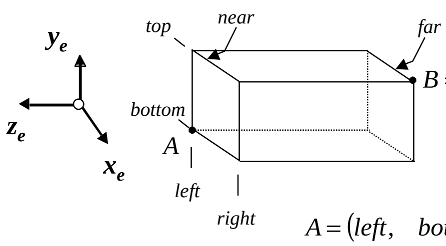
Modelo de Câmera e Projeção no OpenGL

por
Marcelo Gattass
Departamento de Informática
PUC-Rio

(adaptado por Luiz Fernando Martha para a disciplina CIV2801 - Fundamentos de Computação Gráfica Aplicada)

Projeção Paralela (Ortho)



$B = (right, \quad top, \quad -far)$

Obs.: *near* e *far* são distâncias (> 0)

$A = (left, \quad bottom, \quad -near)$

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

Define volume de visão para projeção ortográfica no sistema de coordenadas da câmera (olho).

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

Matriz Ortho do OpenGL

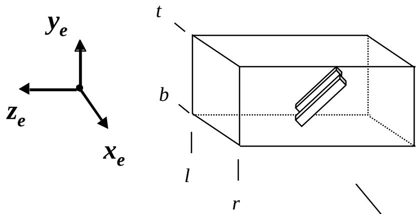
$$[\,T\,] = \begin{bmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & +(f+n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

OpenGL Spec

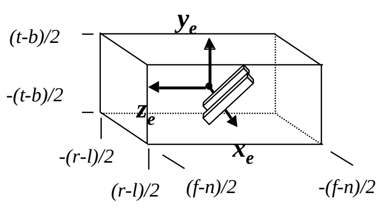
$$[\,S\,][\,T\,] = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matriz Ortho do OpenGL: [*T*] translada o paralelepípedo de visão para origem

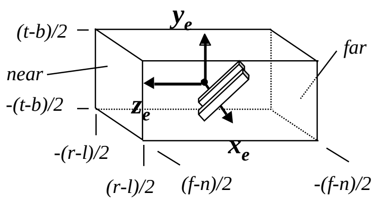


$$[\,T\,] = \begin{bmatrix} 1 & 0 & 0 & -(r+l)/2 \\ 0 & 1 & 0 & -(t+b)/2 \\ 0 & 0 & 1 & +(f+n)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Obs.: *near* e *far* são distâncias (> 0) e o paralelepípedo está no lado negativo do eixo z.

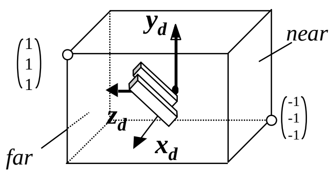


Matriz Ortho do OpenGL: [*S*] escala o paralelepípedo de visão no cubo [-1,1]x[-1,1]x[-1,1]



Inverte a direção de z, de tal forma que o plano near tem o menor valor de z (menor profundidade).

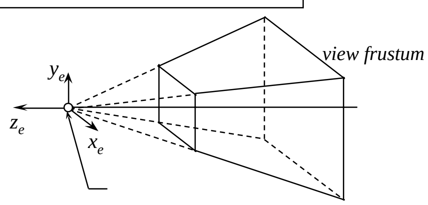
$$[S] = \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & -2/(f-n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



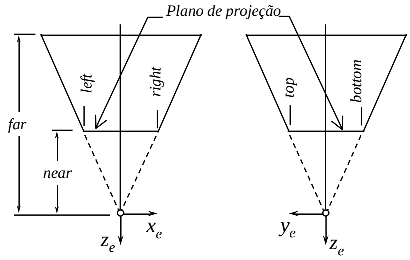
Projeção Cônica (Frustum)

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

Define volume de visão para projeção cônica no sistema de coordenadas da câmera.



