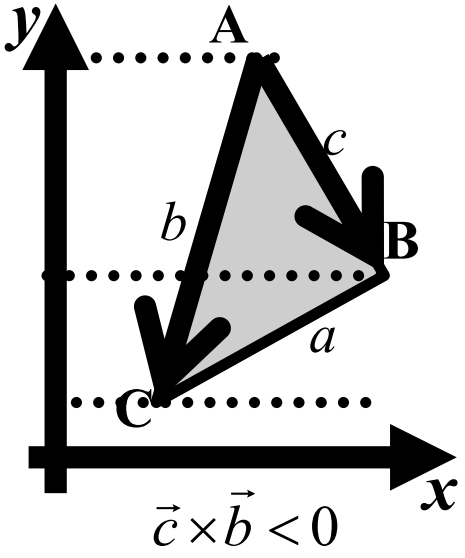




Posições possíveis de um triângulo dado

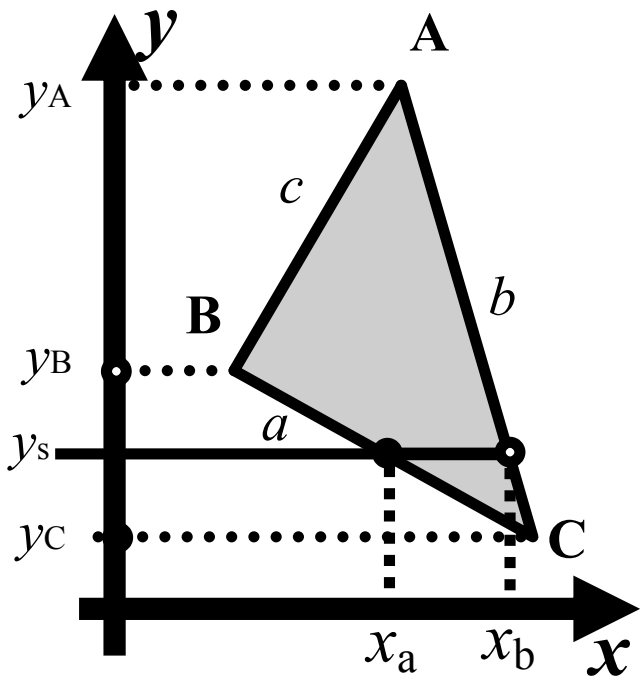
$y_A \geq y_B \geq y_C$

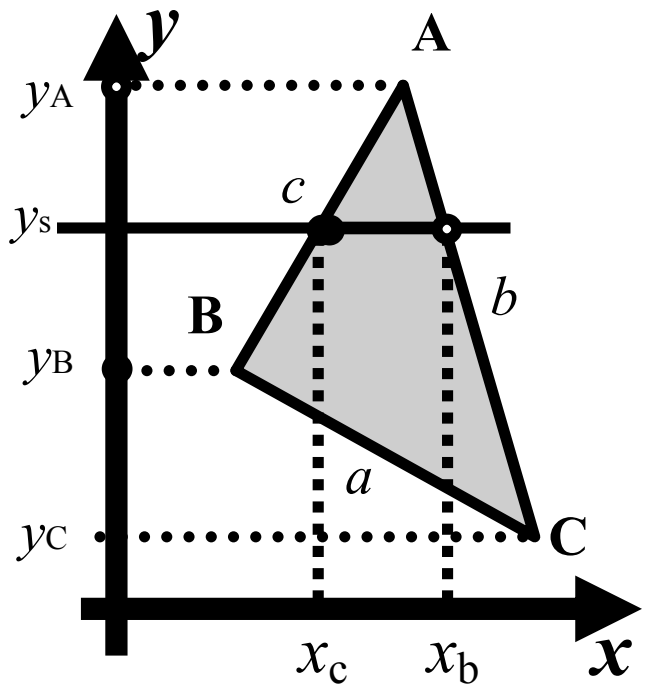




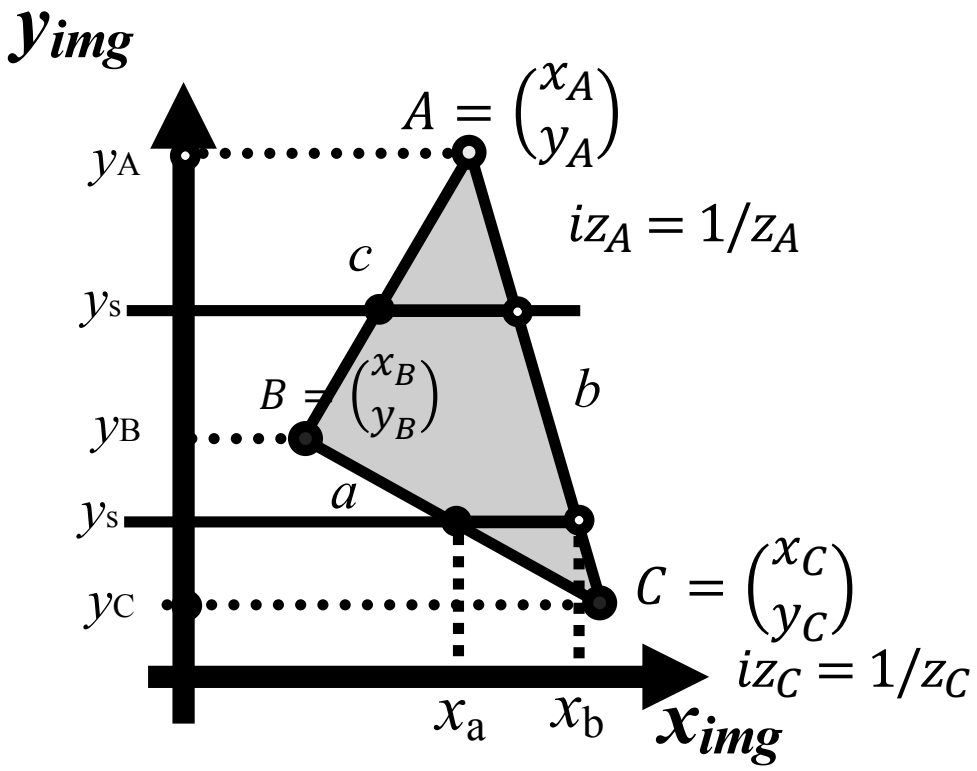
$$\vec{c} \times \vec{b} = (x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)$$

Rastreio de triângulo





Rasterização de triangulo (flat)




```
drawTriangle(T)
      xa=xb=xC,
      Pixel(xC,yC)
      for each y ∈ [yC,yB) do
            xa += dx_a
            xb += dx_b
            hLine(y,xa,xb)
      xc=xB
      for each y ∈ [yB,yA) do
            xc += dx_c
            xb += dx_b
            hLine(y,xc,xb)
```

```
hLine(y,xe,xd)
      for each x ∈ [xe,xd) do
            Pixel(x,y)
```

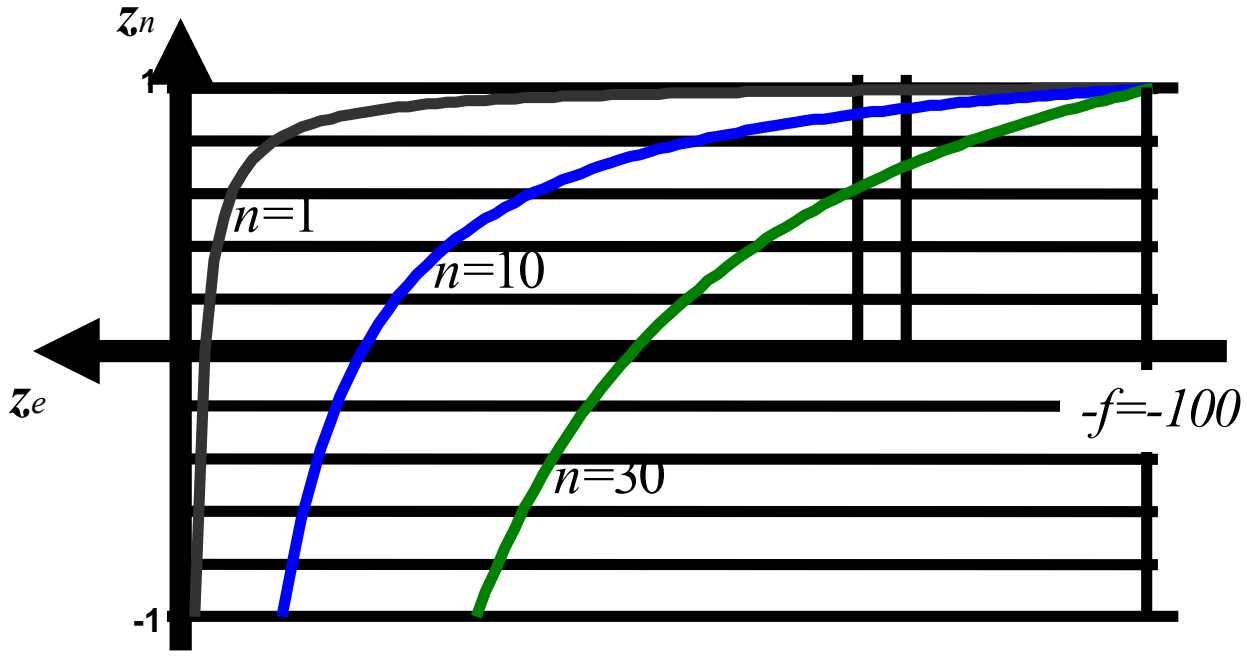
Profundidade

Mapa de profundidade

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

$$\begin{cases} z_c = -\frac{f+n}{f-n} z_e - \frac{2fn}{f-n} \\ w = -z_e \end{cases}$$

$$z_n = \frac{z_c}{w} = \frac{f+n}{(f-n)} + \frac{2fn}{(f-n)} \frac{1}{z_e}$$



Mapa de profundidade

$$z_n = \frac{z_e}{w} = \frac{f+n}{(f-n)} + \frac{2fn}{(f-n)}\frac{1}{z_e}$$

$$z_w = \frac{z_{\max}}{2}\left(z_n + 1\right), \quad z_{\max} = 2^n - 1$$

$$z_w = \frac{z_{\max}}{2}\left(\left(\frac{f+n}{(f-n)} + \frac{2fn}{(f-n)}\frac{1}{z_e}\right) + 1\right)$$

$$\frac{z_w}{z_{\max}} = 0.5\frac{f+n}{f-n} + \frac{fn}{(f-n)}\frac{1}{z_e} + 0.5$$

$$\frac{z_w}{z_{\max}} - 0.5\frac{f+n}{f-n} - 0.5 = \frac{fn}{(f-n)}\frac{1}{z_e}$$

$$\frac{(f-n)}{fn}\left(\frac{z_w}{z_{\max}} - 0.5\frac{f+n}{f-n} - 0.5\right) = \frac{1}{z_e}$$

$$z_e = \frac{fn}{(f-n)\left(\frac{z_w}{z_{\max}} - 0.5\frac{f+n}{f-n} - 0.5\right)}$$

$$z_e = \frac{fn}{\left(\frac{z_w}{z_{\max}}(f-n) - 0.5(f+n) - 0.5(f-n)\right)}$$

$$z_e = \frac{fn}{\frac{z_w}{z_{\max}}(f-n) - f}$$

$$f = 1000, \quad n = 0.01, \quad z_{\max} = 65535$$

$$z_w = 65534 \Rightarrow z_e = -396 \text{ !!!}$$

Transformações de um vértice

OpenGL Spec

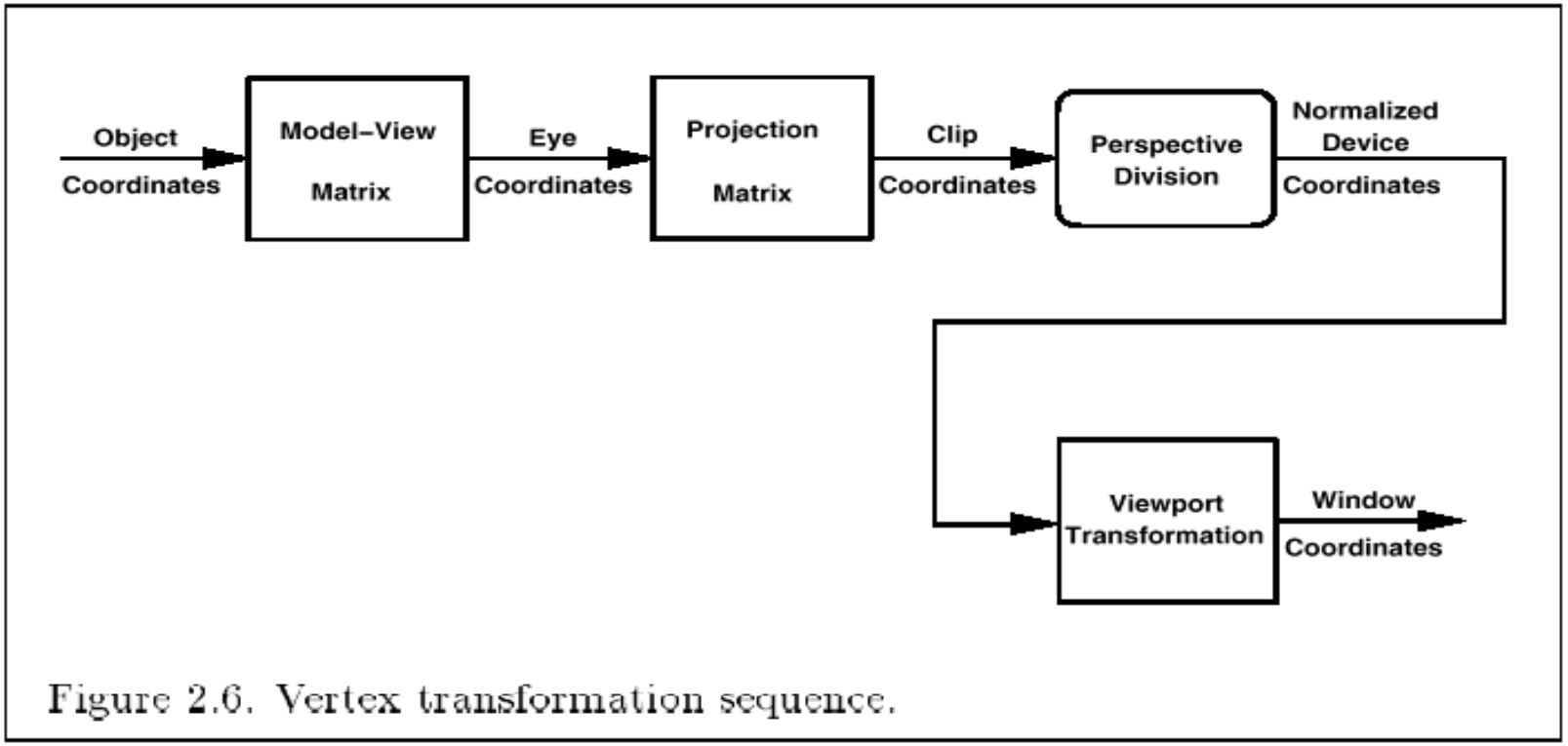
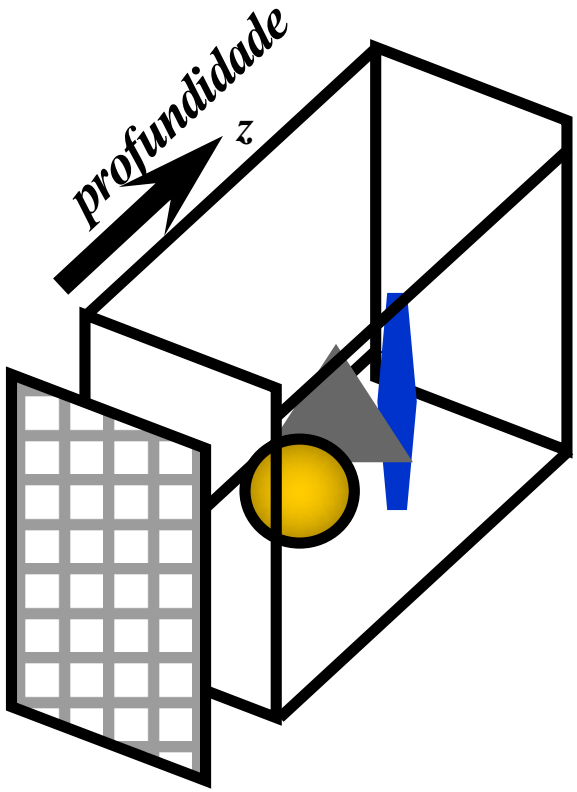
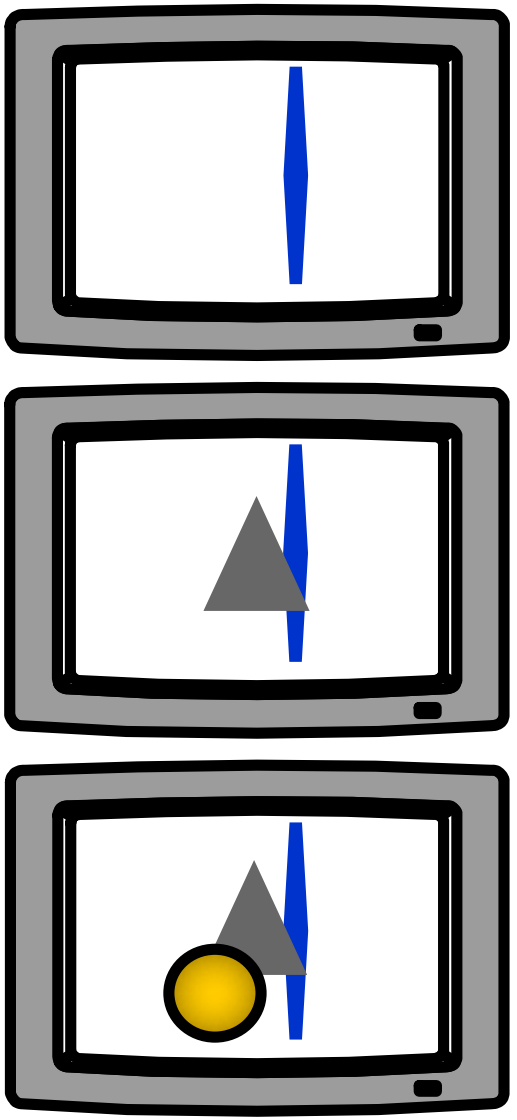


