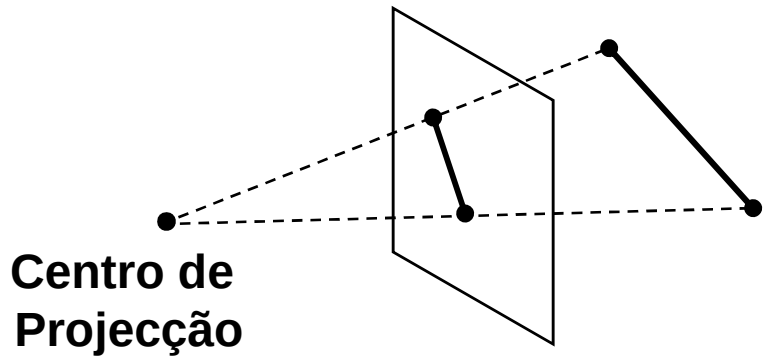
Introdução à Computação Gráfica
Projecções

Adaptação: João Paulo Pereira
António Costa

Autoria: Claudio Esperança
Paulo Roma Cavalcanti

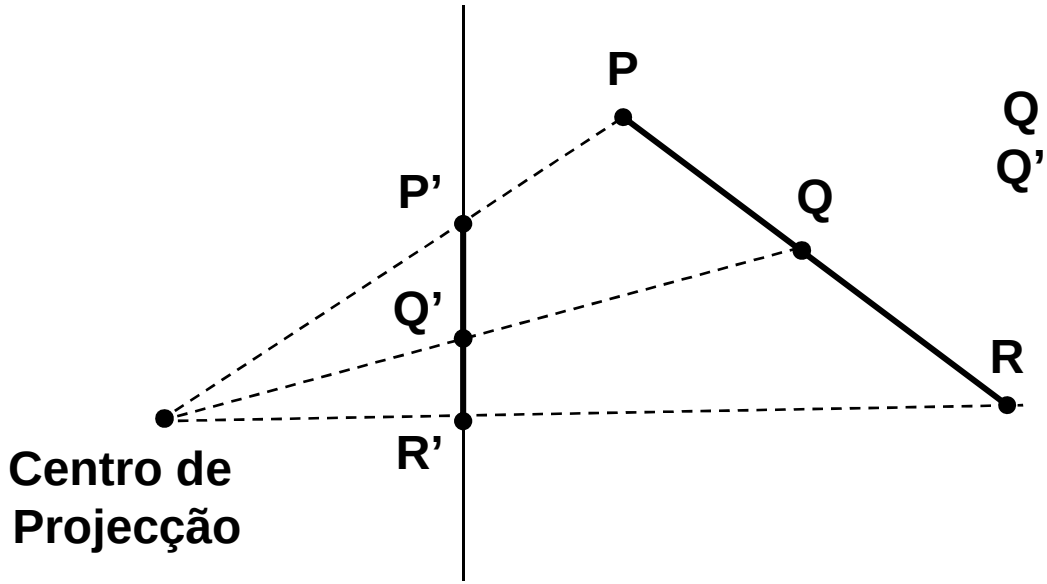
Perspectiva

- É o estudo de transformações projectivas
- O problema consiste em projectar pontos do espaço *d*-dimensional no plano *d*-1 dimensional usando um ponto especial chamado de centro de projecção



Transformações Projectivas

- Transformações projectivas transformam rectas em rectas mas não preservam as combinações afim