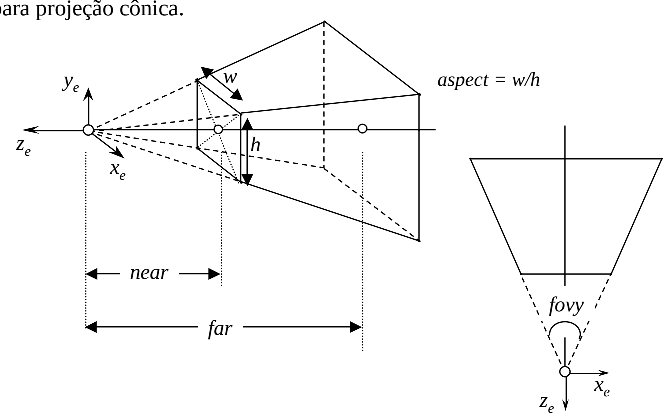
Obs.: *near* e *far* são distâncias (> 0)



Projeção Cônica (Perspective)

```
void glPerspective( GLdouble fovy, GLdouble aspect,
                    GLdouble near_, GLdouble far_ );
```

Alternativa para definir volume de visão para projeção cônica.



Matriz Frustum do OpenGL

$$[\,P\,] = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & n \bullet f \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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

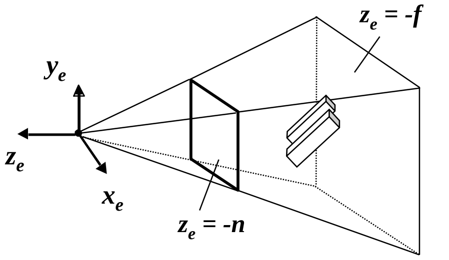
$$[\,S\,][\,T\,][\,P\,] = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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

OpenGL Spec

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

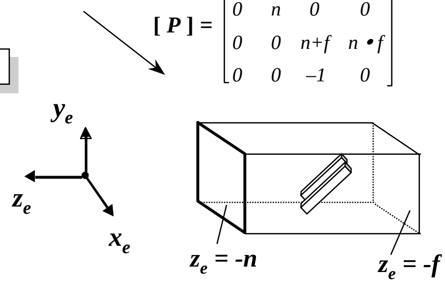
Matriz Frustum do OpenGL: [*P*] distorce o frustum de visão para um paralelepípedo



Mantém a altura do frustum de visão na distorção. Isto faz o problema da projeção cônica recair no problema padrão de projeção ortográfica.

Plano de projeção é o plano *near*: $d = n$

$$[\,P\,] = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & n \bullet f \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