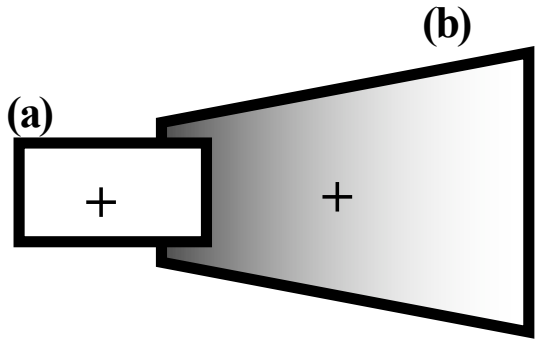
Figure 2.6. Vertex transformation sequence.

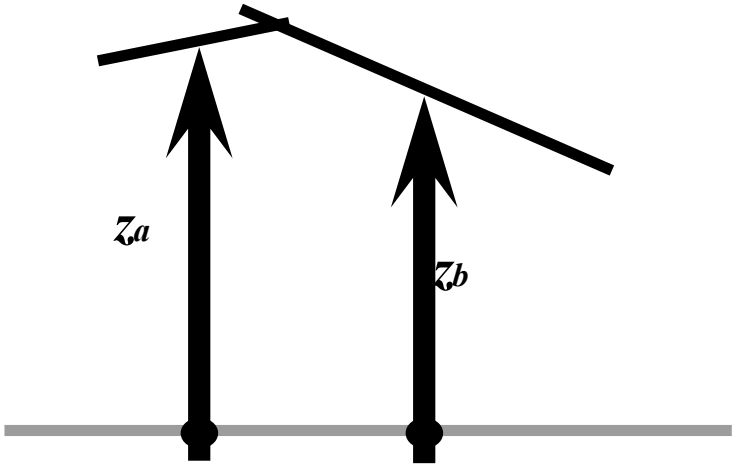
Modelo do Pintor

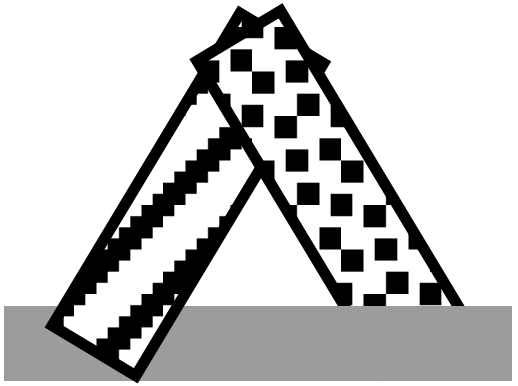


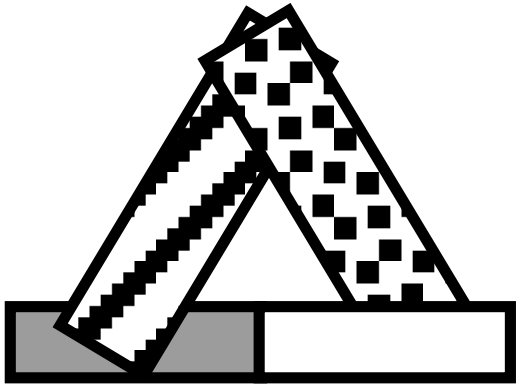


Problemas na ordenação de faces

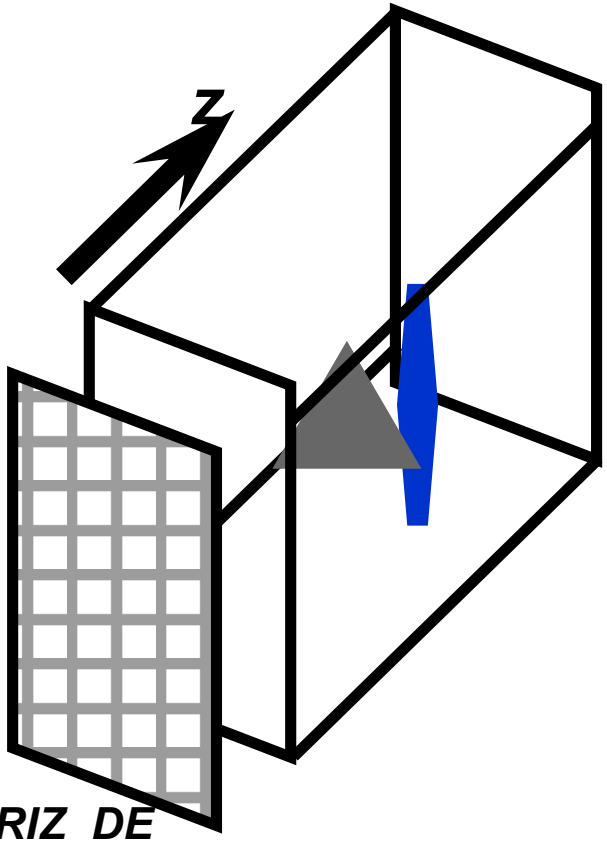




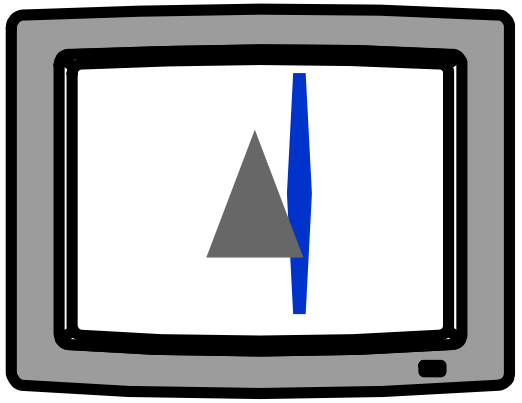




ZBuffer: idéia básica



MATRIZ DE PROFUNDIDADES

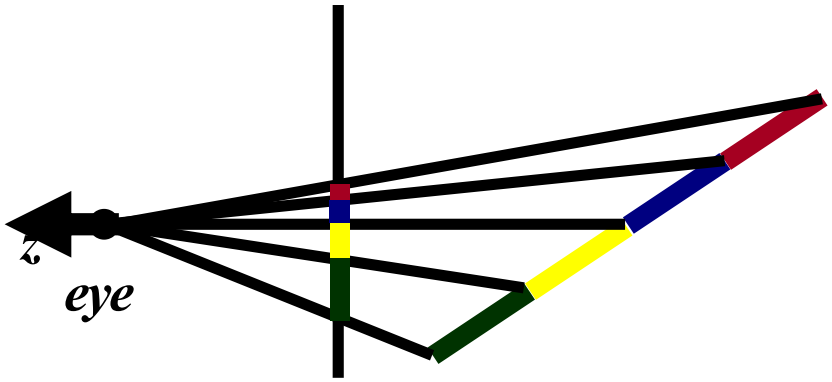


ZBuffer - pseudo-código

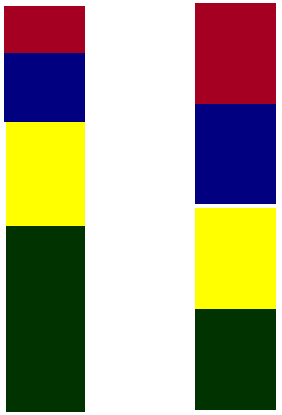
```
void ZBuffer( void)
{
 int x,y;

 for (x=0; x<w; x++) {
   for (y=0;y<h; y++) {
       WritePixel(x,y, bck_color);
       WriteZ(x,y,far+1);
   }
 }

  for (each primitive) {
    for (each pixel in the projected primitive) {
       double pz = z coordinate of the (x,y) pixel;
       if  (pz <= ReadZ(x,y)) {
          WritePixel(x,y, color);
          WriteZ(x,y,pz);
        }
    }
  }

} /* Zbuffer */
```

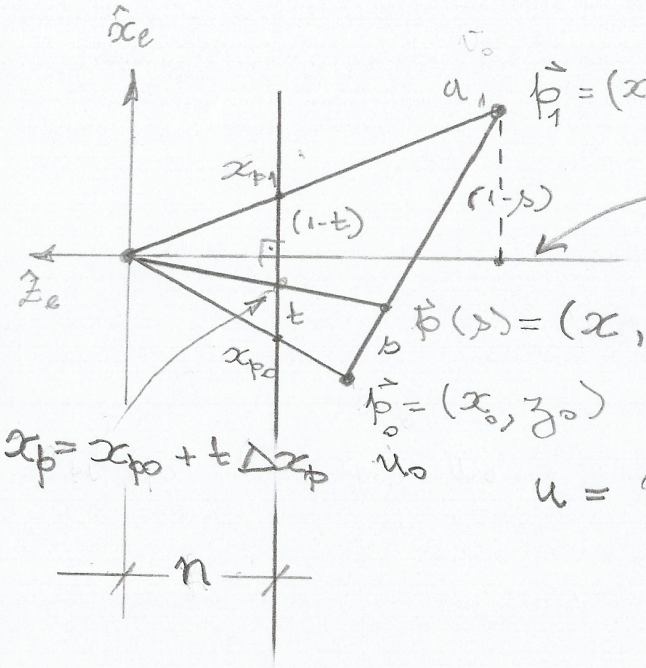
```
void glEnable( GL_DEPTH_TEST );
```

Interpolação perspectiva



plano de projeção





$$x_p = x_{p0} + t\Delta x_p \qquad u = u_0 + s\Delta u_0$$

$$\frac{x_1}{-z_1} = \frac{x_{p1}}{n} \quad (1) \quad \text{ou} \quad \frac{x}{-z} = \frac{x_p}{n}$$

$$(1) \rightarrow n x = -z x_p \rightarrow n(x_0 + s\Delta x) = -(z_0 + s\Delta z)(x_{p0} + t\Delta x_p)$$

$$n x_0 + s n \Delta x + z_0 x_{p0} + t z_0 \Delta x_p + s x_{p0} \Delta z + st \Delta z \Delta x_p = 0$$

$$(1) \rightarrow \boxed{x_0 = \frac{x_{p0} z_0}{-n}} \qquad \boxed{\Delta x = \frac{x_{p1} z_1 - x_{p0} z_0}{-n}} \leftarrow (1)$$

$$-x_{p0} z_0 - s(x_{p1} z_1 - x_{p0} z_0) + z_0 x_{p0} + t z_0 \Delta x_p + s x_{p0}(z_1 - z_0) + st \Delta z \Delta x_p = 0$$

$$-s z_1 \Delta x_p + t z_0 \Delta x_p + st \Delta z \Delta x_p = 0 \Rightarrow s(z_1 + t\Delta z) = t z_0$$

$$\boxed{s = \frac{t z_0}{z_1 - t\Delta z}}$$

$$z = z_0 + t\Delta z = z_0 + \frac{t z_0}{z_1 - t\Delta z}\Delta z$$

$$= \frac{z_0(z_1 - t\Delta z) + t z_0 \Delta z}{z_1 - t\Delta z} = \frac{z_0 z_1}{z_1 - t(z_1 - z_0)}$$

$$= \frac{1}{\frac{z_1}{z_0 z_1} - t\left(\frac{z_1}{z_0 z_1} - \frac{z_0}{z_0 z_1}\right)} = \frac{1}{\frac{1}{z_0} - t\left(\frac{1}{z_0} - \frac{1}{z_1}\right)} = \frac{1}{\frac{1}{z_0} + t\left(\frac{1}{z_1} - \frac{1}{z_0}\right)}$$

$$\boxed{\frac{1}{z} = \frac{1}{z_0} + t\left(\frac{1}{z_1} - \frac{1}{z_0}\right)}$$

$$u = u_0 + t\Delta u = u_0 + \frac{t z_0}{z_1 - t\Delta z}\Delta u$$

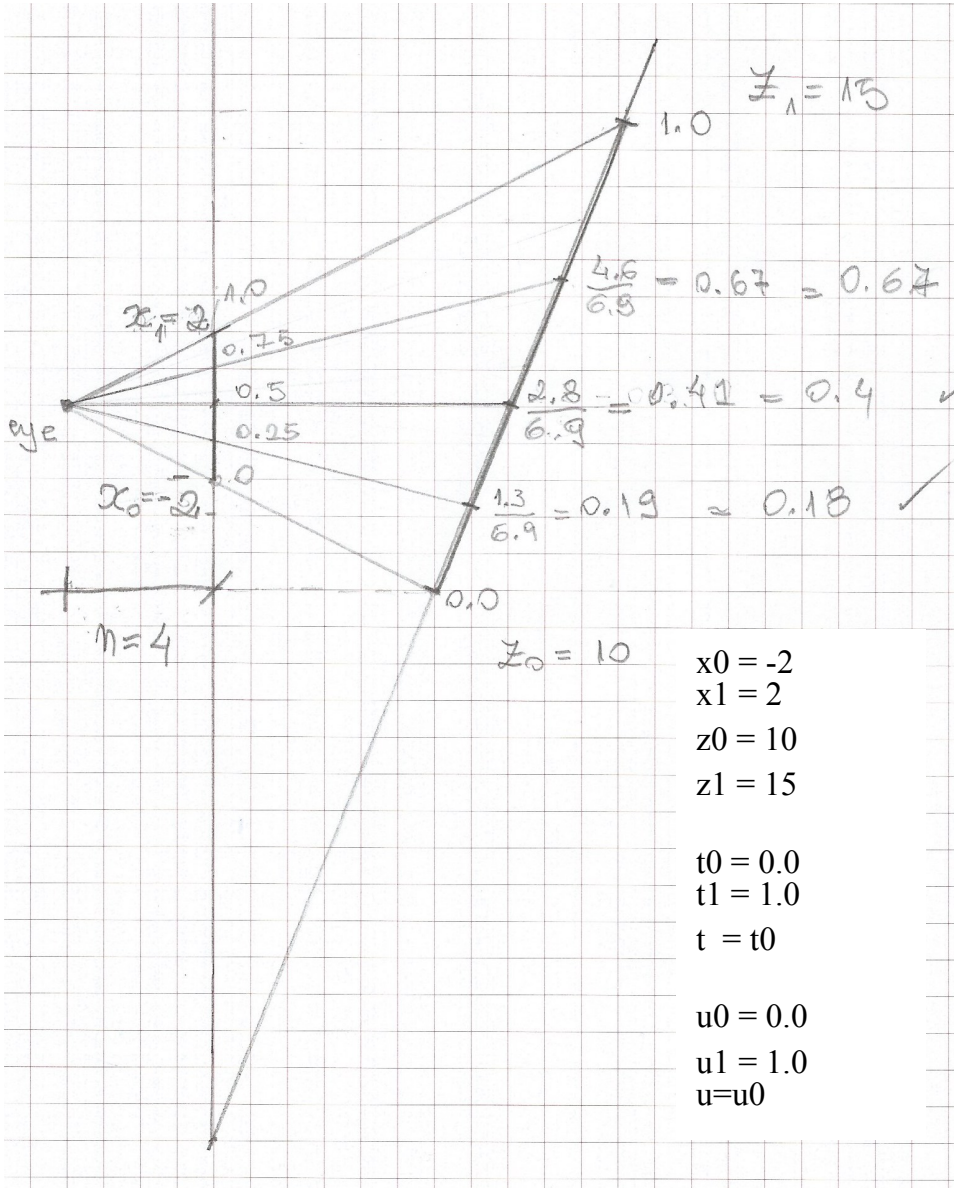
$$= \frac{u_0(z_1 - t\Delta z) + t z_0 \Delta u}{z_1 - t\Delta z} \times \frac{\frac{1}{z_0 z_1}}{\frac{1}{z_0 z_1}}$$

$$= \frac{\frac{u_0}{z_0} - \frac{t u_0 \Delta z}{z_0 z_1} + \frac{t z_0 \Delta u}{z_0 z_1}}{\frac{1}{z_0} + t\left(\frac{1}{z_1} - \frac{1}{z_0}\right)} \quad \leftarrow (A)$$

$$A = -t u_0\left(\frac{1}{z_0} - \frac{1}{z_1}\right) + t\frac{1}{z_1}u_1 - \frac{t u_0}{z_1} = t\left(\frac{u_1}{z_1} - \frac{u_0}{z_0}\right)$$

$$u = \frac{\frac{u_0}{z_0} + t\left(\frac{u_1}{z_1} - \frac{u_0}{z_0}\right)}{\frac{1}{z_0} + t\left(\frac{1}{z_1} - \frac{1}{z_1}\right)}$$