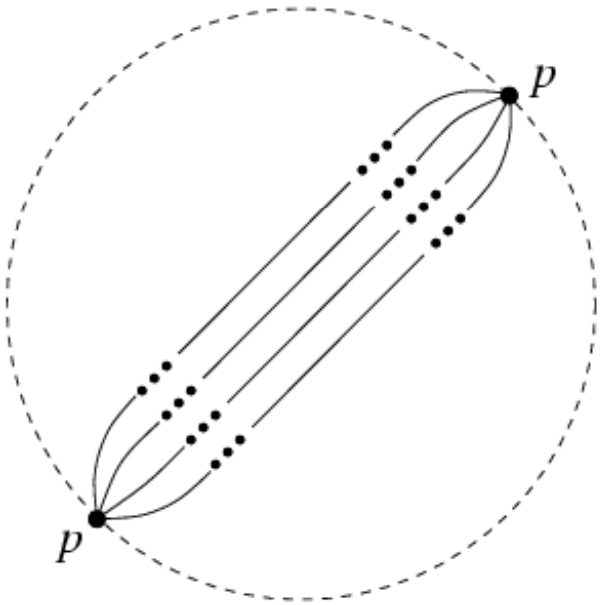


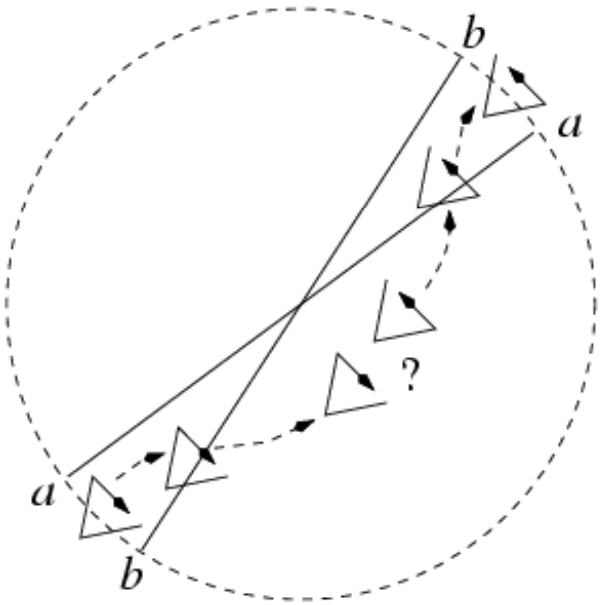
Q é o ponto médio de PR, mas Q' não é o ponto médio de P'R'

Geometria Projectiva

- Geometria euclidiana: duas rectas paralelas não se encontram
- Geometria projectiva: assume-se a existência de pontos *ideais* (no infinito)
  - Rectas paralelas encontram-se num ponto ideal
  - Para não haver mais de um ponto ideal para cada inclinação de recta, assume-se que o plano projectivo se fecha sobre si mesmo
  - Em 2D o plano projectivo tem uma fronteira que é uma recta no infinito (feita de pontos ideais)
  - Transformações projectivas podem transformar pontos ideais em pontos do plano euclidiano e vice-versa
  - Problemas: O plano projectivo é uma variedade não orientável
- Vamos usar geometria projectiva apenas para projectar pontos