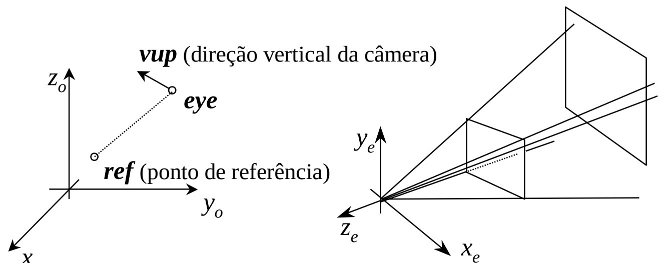


Glu LookAt

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,
               GLdouble refx, GLdouble refy, GLdouble refz,
               GLdouble vupx, GLdouble vupy, GLdouble vupz);
```

Dados: ***eye***, ***ref***, ***vup*** *(definem o sistema de coordenadas do olho)*

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



Sistema de coordenadas dos Objetos

Sistema de coordenadas do Olho

Matriz LookAt do OpenGL

$$[\,\boldsymbol{T_c}\,] = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\boldsymbol{z_e} = -\,\boldsymbol{view} / ||\boldsymbol{view}||$

$\boldsymbol{x_e} = (\boldsymbol{vup} \times \boldsymbol{z_e}) / ||\boldsymbol{vup} \times \boldsymbol{z_e}||$

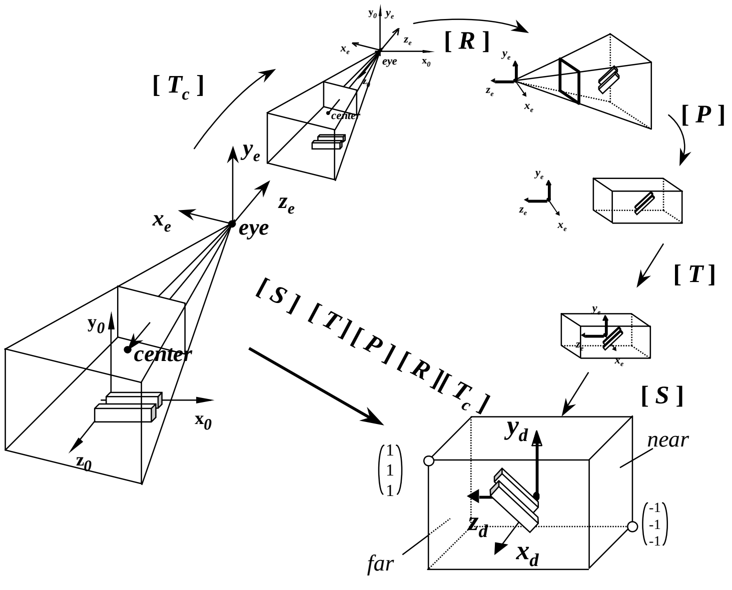
$\boldsymbol{y_e} = \boldsymbol{z_e} \times \boldsymbol{x_e}$

$$[\,\boldsymbol{R}\,] = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ez} & 0 \\ z_{ex} & z_{ey} & z_{ez} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

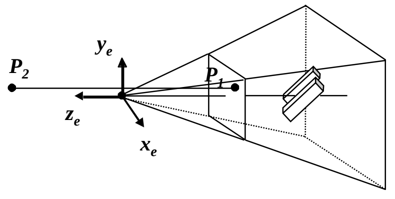
Matriz LookAt do OpenGL:

$[\,\boldsymbol{C}\,] = [\,\boldsymbol{R}\,]\,[\,\boldsymbol{T_c}\,]$

Concatenação das transformações



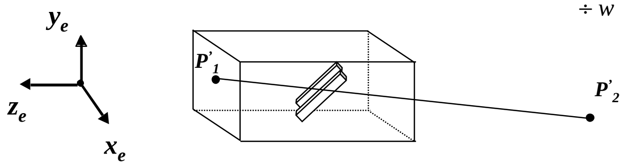
Problema do *clipping* (cerceamento contra volume de visão)



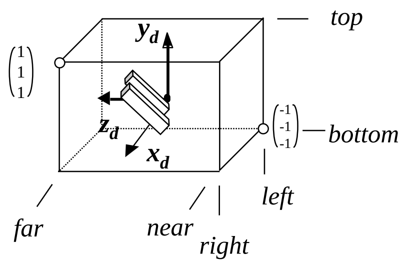
$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & n \bullet f \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ n \\ -n^2 \\ n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -n \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & n \bullet f \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} n \\ n \\ n^2+2nf \\ -n \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -n-2f \\ 1 \end{bmatrix}$$