Interpolação perspectiva



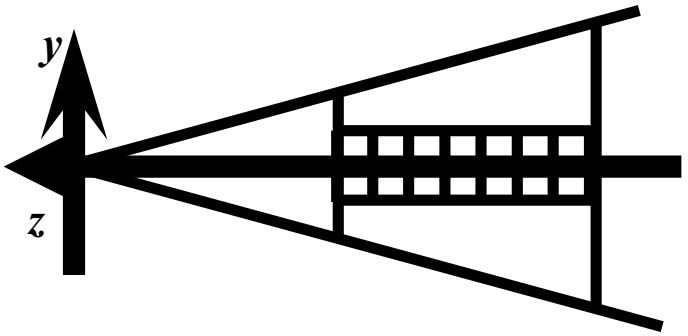

```
x0 = -2
x1 = 2
z0 = 10
z1 = 15

t0 = 0.0
t1 = 1.0
t  = t0

u0 = 0.0
u1 = 1.0
u=u0
```

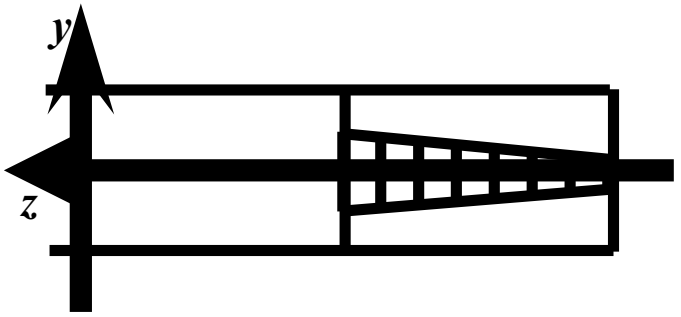
```
uz0 = u0/z0
uz1 = u1/z1
uz = uz0

zi0 = 1./z0
zi1 = 1./z1
zi = zi0

duz = (uz1-uz0)/(x1-x0)
dzi = (zi1-zi0)/(x1-x0)
dt  = (t1-t0)/(x1-x0)
z = z0
for x in xrange(x0,x1+1):
    print t,z,u
    uz+=duz
    zi += dzi
    z = 1./zi
    u = uz*z
    t+=dt
```

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

t	z	s
0.00	10	0.00
0.25	10.9	0.18
0.50	12.0	0.40
0.75	13.3	0.67
1.0	15.0	1.0

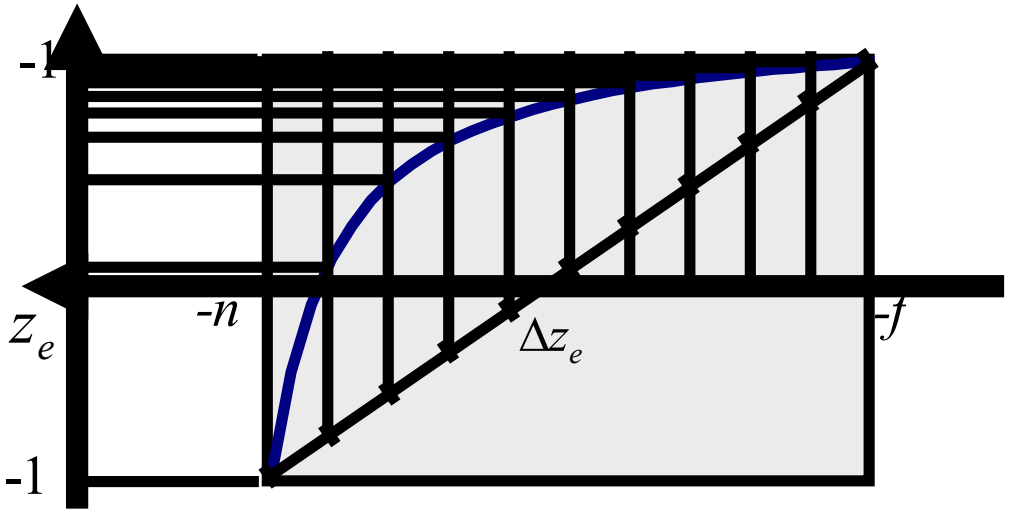
Uma outra forma de ver a interpolação da profundidade

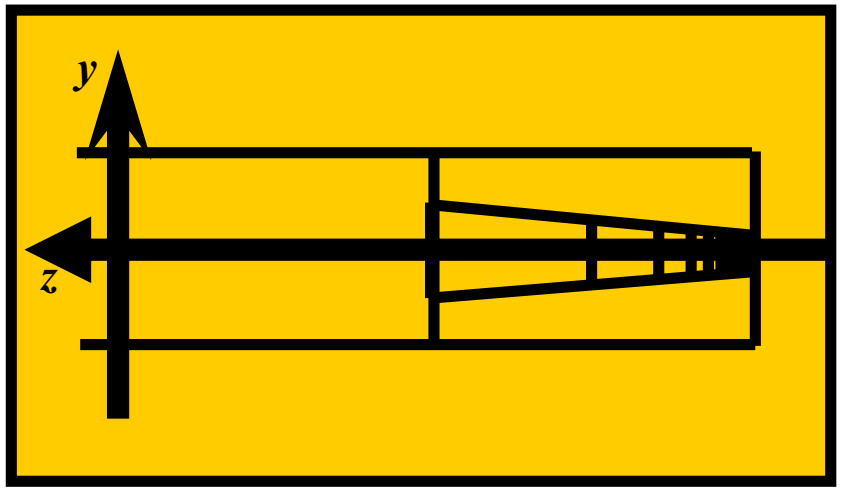




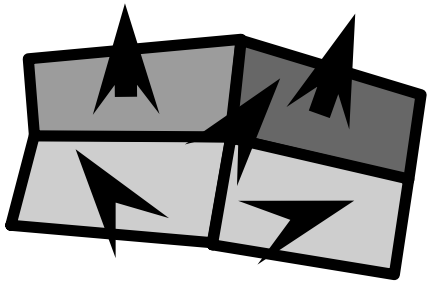
☹

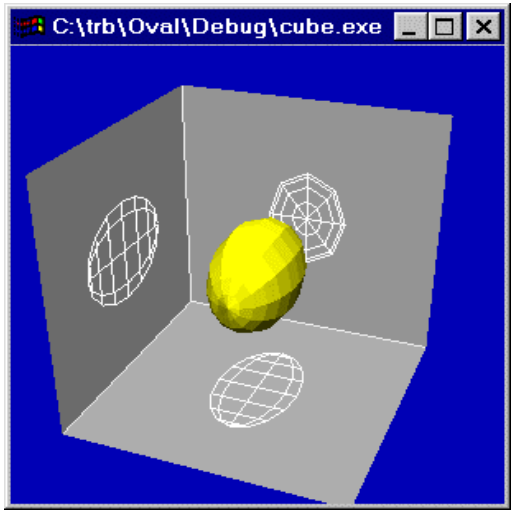
$$z_n = \frac{z_e}{w} = \frac{f+n}{(f-n)} - \frac{2fn}{(f-n)}\frac{1}{z_e}$$





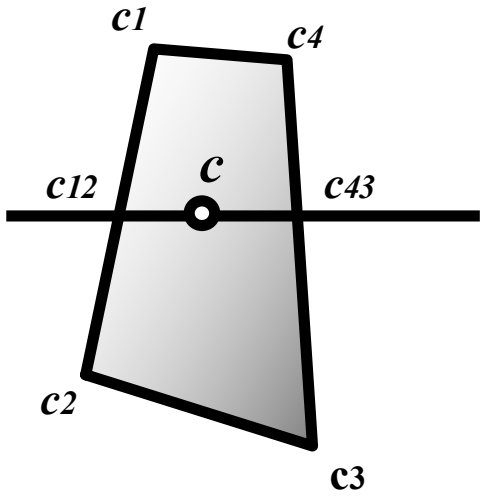
Suavização da tonalização

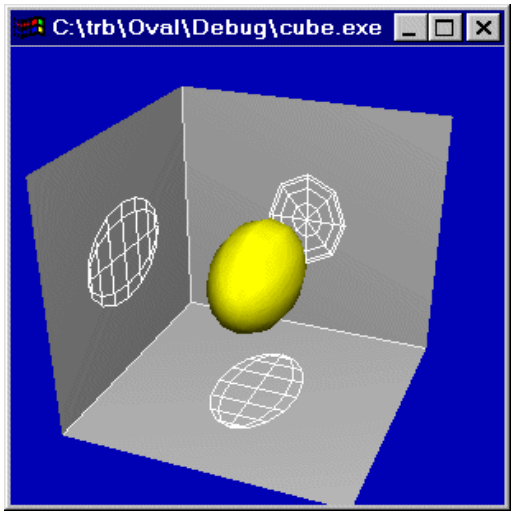




void **glShadeModel** (**GL_FLAT**);

Gouraud





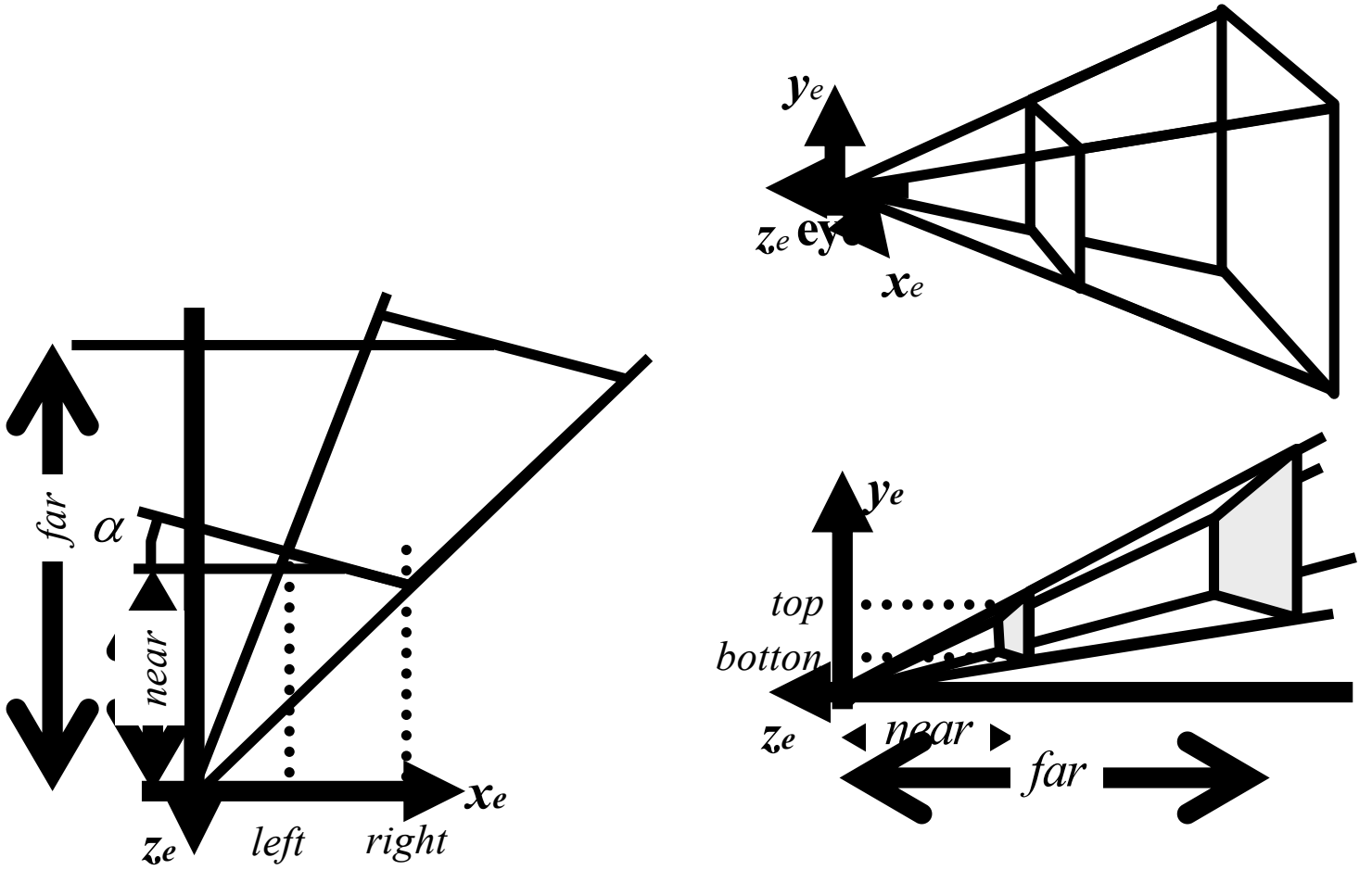
void **glShadeModel** (**GL_SMOOTH**);

Algortimo de ZBuffer

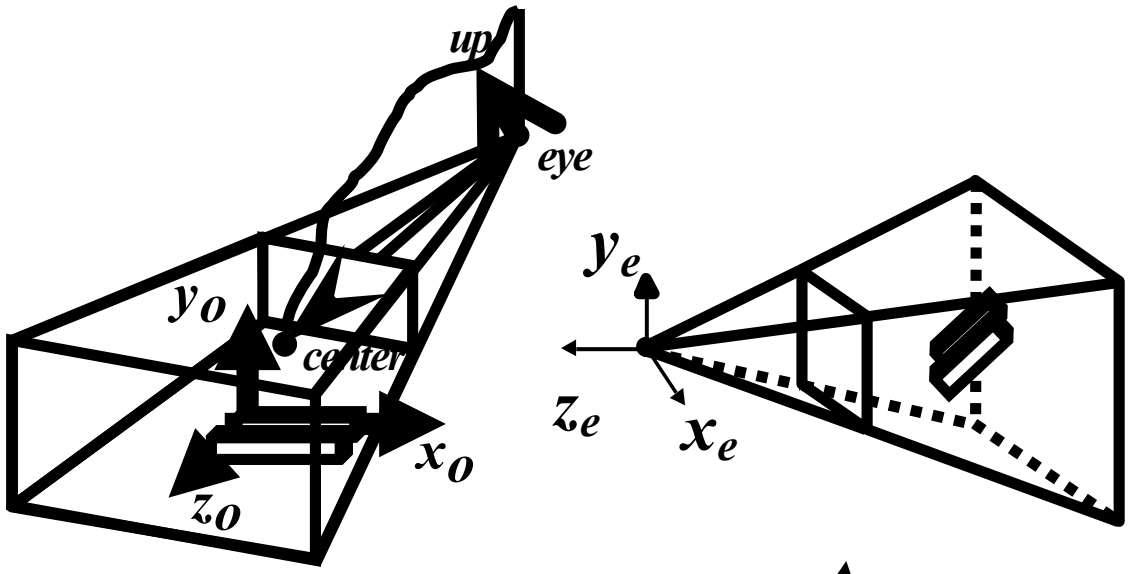
```
void ZBuffer( void)
{
 for (cada primitva gráfica dos objetos da cena) {
    for (cada fragmento gerado pelo rastreio da primitiva) {
       if (o fragmento não pertencer a janela) break; /* pertinencia */
       if (o fragmento não pertencer ao retangulo de  janela de recorte) break; /* scissor test */
       if (a profundidade do fragmento for maior que a corrente) break; /* profundidade */

       . . .
         WritePixel(x,y, color);
         WriteZ(x,y,pz);
        }
    }
  }

}
```

Exercício 1

Derive a matriz de projeção da seguinte câmera

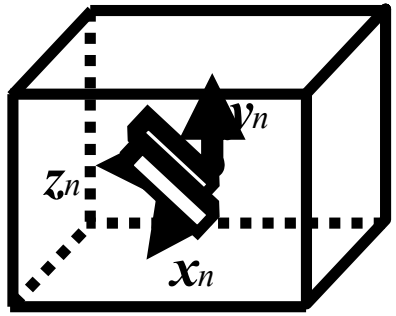


Resumo das transformações
(até o momento)



III.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix}$$



$$\begin{pmatrix} x_o \\ y_o \\ z_o \end{pmatrix}$$

entrada

I.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = [\mathbf{M}_{view}] \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

II.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix} = [\mathbf{P}] \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$

$$\frac{1}{w} \begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix}$$

IV.