Geometria Projectiva

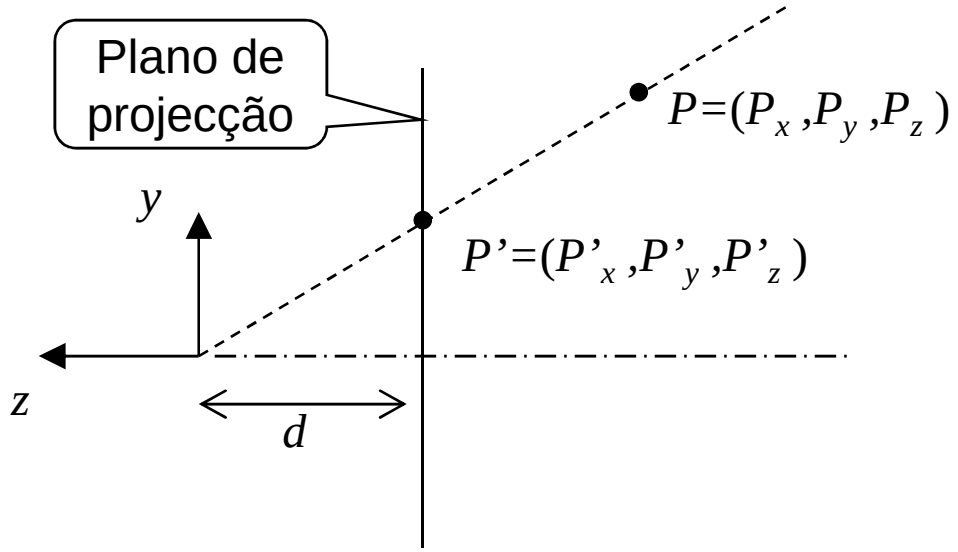




Coordenadas homogéneas em espaço projectivo

- Representamos apenas pontos (não vectores)
- Em 2D, um ponto (x,y) será representado em c.h. pela matriz-coluna $[x \cdot w \quad y \cdot w \quad w]^T$, para $w \neq 0$
  - Assim, o ponto (4,3) pode ser representado por $[8\ 6\ 2]^T$, $[12\ 9\ 3]^T$, $[-4\ -3\ -1]^T$, etc.
- Dado um ponto com coordenadas homogéneas $[x \ \ y \ \ w]^T$, a sua representação canônica é dada por $[x/w \ \ y/w \ \ 1]^T$
  Designamos essa operação por *divisão perspectiva*
- Considere os pontos da recta $x=y$: (1,1), (2,2), etc
  - Podemos representá-los em c.h. por $[1\ 1\ 1]^T$, $[1 \ \ 1 \ \ ½]^T$, etc.
  - Claramente, o ponto ideal dessa recta é dado por $[1\ 1\ 0]^T$

Transformações projectivas



- Se o plano de projecção é perpendicular ao eixo *z*, está a uma distância *d* do C.P. (que está na origem) e intercepta o semi-eixo z negativo, então a projecção de um ponto P é dada por

$$P' = \begin{bmatrix} \frac{P_x}{-P_z/d} & \frac{P_y}{-P_z/d} & -d & 1 \end{bmatrix}^T$$

- Por semelhança de triângulos, vemos que $P_y / -P_z = P'_y / d$

Transformação perspectiva em coordenadas homogéneas

- Não existe matriz 4x4 capaz de realizar tal transformação em espaços euclidianos, mas se assumirmos que o ponto está no espaço projectivo, então

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ -P_z/d \end{bmatrix} = \begin{bmatrix} \dfrac{P_x}{-P_z/d} \\ \dfrac{P_y}{-P_z/d} \\ -d \\ 1 \end{bmatrix}$$

Perspectiva - Sumário

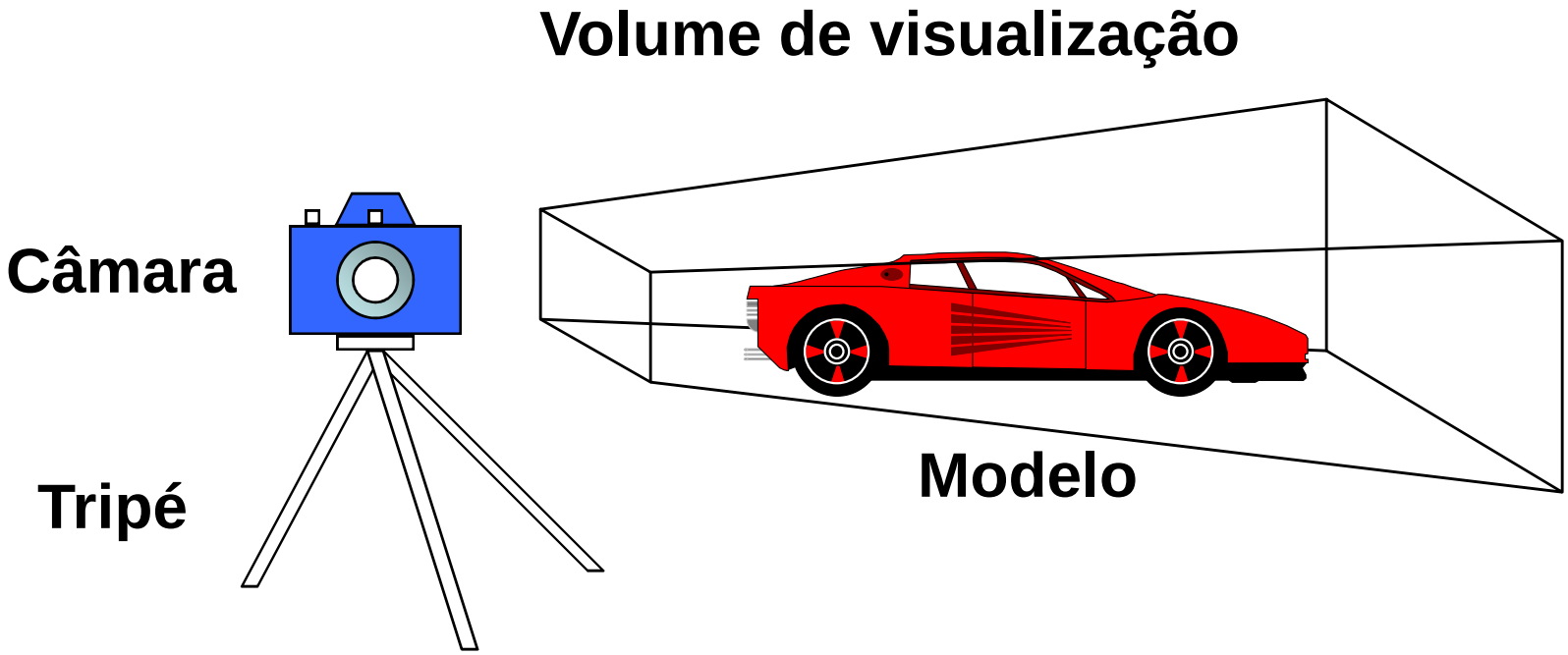
- Para fazer projecção perspectiva de um ponto *P*, seguem-se os seguintes passos
  - *P* é levado do espaço euclidiano para o projectivo
    - Trivial – mesmas coordenadas homogéneas
  - *P* é multiplicado pela matriz de transformação perspectiva resultando em *P'*
  - *P'* é levado novamente ao espaço euclidiano
    - Operação de divisão perspectiva