$\div\, w$

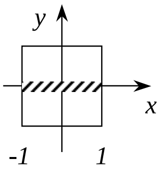


Clipping em coordenadas homogêneas



$x \in$ [*left*, *right*]	$-1 \le x \le 1$	$-1 \le x_h/w \le 1$
$y \in$ [*bottom*, *top*]	$-1 \le y \le 1$	$-1 \le y_h/w \le 1$
$z \in$ [*near*, *far*]	$-1 \le z \le 1$	$-1 \le z_h/w \le 1$

Clipping em coordenadas homogêneas



***x* ∈ [*left, right*]**

$-1 \leq x_h/w \leq 1$

$x_h/w \leq 1$ → $x_h \leq w$, se $w>0$

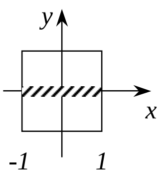
$x_h/w \leq 1$ → $x_h \geq w$, se $w<0$

OpenGL Spec

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

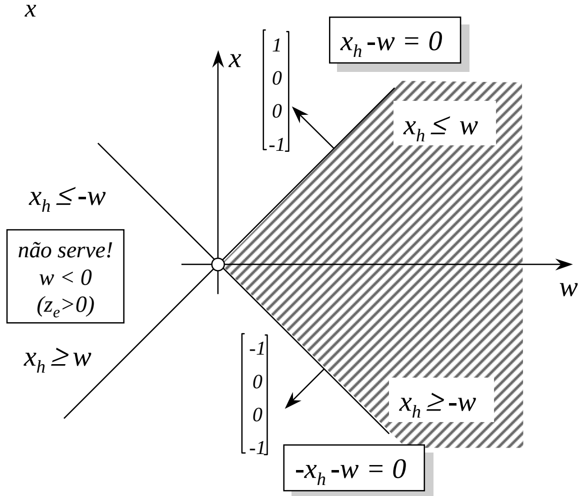
$$-w_c \leq x_c \leq w_c$$
$$-w_c \leq y_c \leq w_c \;.$$
$$-w_c \leq z_c \leq w_c$$

Clipping em coordenadas homogêneas



***x* ∈ [*left, right*]**

$-1 \leq x_h/w \leq 1$

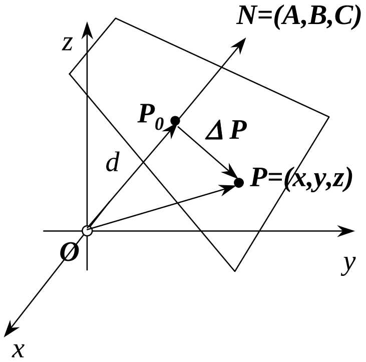


Equação de um plano

$N.P = Ax + By + Cz$

$N.P = N.(P_0 + \Delta P) = N.P_0 = d$

$d = Ax + By + Cz$



$Ax + By + Cz + D = 0$

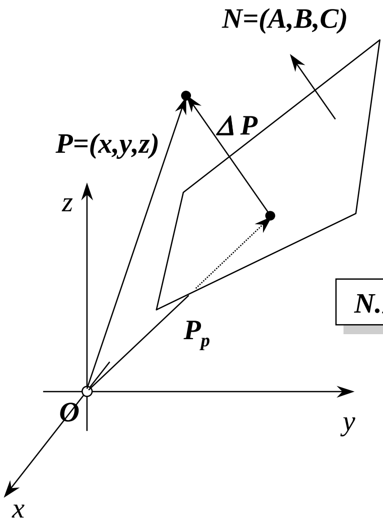
$(A, B, C) = N$
e
$D = -d = N.(-P_0)$

Distância de um ponto a um plano

$N.P = Ax + By + Cz$

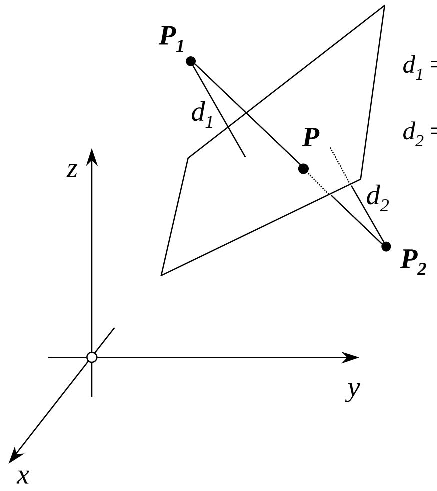
$N.P = N.(P_p + \Delta P)$

$N.P = d + N.\Delta P$



$N.\Delta P = Ax + By + Cz + D$

Interseção de reta com plano



$$d_1 = | Ax_1 + By_1 + Cz_1 + D |$$

$$d_2 = | Ax_2 + By_2 + Cz_2 + D |$$

$$\boldsymbol{P} = \frac{d_1 \boldsymbol{P_2} + d_2 \boldsymbol{P_1}}{d_1 + d_2}$$

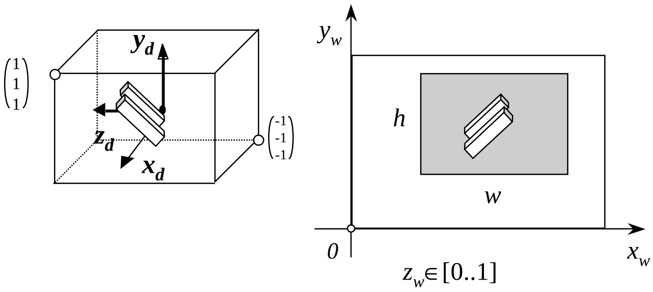
Cálculo das distâncias