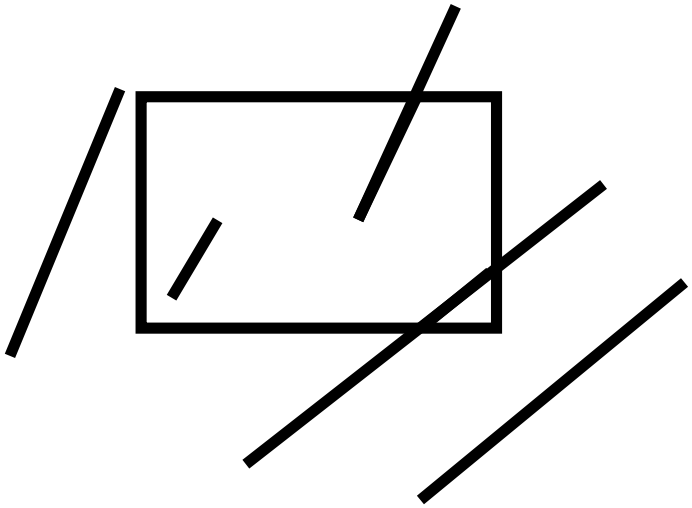
$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}$$

saída

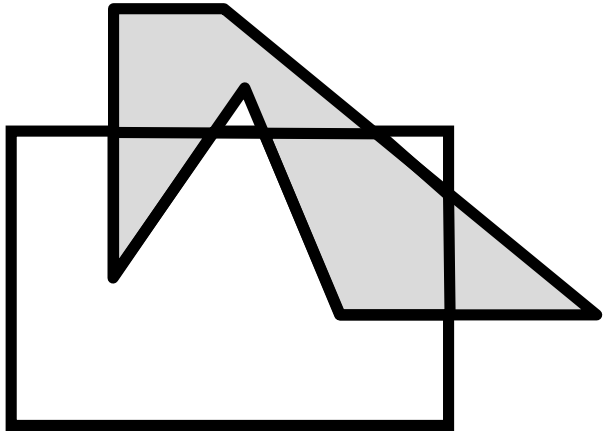
Recorte

Diferentes problemas de recorte

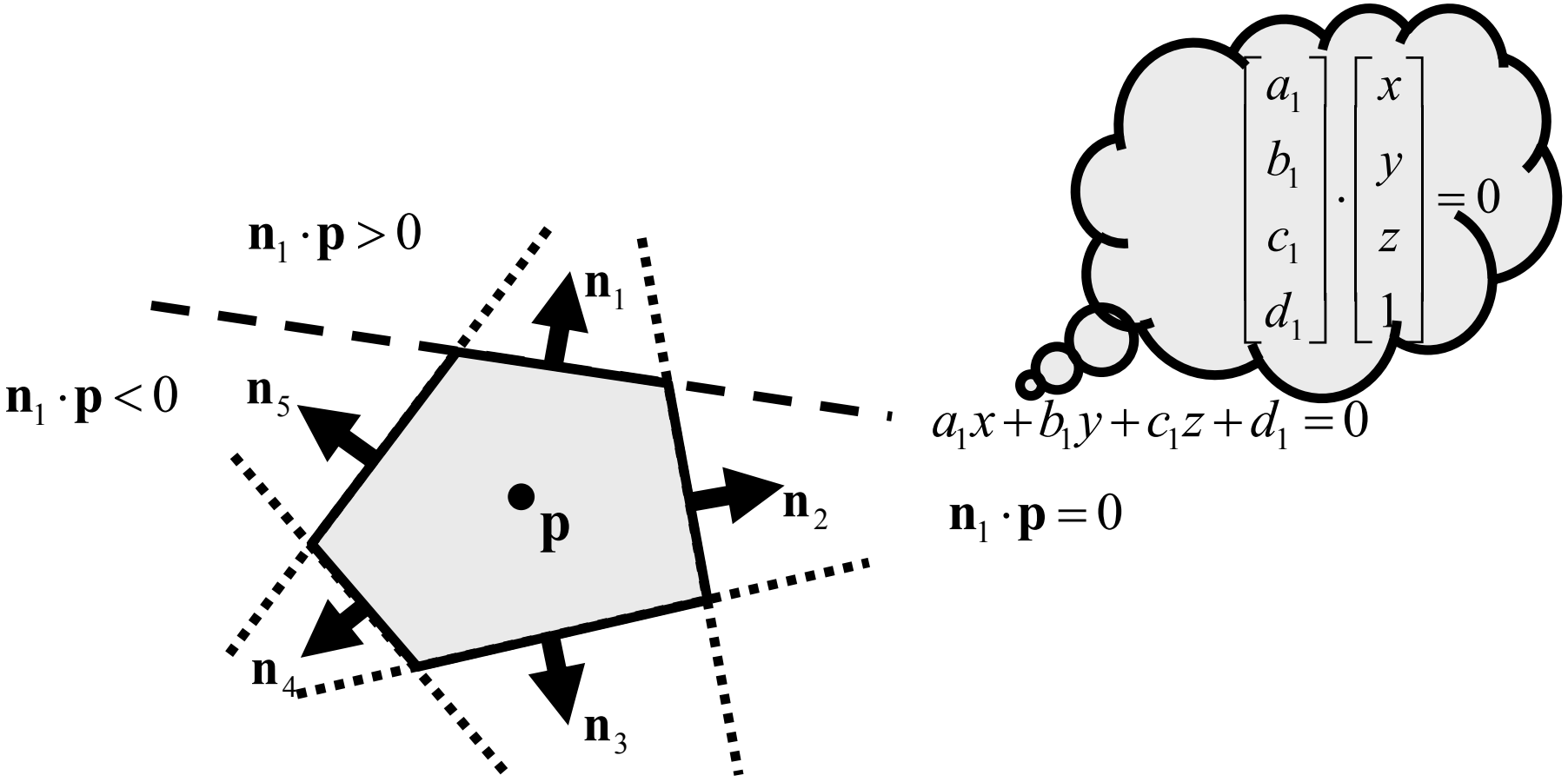
linhas



polígonoss

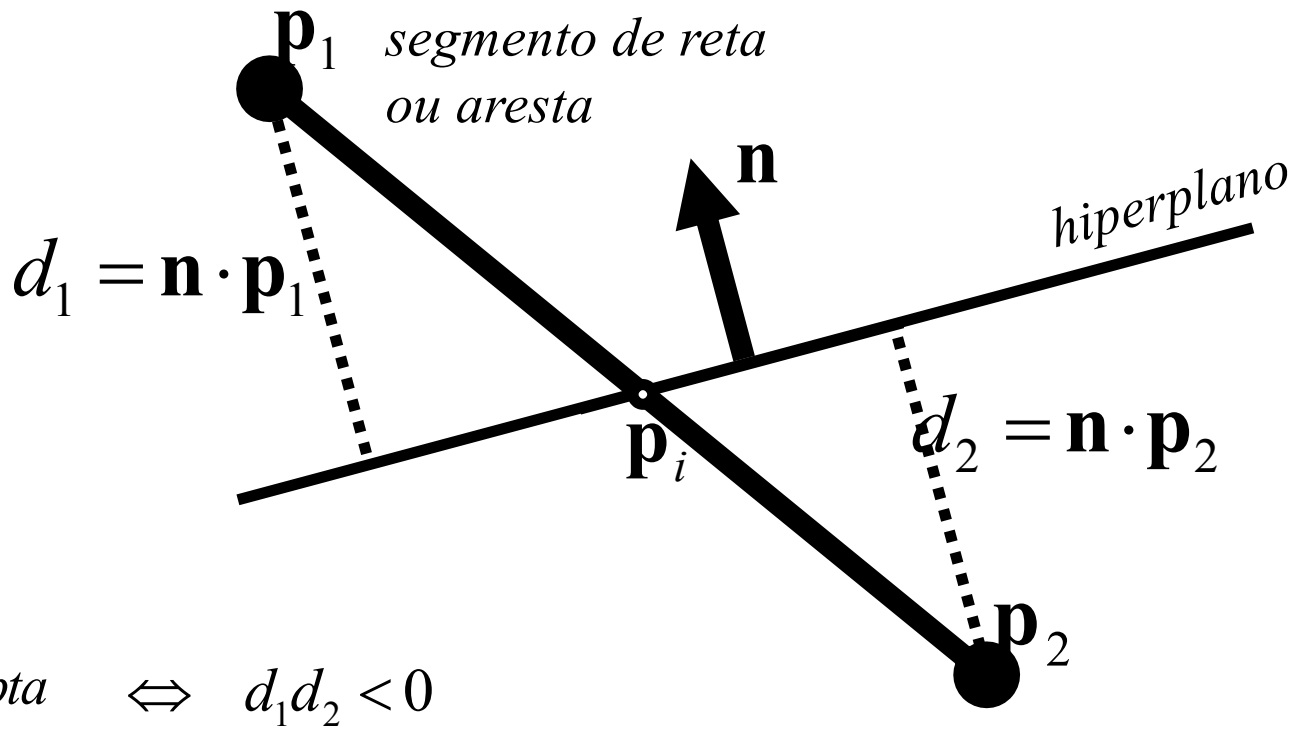


Condição de pertinência



$\mathbf{p}$ é interior $\Leftrightarrow \mathbf{n}_i \cdot \mathbf{p} < 0 \quad \forall i$

Interseção de aresta × hiperplano

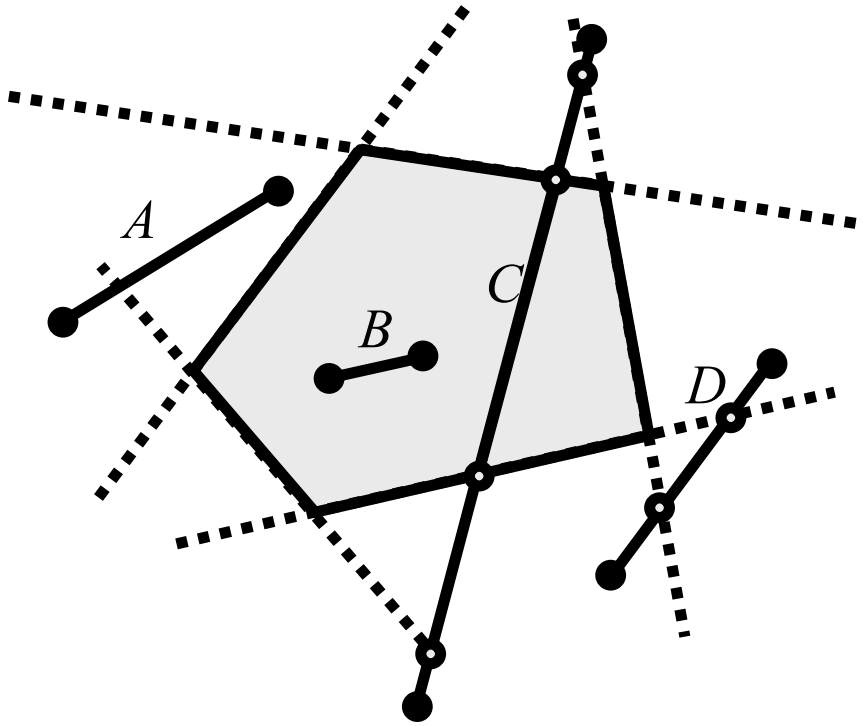


intercepta $\Leftrightarrow \quad d_1 d_2 < 0$

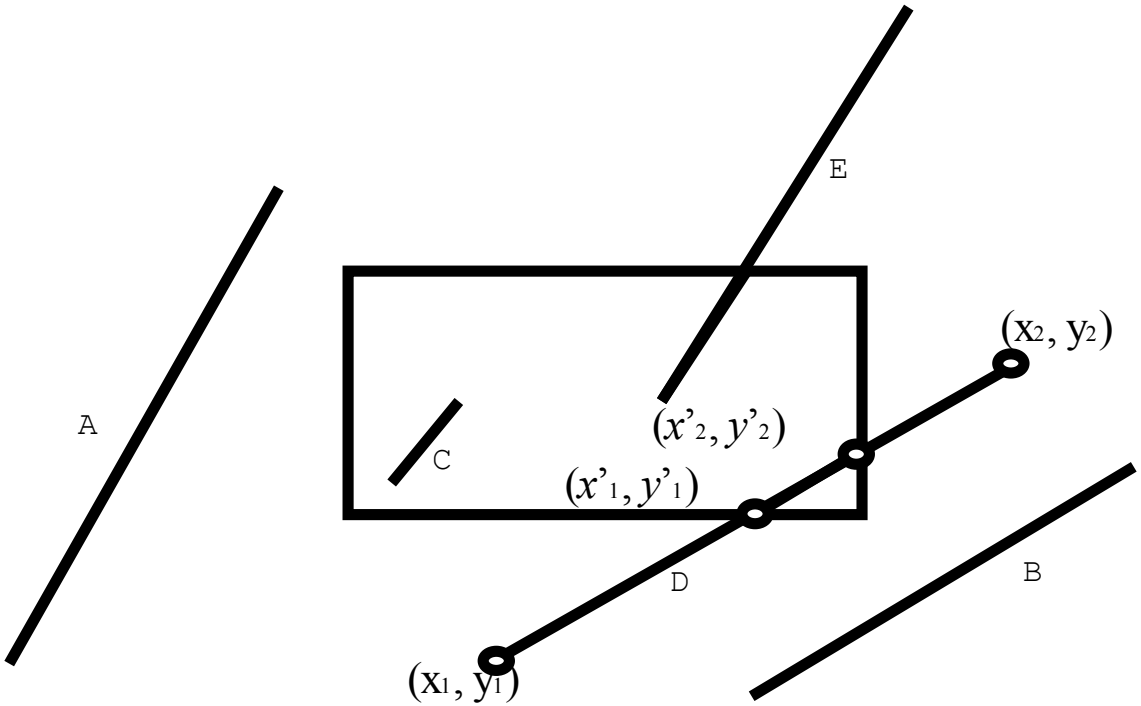
$$\mathbf{p}_i = \frac{|d_1|}{|d_1|+|d_2|}\mathbf{p}_2 + \frac{|d_2|}{|d_1|+|d_2|}\mathbf{p}_1$$

$$\mathbf{p}_i = \frac{d_1}{d_1-d_2}\mathbf{p}_2 - \frac{d_2}{d_1-d_2}\mathbf{p}_1$$