Projecção genérica

- E se não queremos que o Centro de Projecção esteja na origem ou se a cena não está correctamente posicionada no semi-eixo z negativo?
  - Aplicam-se transformações afim para posicionar todos os elementos correctamente
  - As maneiras pelas quais essas transformações são feitas caracterizam um dado modelo de projecção

Modelo de câmara sintética

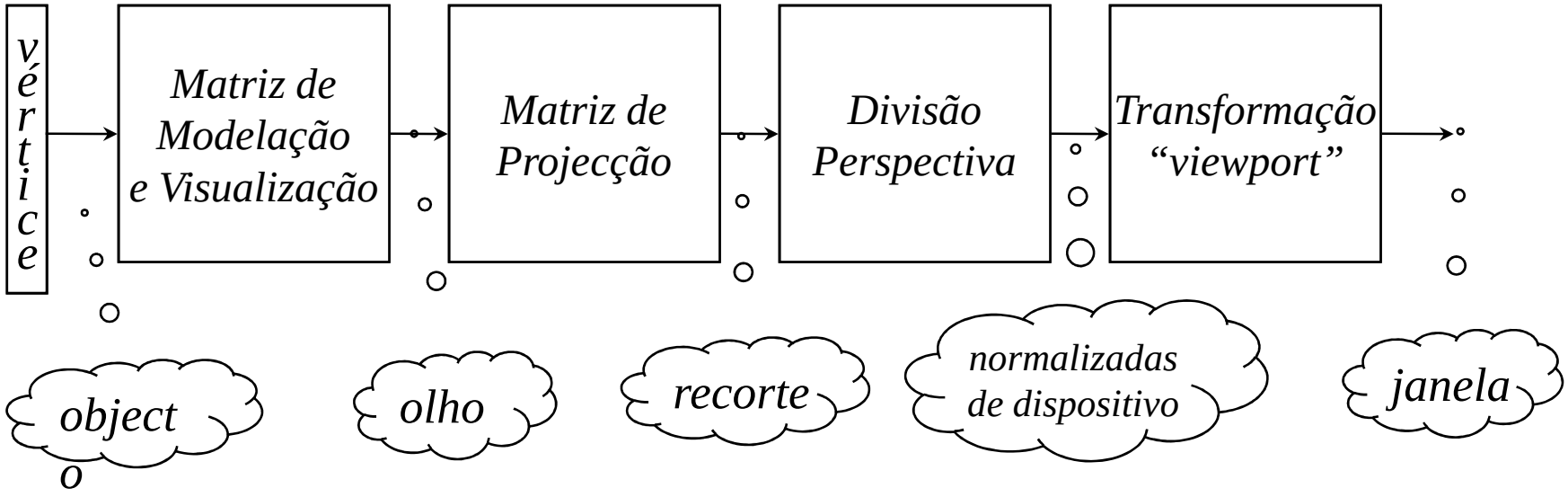
- OpenGL utiliza uma analogia comparando visualização 3D com tirar fotografias com uma câmara