```
/* ==================== Distance =====================
**
** This function computes and returns the distance between a
** point and a plane. Normal points toward out.
*/
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 );
}
```

Transformação para o Viewport

```
void glViewport(GLint x0, GLint y0,
                GLsizei width, GLsizei height );
```

$$x_w = x_0 + w*(x_d - (-1)) / 2$$
$$y_w = y_0 + h*(y_d - (-1)) / 2$$
$$z_w = z_d / 2 + 1/2$$



Transformações de um vértice

OpenGL Spec

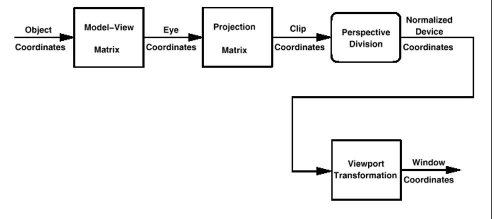
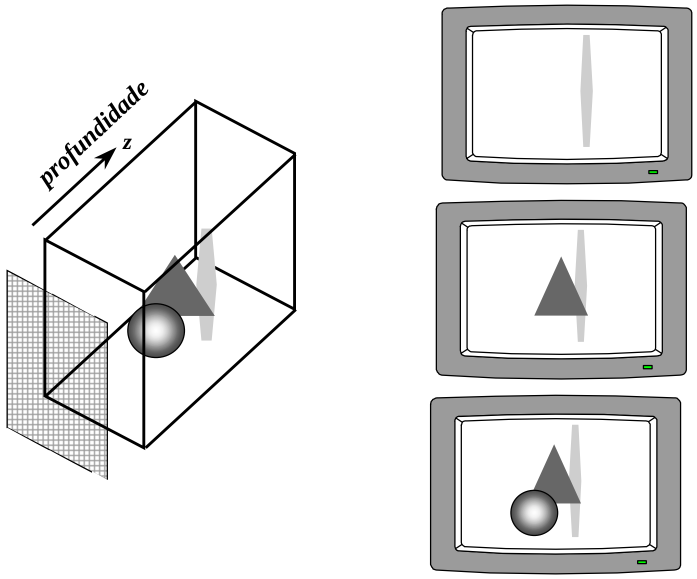
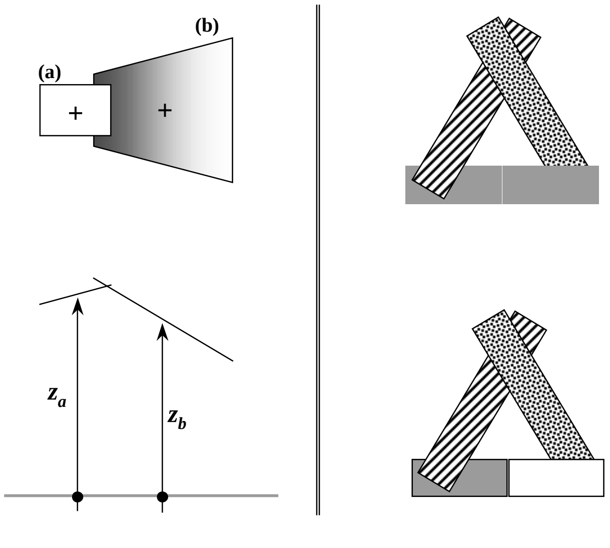


Figure 2.6. Vertex transformation sequence.

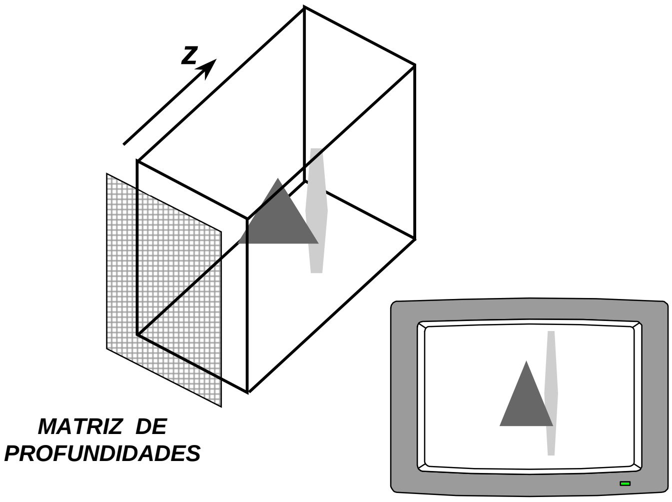
Modelo do Pintor