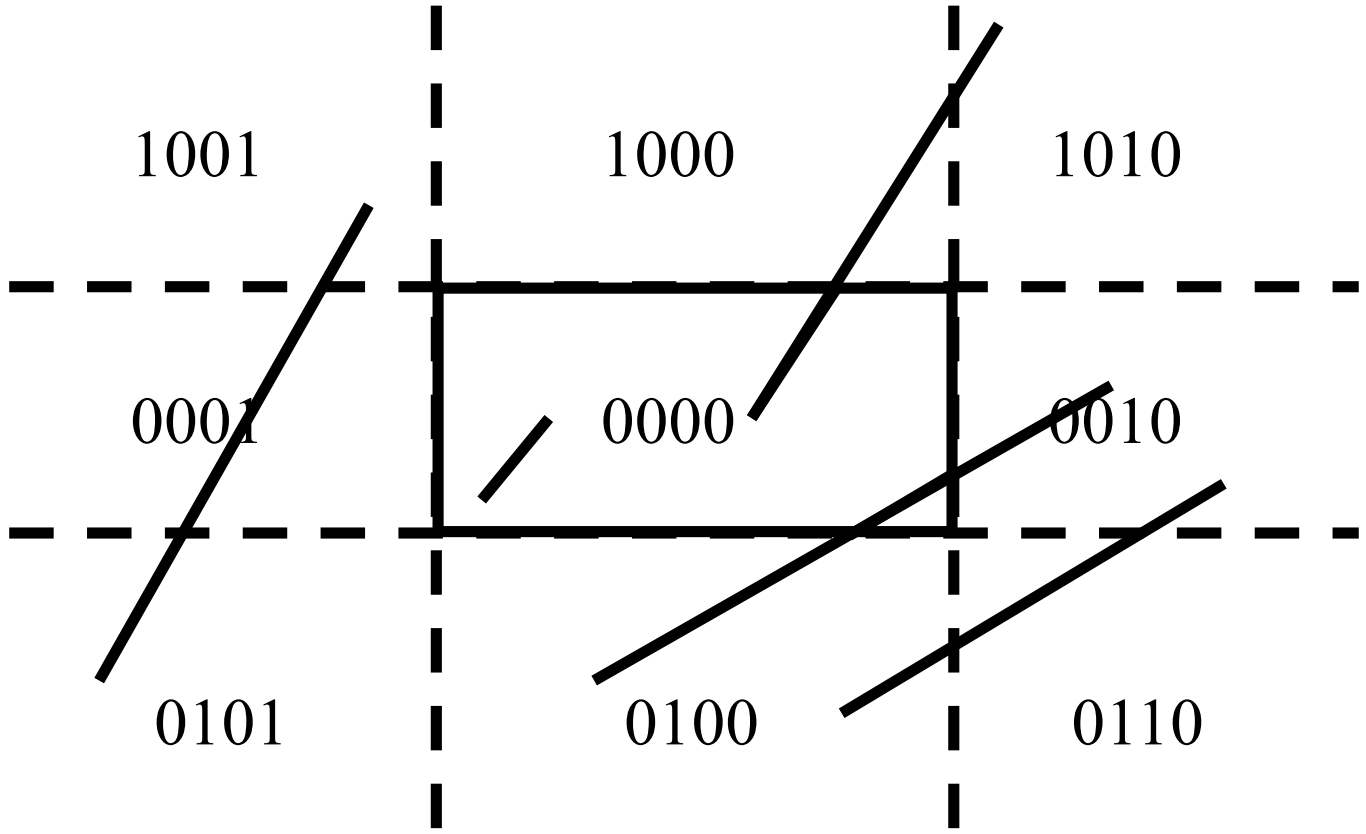
Recorte de segmentos com base no algoritmo Cohen-Sutherland



Casos de *clipping* de linhas



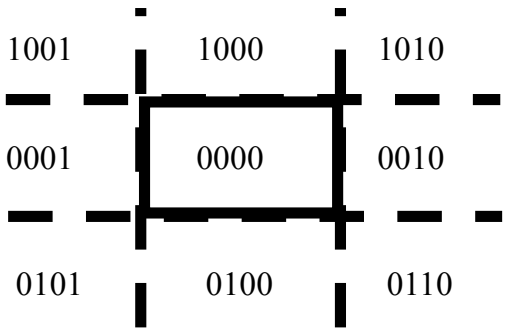
Códigos das regiões



tbrl

Cálculo do código de um vértice

```
unsigned char code(double x, double y,
                   double xmin, double xmax, double ymin, double ymax)
{
   unsigned char code=0;

   if (y > ymax) code += 8;
   if (y < ymin) code += 4;
   if (x > xmax) code += 2;
   if (x < xmin) code += 1;

   return code;
}
```

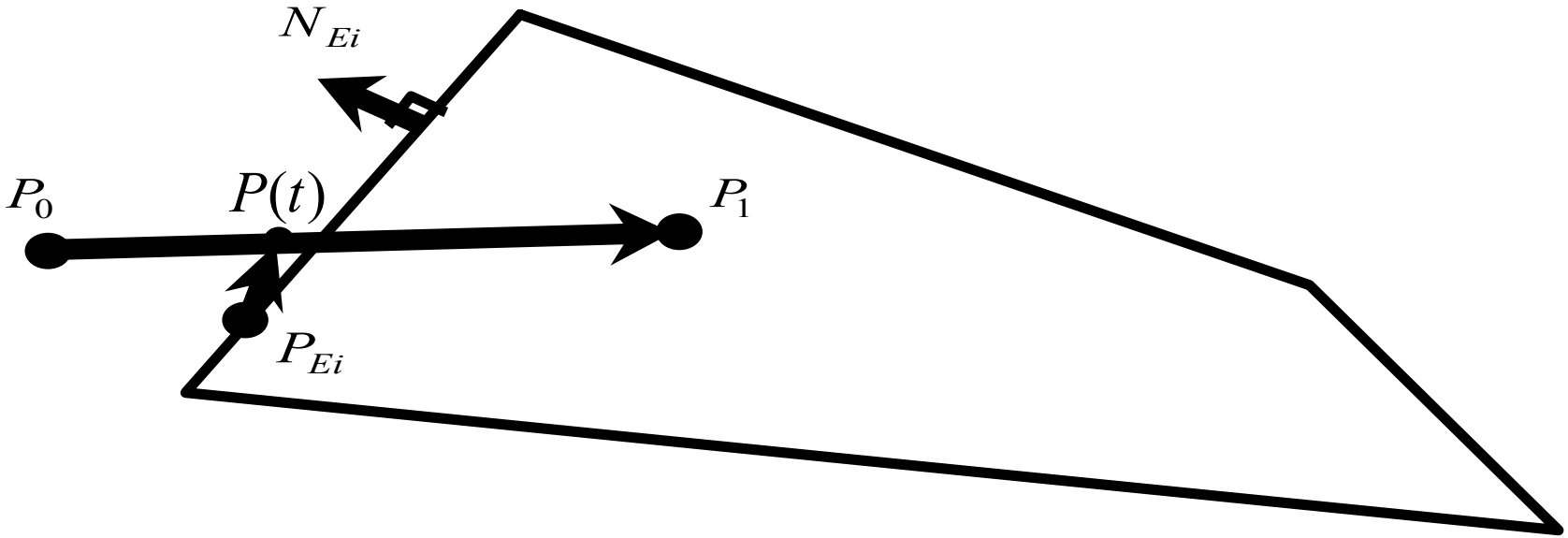


tbrl

```
void CohenSutherlandLineClip(double x0, double y0, double x1, double y1,
                             double xmin, double xmax, double ymin, double ymax)
{
 unsigned char outcode0, outcode1, outcodeOut;
 double x, y;  boolean accept = FALSE, done = FALSE;

 outcode0 = code(x0, y0, xmin, xmax, ymin, ymax);
 outcode1 = code(x1, y1, xmin, xmax, ymin, ymax);

 do {
   if (outcode0 == 0 && outcode1 == 0) {
       accept = TRUE;  done = TRUE;                /* trivial draw and exit */
   } else if((outcode0 & outcode1) != 0) {
       done = TRUE;                                /* trivial reject and exit */
   } else {                                        /* discart an out part */
     outcode = (outcode0 != 0) ?  outcode0 : outcode1;          /* pick an out vertice */
     if (outcodeOut & 8) {                                      /* discart top */
       x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);  y = ymax;
     } else if(outcodeOut & 4) {                                /* discart bottom */
       x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);  y = ymin;
     } else if(outcodeOut & 2) {                                /* discart right */
       y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);  x = xmax;
     } else if(outcodeOut & 1) {                                /* discart left */
       y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);  x = xmin;
     }

     if (outcodeOut == outcode0) {
        x0 = x; y0 = y; outcode0 = code(x0, y0, xmin, xmax, ymin, ymax);
     } else {
        x1 = x; y1 = y;  outcode1 = code(x1, y1, xmin, xmax, ymin, ymax);
     }

   }
  } while (!done);

   if (accept) DrawLineReal(x0, y0, x1, y1);
}
```

Algoritmo de Cohen & Sutherland

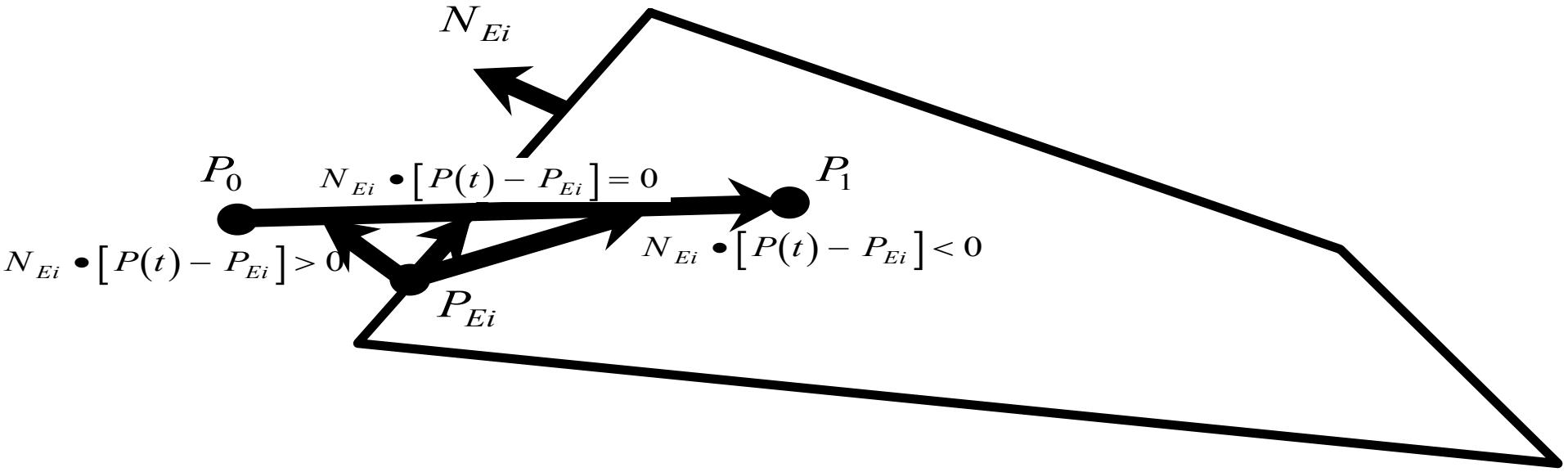
Algoritimo de Cyrus-Beck (Liang-Barsky)



$$P(t) = P_0 + (P_1 - P_0)t$$

$$N_{Ei} \bullet [P(t) - P_{Ei}] = 0$$

Algoritimo de Cyrus-Beck (Liang-Barsky)

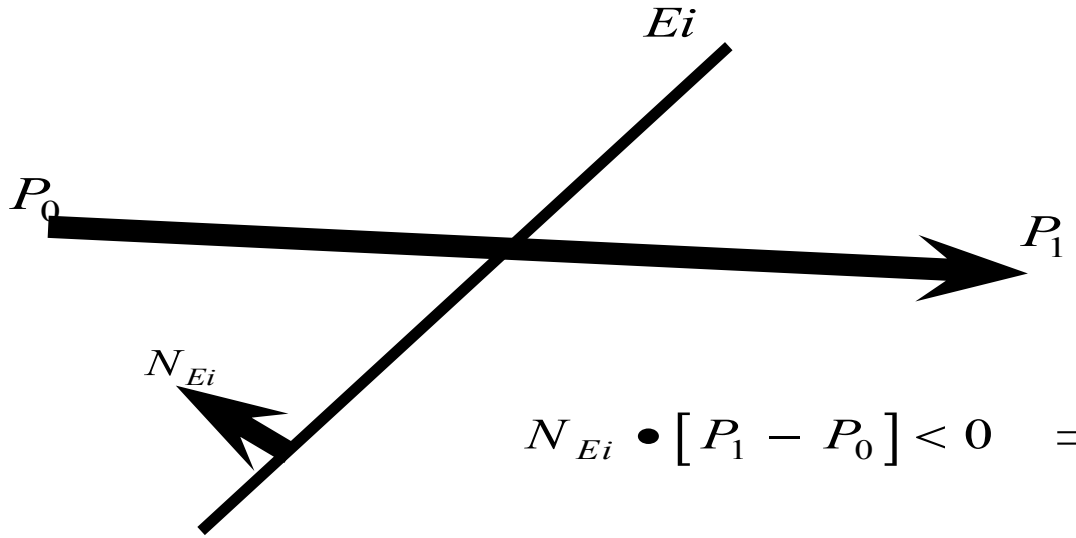


$$P(t) = P_0 + (P_1 - P_0)t$$

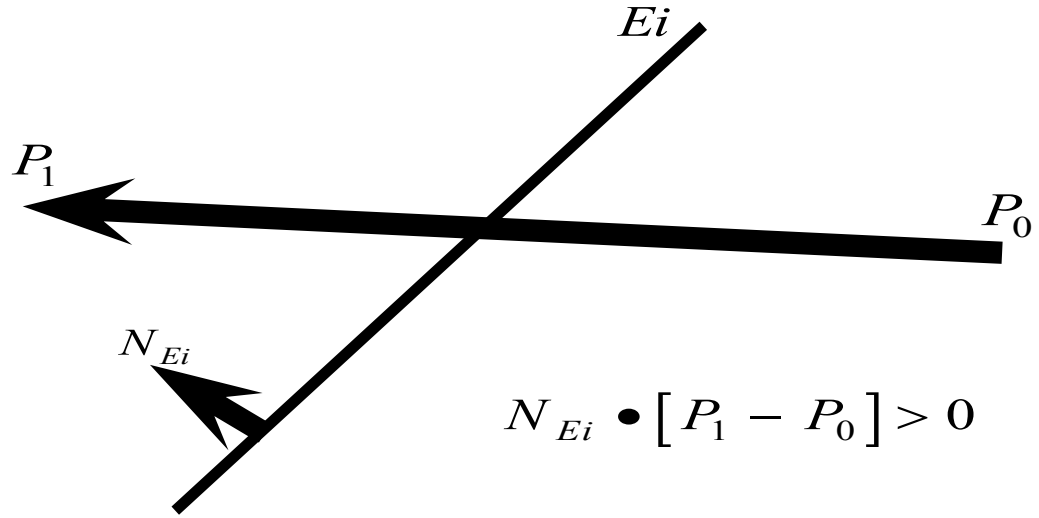
$$N_{Ei} \bullet [P(t) - P_{Ei}] = 0$$

$$t = \frac{N_{Ei} \bullet [P_0 - P_{Ei}]}{-N_{Ei} \bullet [P_1 - P_0]}$$

Entrando ou Saindo ?