Transformações em OpenGL

- Modelação
  - Mover / deformar os objectos
- Visualização
  - Mover e orientar a câmara
- Projecção
  - Ajuste da lente / objectiva da câmara
- "*Viewport*"
  - Aumentar ou reduzir a fotografia

Pipeline OpenGL de Transformações



Estado inicial do *pipeline*

- Inicialmente,
  - As matrizes "*modelview*" e "*projection*" são matrizes-identidade
    - Vértices não são transformados e a projecção é paralela ao plano *xy*
    - O mundo visível é restrito ao cubo $-1 \leq x,y,z \leq 1$
  - A transformação "*viewport*" transforma o quadrado $-1 \leq x,y \leq 1$ (em coordenadas normalizadas de dispositivo) na superfície total da janela

Especificação do *viewport*

- Para especificar a área da janela na qual será transformado o quadrado do plano de projecção, utiliza-se

  `glViewport (`*`x0, y0, largura, altura`*`)`

  - (parâmetros em *pixels,* sendo que (0, 0) se refere ao canto inferior esquerdo da janela)
- Normalmente não é necessário modificar, mas é útil para
  - Manter a razão de aspecto da imagem
  - Fazer *zooming* e *panning* sobre a imagem

Especificação de transformações

- As matrizes *modelview* e *projection* usadas no pipeline são aquelas que se situam no topo de duas pilhas que são usadas para fazer operações com matrizes
- Para selecionar em qual pilha queremos operar, usamos

  glMatrixMode(*GL_MODELVIEW* ou *GL_PROJECTION*)
- Existe uma série de funções para operar com a pilha corrente, incluindo

  glLoadIdentity ()
  glLoadMatrix* ()
  glMultMatrix* ()
  glPushMatrix ()
  glPopMatrix ()

Transformação de objectos

- Usam-se funções para multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros
  - **glTranslate* (*x, y, z*)**
  - **glRotate* (*ângulo, x, y, z*)**
  - **glScale* (x, y, z)**

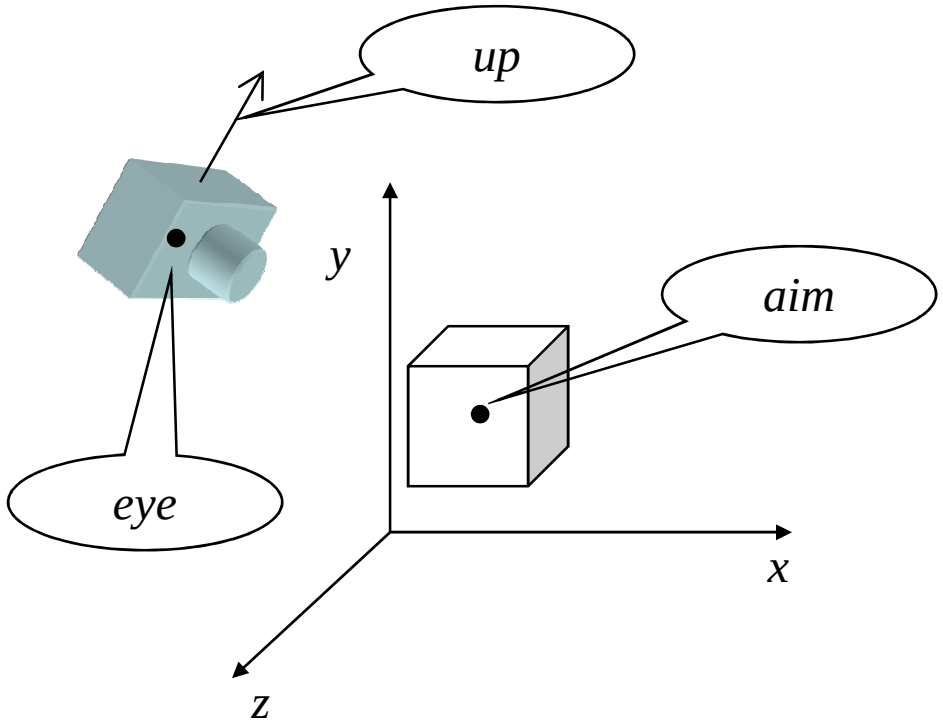
- Cuidado: a ordem é importante:

```
glTranslatef (10, 5, 3);
glRotatef (10, 0, 0, 1);
glBegin (GL_TRIANGLES);
…
```