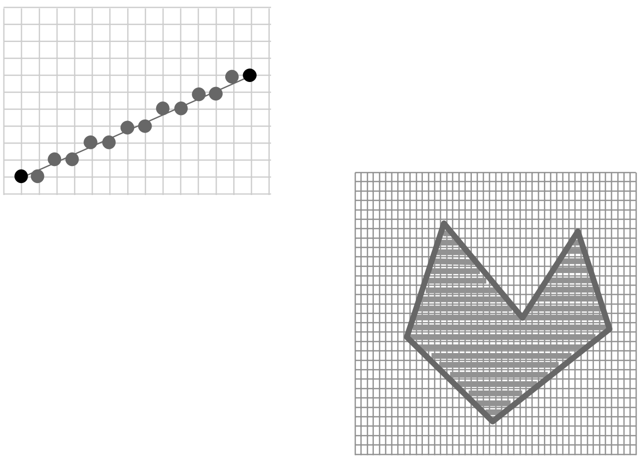
Problemas na ordenação de faces



ZBuffer: idéia básica



Rasterização de Polígonos e Linhas

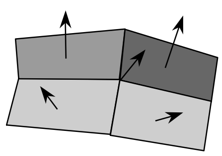


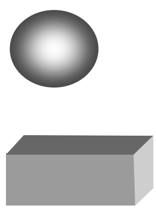
ZBuffer - pseudo-código

```
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,0);
   }
 }

  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 */
```

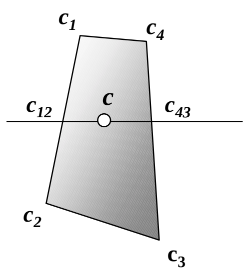
```
void glEnable( GL_DEPTH_TEST );
```

Suavização da tonalização

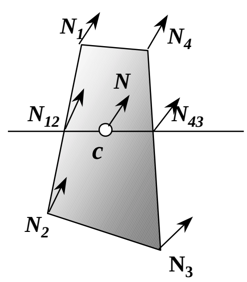




Gouraud

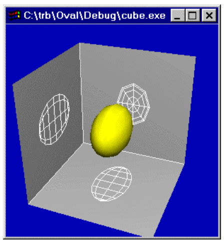


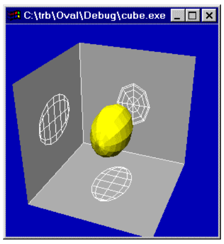
Phong



Interpolação de cores

```
void glShadeModel (GL_SMOOTH);
```






```
void glShadeModel (GL_FLAT);
```