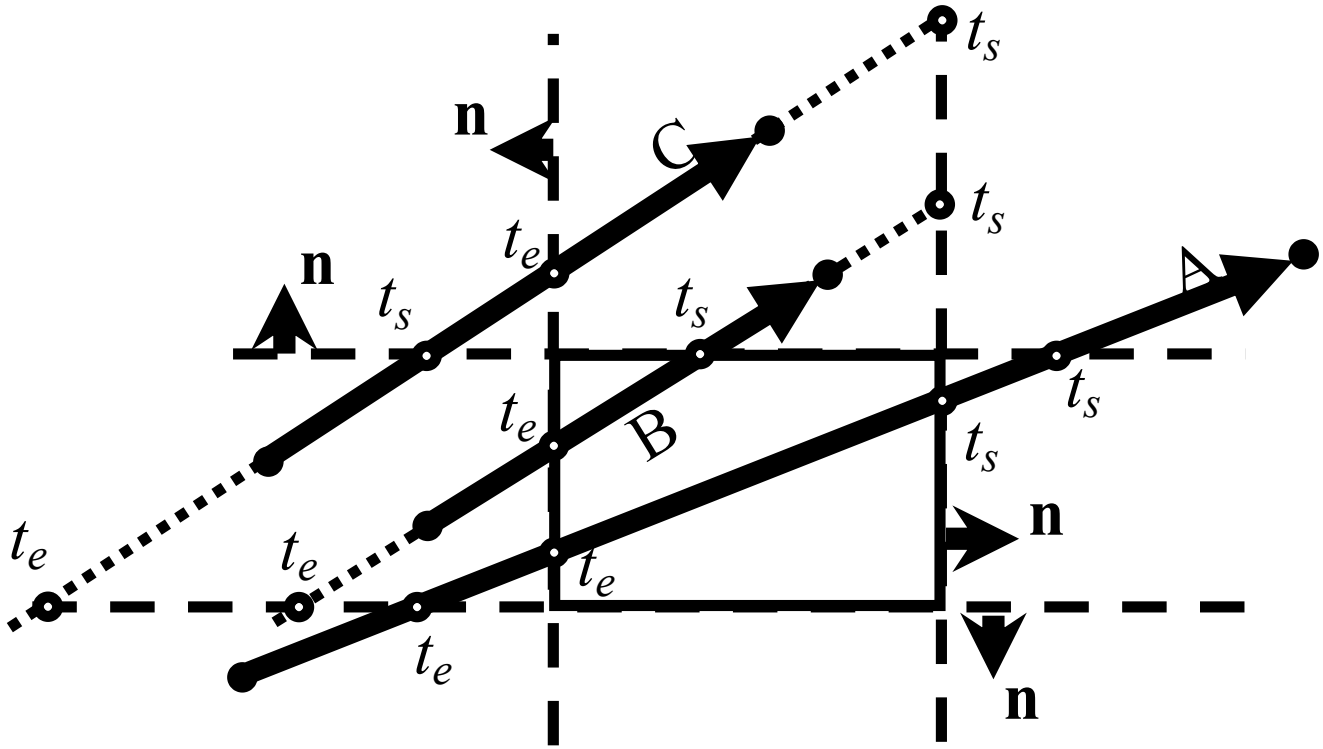


$$N_{Ei} \bullet [P_1 - P_0] < 0 \quad \Rightarrow \quad PE$$



$$N_{Ei} \bullet [P_1 - P_0] > 0 \quad \Rightarrow \quad PL$$

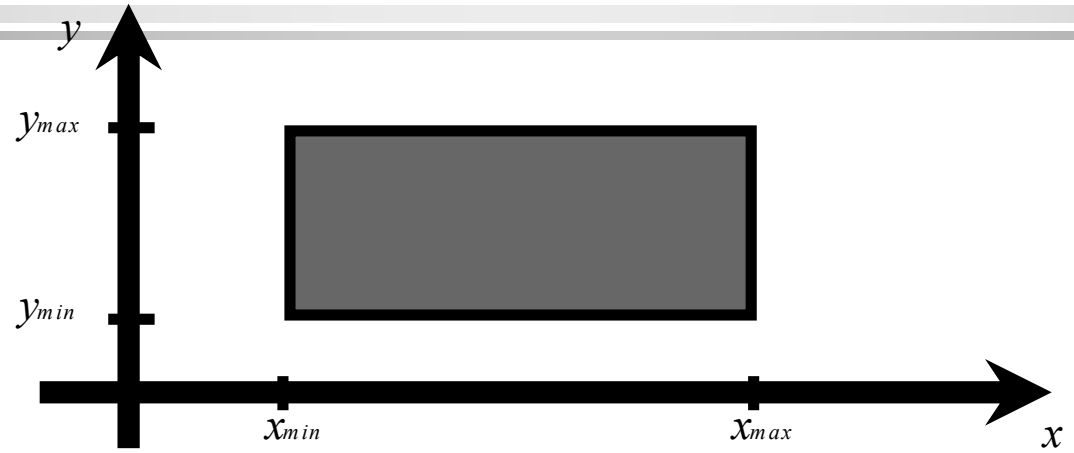
Recorte de segmentos com base no algoritmo de Cyrus-Beck



Cyrus-Beck - caso geral

```
{
   Calcule Ni e escolha um PEi para cada aresta

     tE = 0;
     tL = 1;
     for (cada aresta ){
        if (Ni.(P1-P0)!=0 ){ /* aresta não é paralela ao segmento */
           calcule t;
           use sign of Ni.(P1-P0) para categorizar como PE ou PL;
               if( PE ) tE = max(tE, t);
               if( PL ) tL = min(tL, t);
        } else {   /* aresta paralela ao segmento */
               if (Ni.(P0-PEi) > 0) /* está fora */
                   return nil;
        }
        if (tE > tL)
            return nil;
        else
            return P(tE) and P(tL) as true clip intersections;
      }
   }
}
```

Liang e Barsky
- caso particular -



Ei	N_{Ei}	P_{Ei}	t
left: $x = x_{min}$	(-1, 0)	(x_{min}, y)	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
right: $x = x_{max}$	(1, 0)	(x_{max}, y)	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
bottom: $y = y_{min}$	(0,-1)	(x, y_{min})	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: $y = y_{max}$	(0, 1)	(x, y_{max})	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

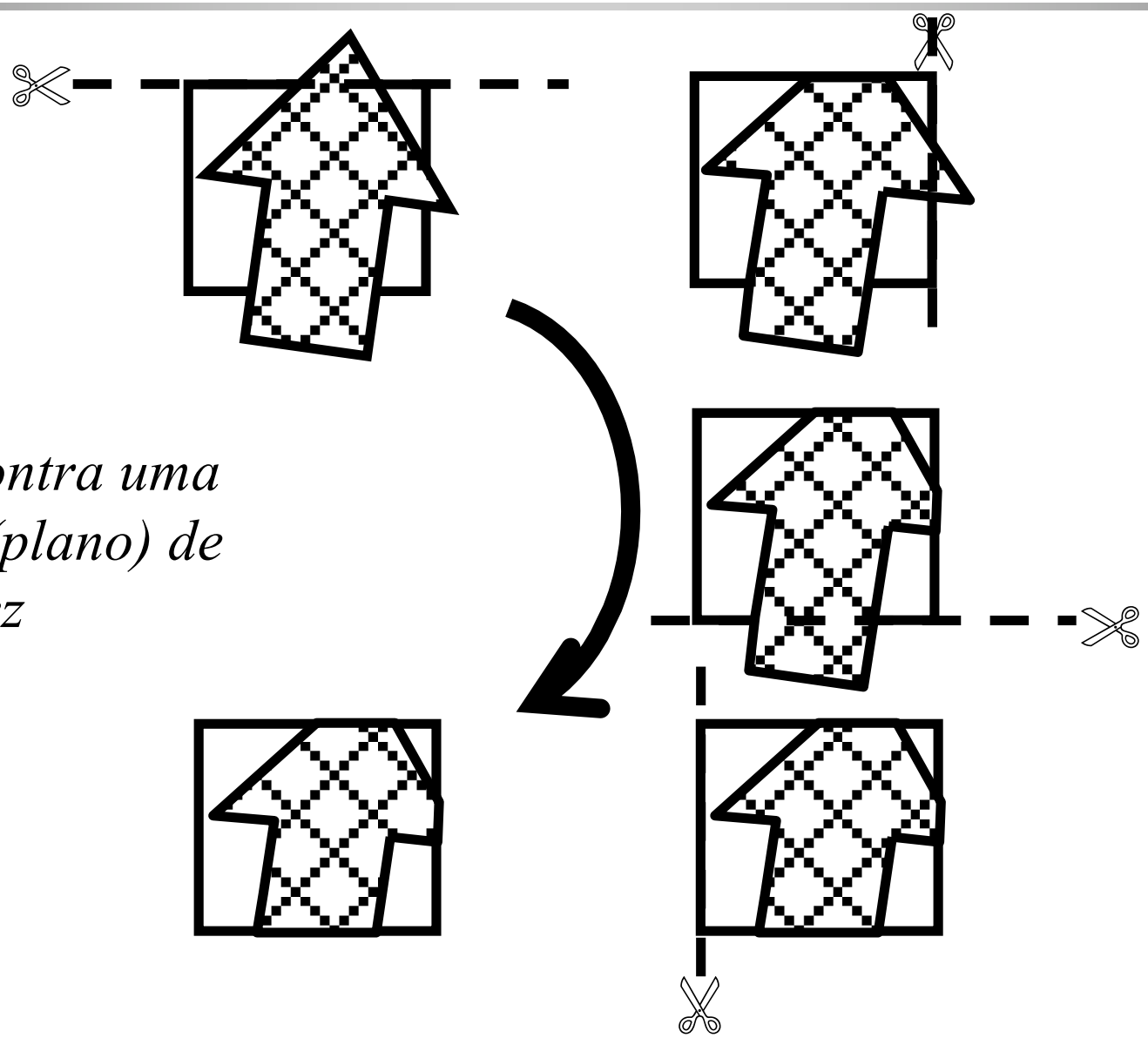
Liang e Barsky
- Cálculo da interseção em uma fronteira -

```
boolean Clipt(double den, double num, double *tE, double *tL)
{
 double t;

 if (den > 0)                  /* intersecao PE */
  {
   t = num/den;
   if (t > *tL)                /* tE > tL */
     return FALSE;
   else
     if (t > *tE) *tE = t;
  }
 else if (den < 0)             /* intersecao PL */
  {
   t=num/den;
   if (t < *tE)                /* tL < tE */
     return FALSE;
   else
     if (t < *tL) *tL = t;
  }
 else
   if (num > 0)                /* linha esta fora */
      return FALSE;
 }
 return TRUE;
}
```

Algoritimo de Liang e Barsky

```
boolean Clip2D(double *x0, double *y0, double *x1, double *y1,
               double xmin, double max, double ymin, double ymax)
{
 double  dx = *x1 - *x0;
 double  dy = *y1 - *y0;
 boolean visible = FALSE;

 if (dx==0)&&(dy==0)&&ClipPoint(*x0,*y0,xmin,xmax,ymin,ymax)
  visible=TRUE;
 else
 {
  double tE=0.0, tL=1.0;

  if ( Clipt(dx,xmin-*x0,&tE,&tL) )
    if ( Clipt(-dx,*x0-max,&tE,&tL) )
      if ( Clipt(dy,ymin-*y0,&tE,&tL) )
        if ( Clipt(-dy,*y0-ymax,&tE,&tL) ) {
           visible=TRUE;
           if (tL<1.0) { (*x1)=(*x0)+tL*dx; (*y1)=(*y0)+tE*dy; }
           if (tE>0.0) { (*x0)=(*x0)+tE*dx; (*y0)=(*y0)+tE*dy; }
           }
 }
 return visible;
}
```

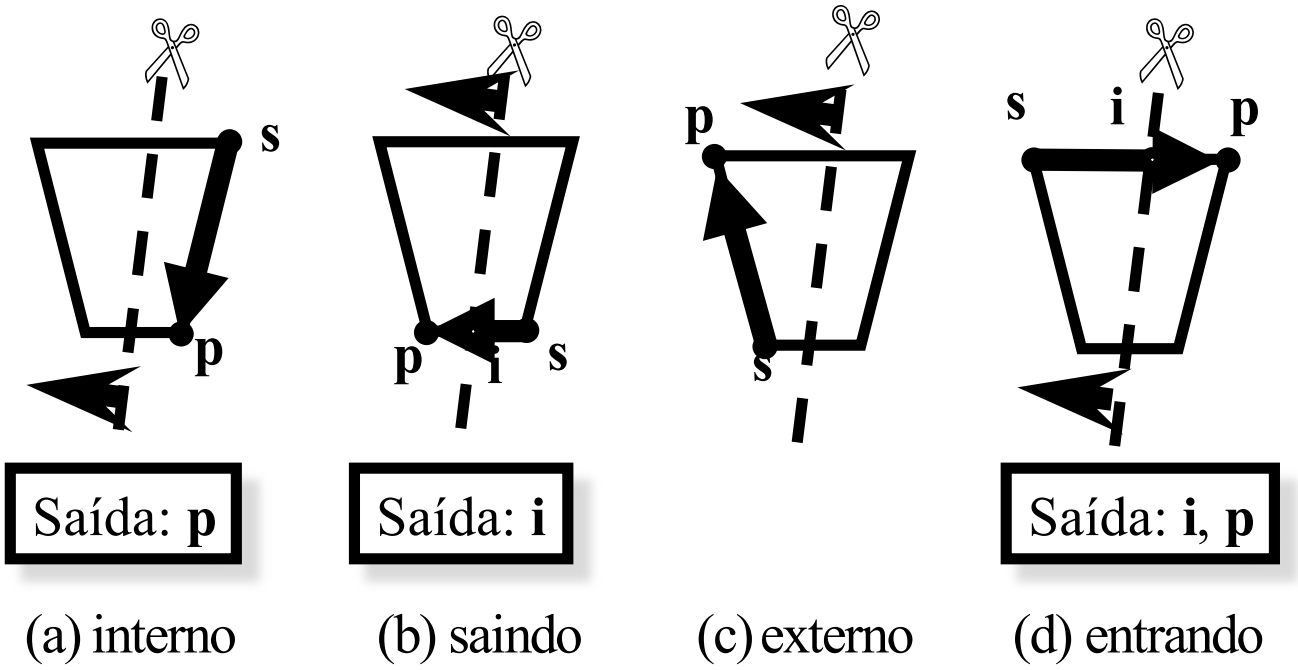
Recorte de polígonos com base no algoritmo de Hodgman-Sutherland



Clip contra uma aresta (plano) de cada vez

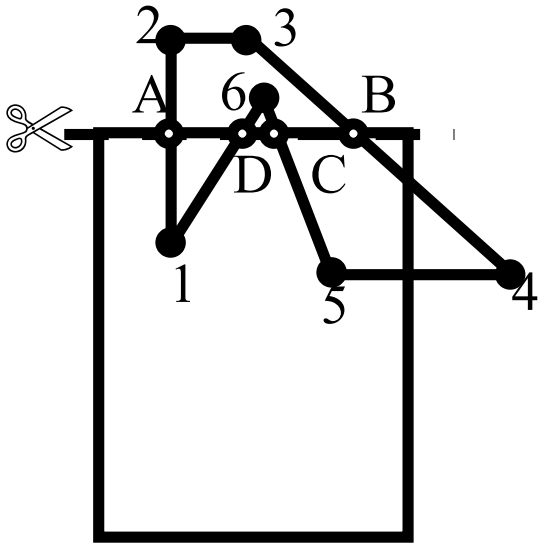
Classificação de um segmento contra um hiperplano no algoritmo de Hodgman-Sutherland

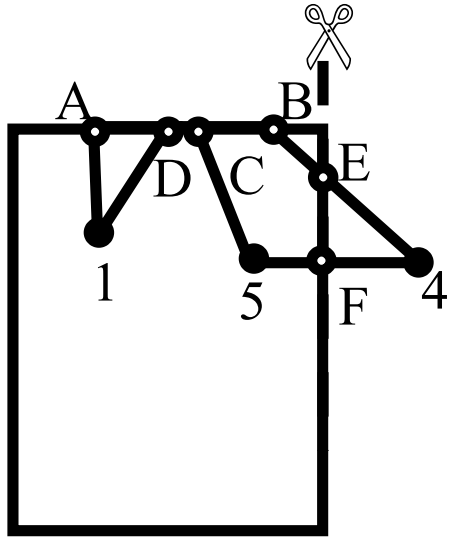
- *Para cada aresta (hiperplano) da região de recorte*
  - *Para cada segmento do polígono*
    - *Existem quatro casos possiveis para um vértice e seu antessessor*

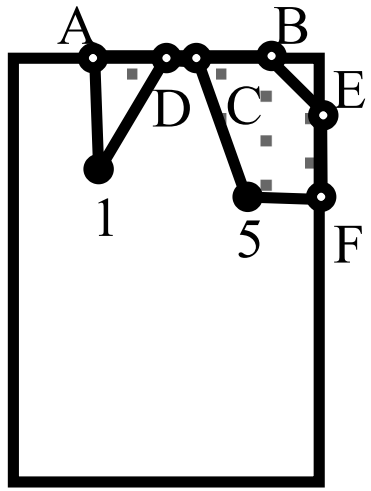


(a) interno (b) saindo (c) externo (d) entrando

Exemplo do algoritmo de Hodgman-Sutherland



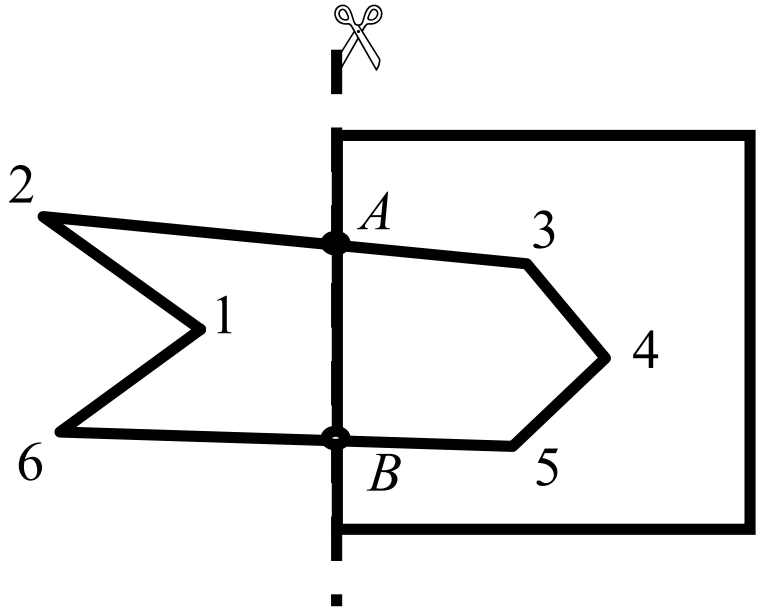




s	**p**	saída
1	2	A
2	3	
3	4	B e 4
4	5	5
5	6	C
6	1	D e 1

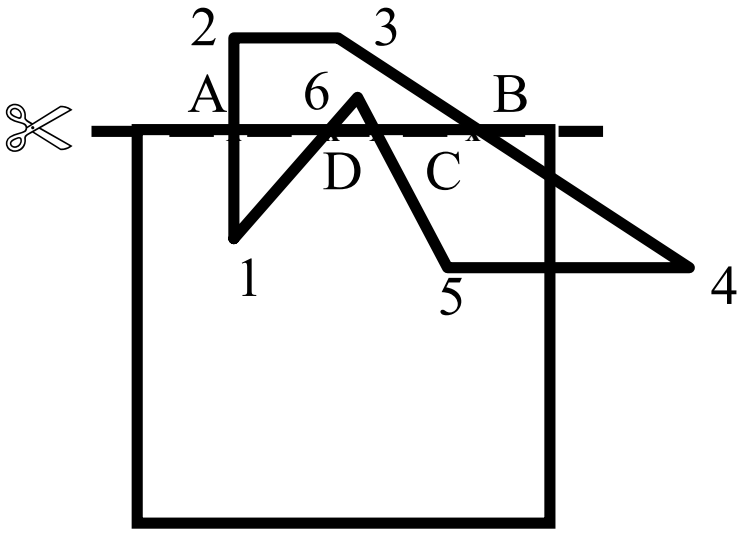
s	**p**	saída
A	B	B
B	4	E
4	5	F e 5
5	C	C
C	D	D
D	1	1
1	A	A

Clipping de polígonos (Exemplo 1)

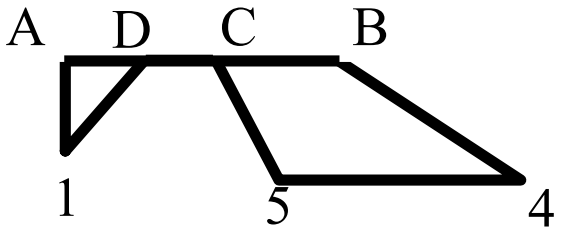


S	P	Ação
1	2	x
2	3	store A,3
3	4	store 4
4	5	store 5
5	6	store B
6	1	x

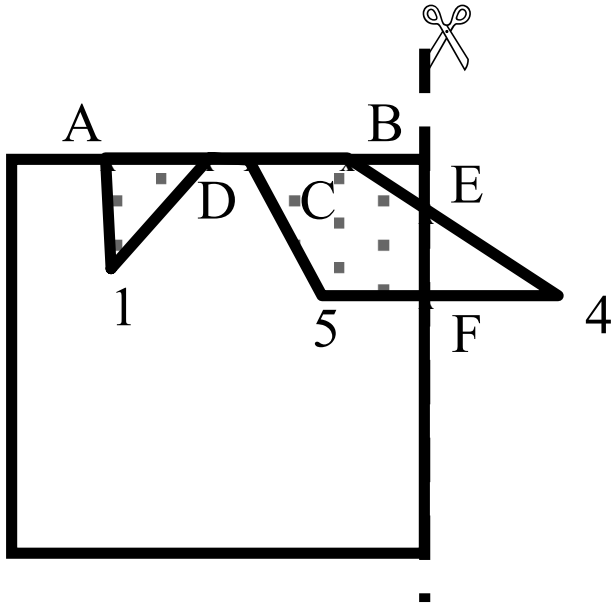
Clipping de polígonos (Exemplo 2)



S	P	Ação
1	2	store A
2	3	x
3	4	store B,4
4	5	store 5
5	6	store C
6	1	store D,1



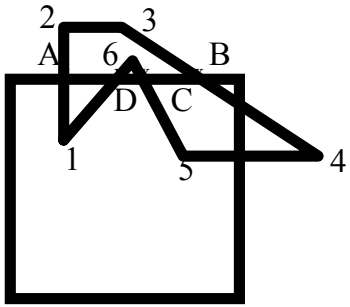
Clipping de polígonos (Exemplo 2)

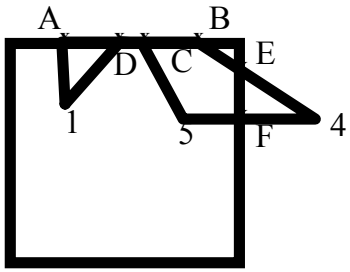


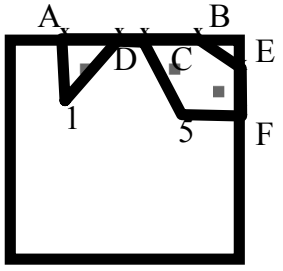
S	P	Ação
A	B	store B
B	4	store E
4	5	store F,5
5	C	store C
C	D	store D
D	1	x
1	A	store 1, A

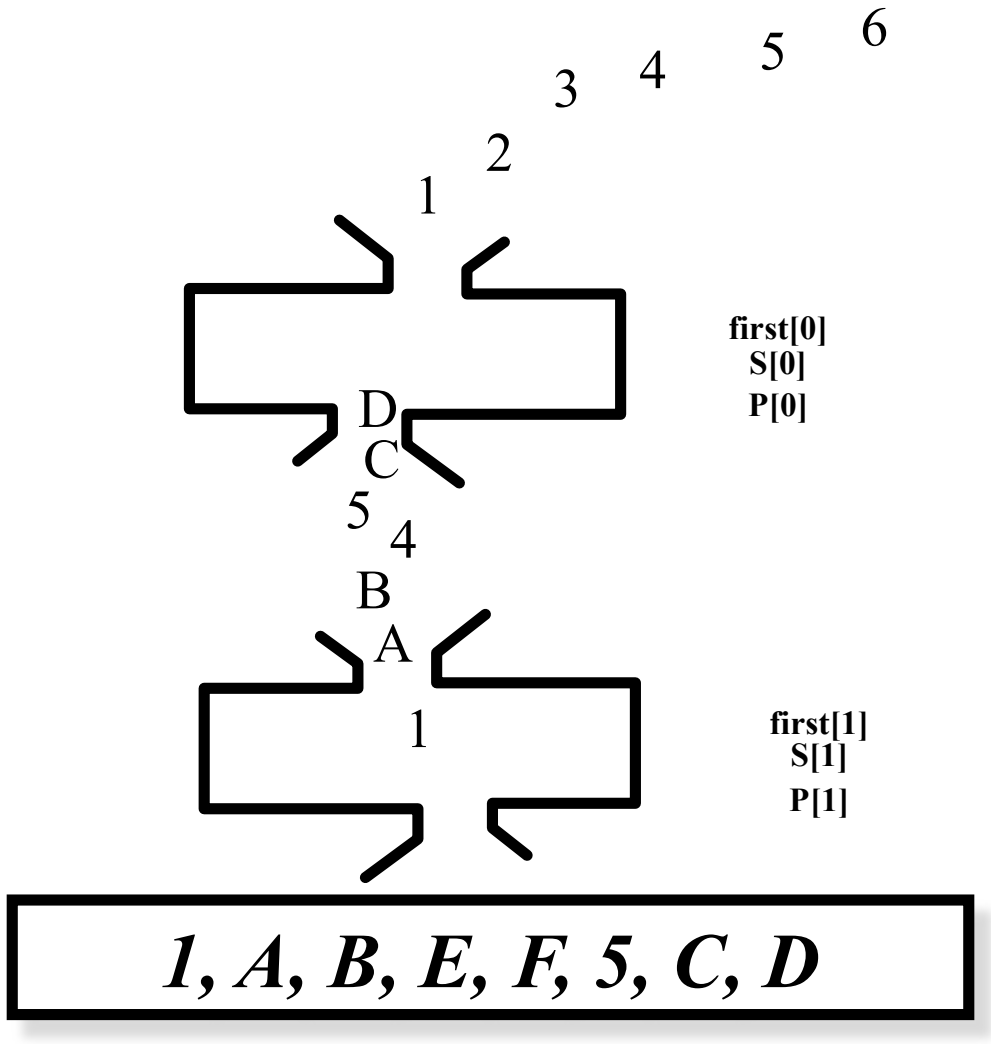
B, E, F, 5, C, D, 1, A

Clipping de polígonos (Concatenação de planos)









APIs para definição de polígonos

```
int npoints; double x[10],y[10];

FillPoly(npoints, x, y);
```

```
typedef struct { double x,y; } Point;

Point vertex[10];

FillPoly(npoints, vertex)
```

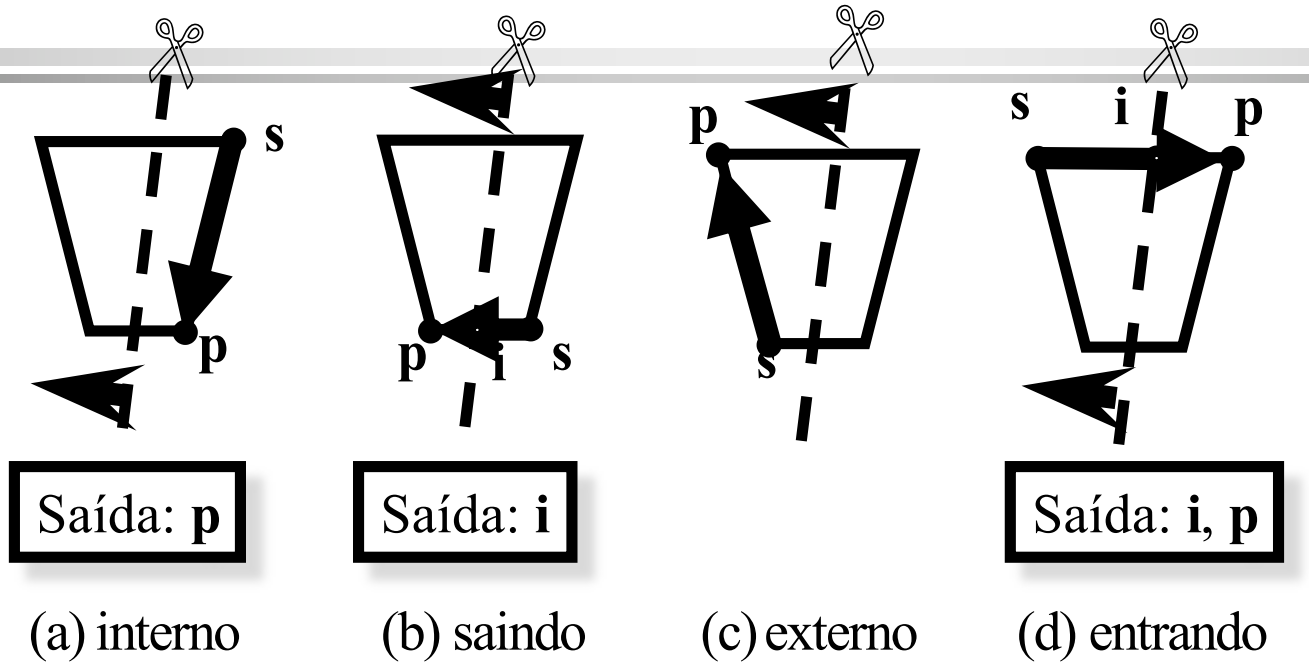
```
Begin(FILL);
  Vertex(30.,20.);
  Vertex(15.,18.);
  …
End( );
```

☑

```
void Vertex(double x, double y, double z)
{
 ClipAtPlane( x, y, z);
}
```

```
void End( );{
 ClipAtPlane(xf,yf,zf);
 FillSavedPoly( );
}
```

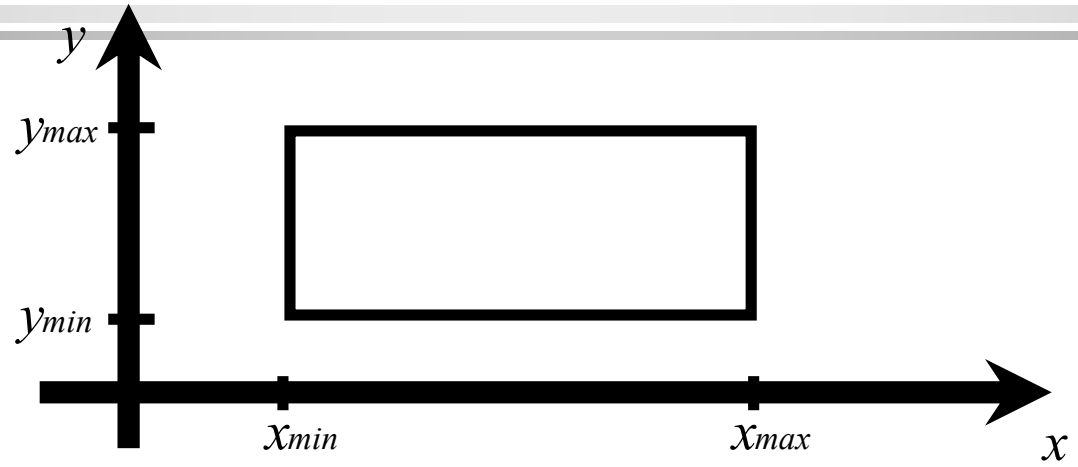
Recorte de polígono contra um plano



(a) interno (b) saindo (c) externo (d) entrando

```
void ClipAtPlane( double x0, double y0, double x1, double y1, int plane )
{
 double d0 = Distance(x0,y0, plane);
 double d1 = Distance(x1,y1, plane);

 if ((d0<0)^(d1<0)) {
      double t=d0/(d0-d1);
      SaveVertex(x0+t*(x1-x0),y0+t*(y1-y0),plane);
   }
 if (d1<=0.0) SaveVertex(x1,y1,plane);
}
```

Recorte de polígono contra um plano




```
double Distance (double x, double y, int plane )
{
 switch( plane )
 {
  case 0: return( x_min - x     );
  case 1: return( x     - x_max );
  case 2: return( y_min - y     );
  case 3: return( y     - y_max );
 }
 return( 0.0 );
}
```

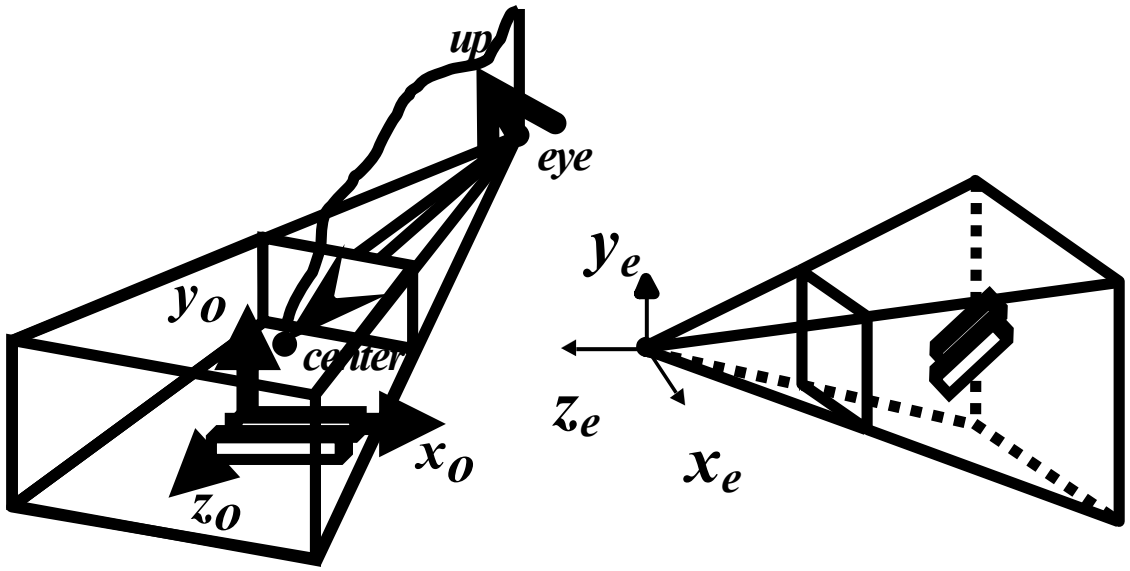
Recorte de polígono contra um plano

```
void ClipVertex(double x, double y, int plane, int last)
{
   static int first=1;
   static double x_first, y_first;
   static double x_previous, y_previous;

   if (first) {
      x_first=x, y_first=y; first =0;
   } else if (!last) {
      ClipAtPlane(x_previous,y_previous,x,y, plane);
   } else {
      ClipAtPlane(x_previous,y_previous,x_first, y_first, plane);
   }
   x_previous=x; y_previous=y;
}
```

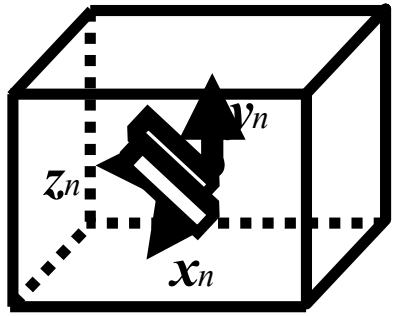
Recorte de polígono contra um plano

```
void Begin(void){
}

void Vertex(double x, double y) {
   int plane = 0;
   int last = 0;
   ClipVertex(x,y,plane,last);
}

void End(void) {
   ClipVertex(0,0,0,1);
}
```

Regiões de recorte do *rendering pipeline*

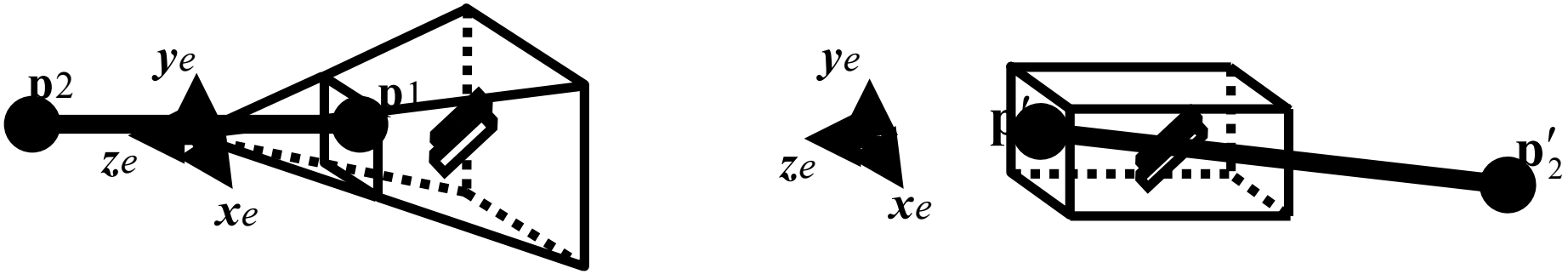


$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w \end{bmatrix}$$



$$\begin{aligned} -1 \le x_n \le 1 \\ -1 \le y_n \le 1 \\ -1 \le z_n \le 1 \end{aligned}$$

Problema do *clipping* no R^3

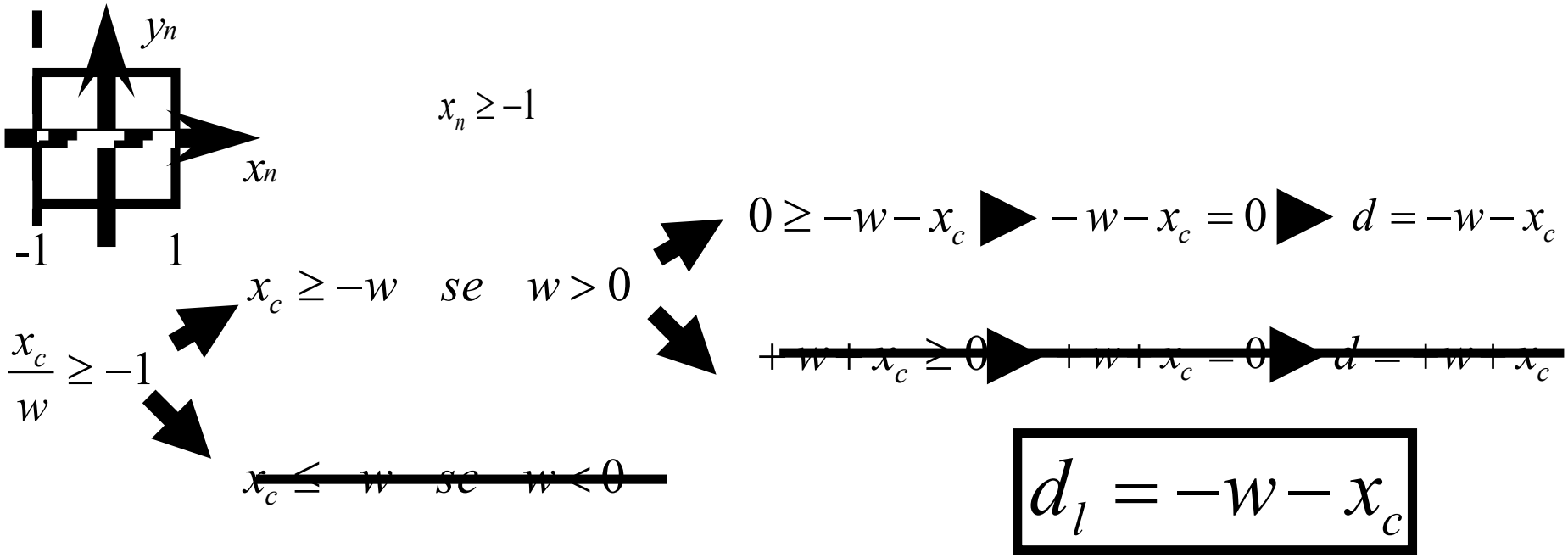


$$\mathbf{p}'_1 = \mathbf{H}\mathbf{p}_1 = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ n \\ -n^2 \\ n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -n \\ 1 \end{bmatrix}$$

$$\mathbf{p}'_2 = \mathbf{H}\mathbf{p}_2 = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ n \\ n^2+2nf \\ -n \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -n-2f \\ 1 \end{bmatrix}$$

$\div w$

Escolhas dos hiperplanos do PR^3



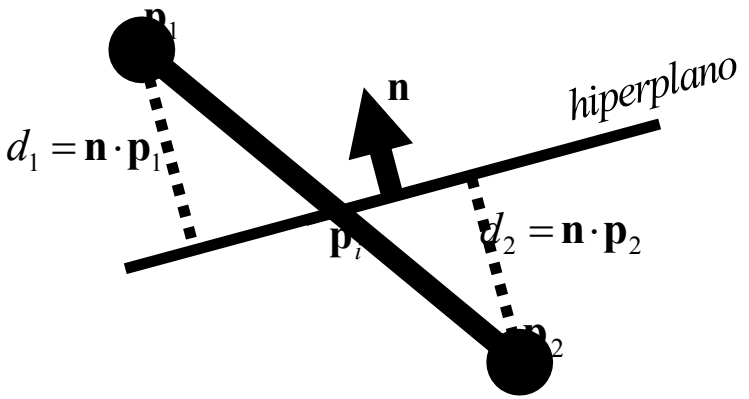
OpenGL Spec

Primitives are clipped to the *clip volume*. In clip coordinates, the *view volume* is defined by

$$\begin{aligned} -w_c \leq x_c \leq w_c \\ -w_c \leq y_c \leq w_c \\ -w_c \leq z_c \leq w_c \end{aligned},$$

Cálculo das distâncias aos hiperplanos no PR^3

```
double Distance (double x, double y, double z, double w, int plane )
{
 switch( plane )
 {
  case 0: return( -w - x );
  case 1: return( -w + x );
  case 2: return( -w - y );
  case 3: return( -w + y );
  case 4: return( -w - z );
  case 5: return( -w + z );
 }
 return( 0.0 );
}
```



$$\textit{intercepta} \quad \Leftrightarrow \quad d_1 d_2 < 0$$

$$\mathbf{p}_i = \frac{|d_1|}{|d_1|+|d_2|}\mathbf{p}_2 + \frac{|d_2|}{|d_1|+|d_2|}\mathbf{p}_1 \qquad \mathbf{p}_i = \frac{d_1}{d_1-d_2}\mathbf{p}_2 - \frac{d_2}{d_1-d_2}\mathbf{p}_1$$

Recorte de polígonos de Sutherland-Hodgman

```
static int first[]={1,1,1,1,1};
static double fx[5],fy[5],fz[5],fw[5],fd[5];
static double sx[5],sy[5],sz[5],sw[5],sd[5];
```

```
void ClipAtPlane( double x, double y, double z, double w, int plane )
{
 double d = Distance(x, y, z, w, plane);

 /* Check whether it is the first point */
 if( first[plane] ) {
      fx[plane]=x;  fy[plane]=y;  fz[plane]=z; fw[plane]=w;
      fd[plane]=d;  first[plane]=0;
 } else if ((sd[plane] < 0)^(d < 0))
      Intersect( x, y, z, w, plane, sd[plane], d );

 /* Save as previous */
 sx[plane]=x; sy[plane]=y; sz[plane]=z; sw[plane]=w; sd[plane]=d;

 /* Check whether it is a visible point */
 if ( d <= 0.0 ) {
    if ( plane == LAST_PLANE )
       SaveVertex( x, y, z, w );
    else
       ClipAtPlane( x, y, z, w, plane+1 );
 }
}
```

Teste de interseção com uma fronteira

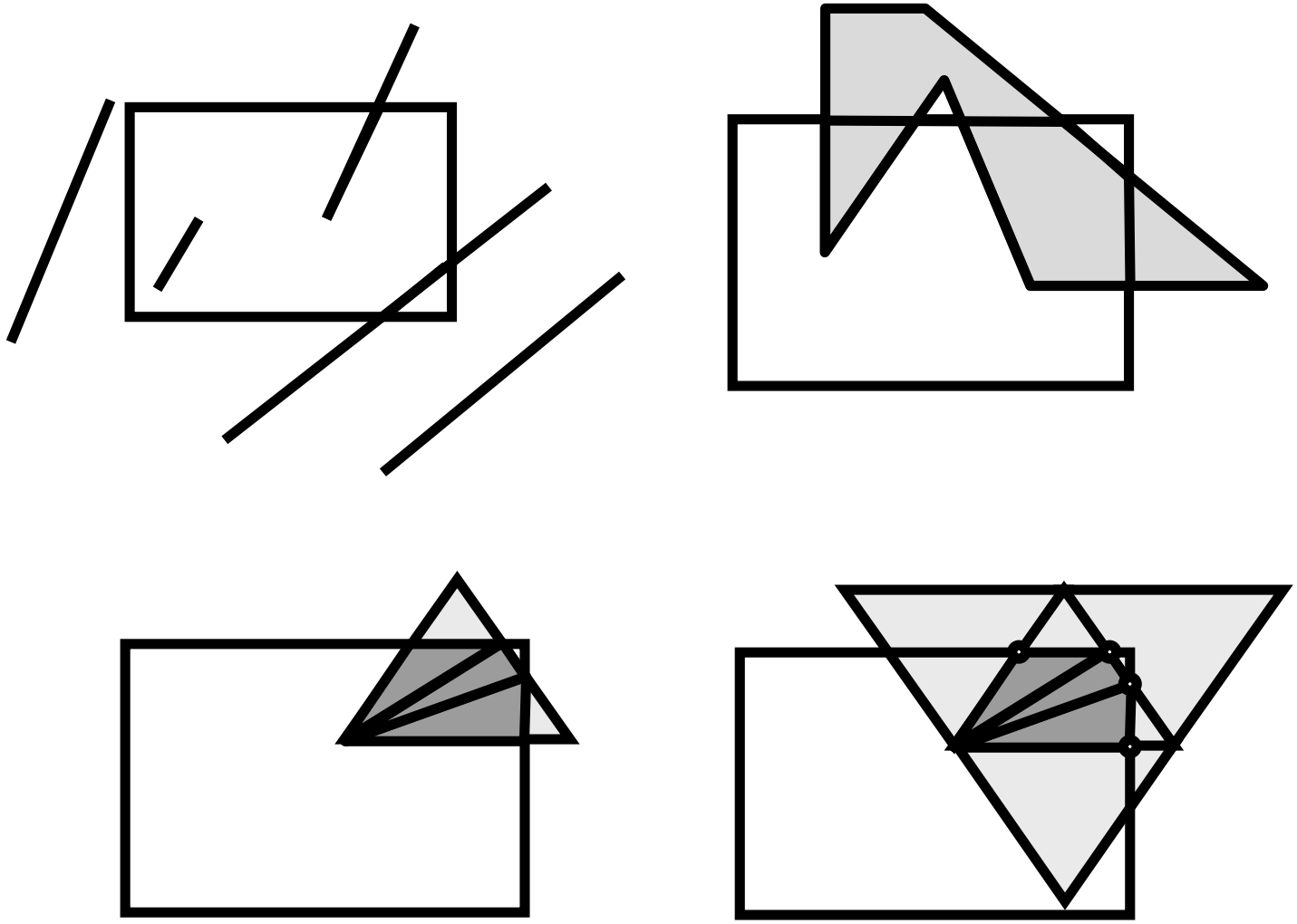
```
void Intersect(double x, double y, double z, double w,
               int plane, double d1, double d2)
{
 double t  = d1 / (d1 - d2);

 double xi = sx[plane] + t * (x - sx[plane]);
 double yi = sy[plane] + t * (y - sy[plane]);
 double zi = sz[plane] + t * (z - sz[plane]);
 double wi = sw[plane] + t * (w - sw[plane]);

 if( plane == LAST_PLANE )
   SaveVertex( xi, yi, zi, wi );
 else
   ClipAtPlane( xi, yi,zi, wi, plane+1 );
}
```

Fechando o processo

```
Begin(FILL);
  Vertex(30.,20.);
  Vertex(15.,18.);
    …
End( );
```

```
void ClipClose( int plane )
{
 if ((sd[plane]<0)^(fd[plane]<0))
     Intersect( fx[plane], fy[plane], plane, sd[plane], fd[plane] );

 first[plane]=1;

 if( plane == LAST_PLANE )
   FillSavedPolygon( );
 else
   ClipClose( plane+1 );
}
```

Diferentes problemas de recorte



FIM