  - **O objecto é rodado e depois transladado!**

Transformações de visualização

- Duas interpretações:
  - Levam a câmara até a cena que se quer visualizar
  - Levam os objectos da cena até uma câmara estacionária
- `gluLookAt(`
  `eye`$_x$`, eye`$_y$`, eye`$_z$`,`
  `aim`$_x$`, aim`$_y$`, aim`$_z$`,`
  `up`$_x$`, up`$_y$`, up`$_z$`);`
  - eye = ponto onde a câmara será posicionada
  - aim = ponto para onde a câmara será apontada
  - up = vector que dá a direção "para cima" da câmara
  - *Cuidado com casos degenerados*

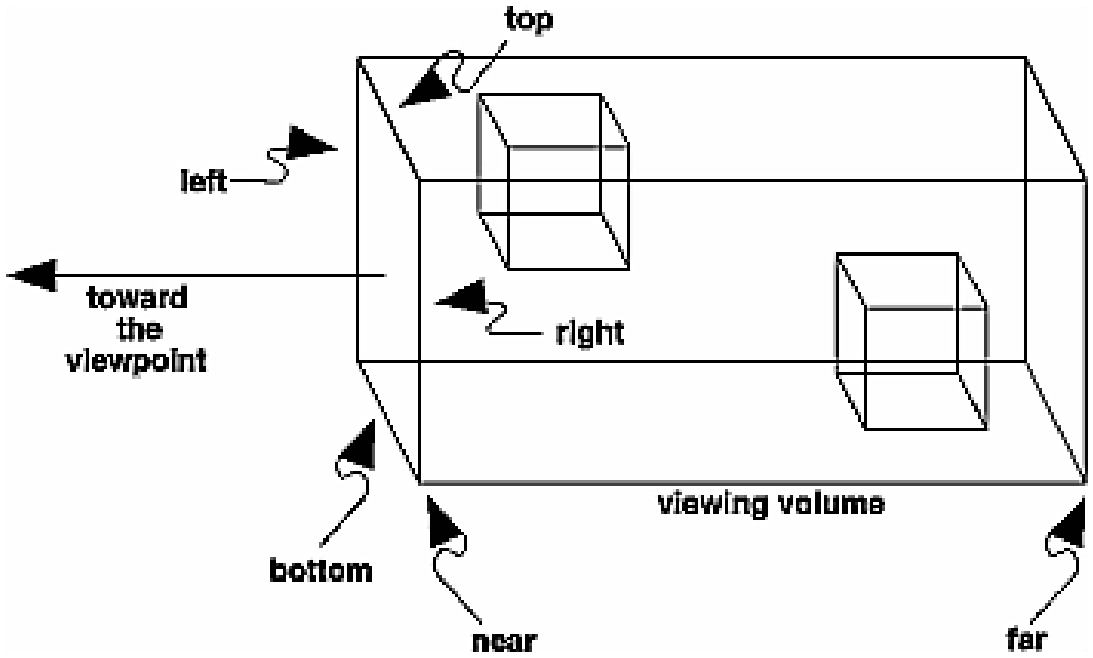


Projecção Paralela

- *Default* em OpenGL
- Para ajustar o volume visível, a matriz de projecção é inicializada com

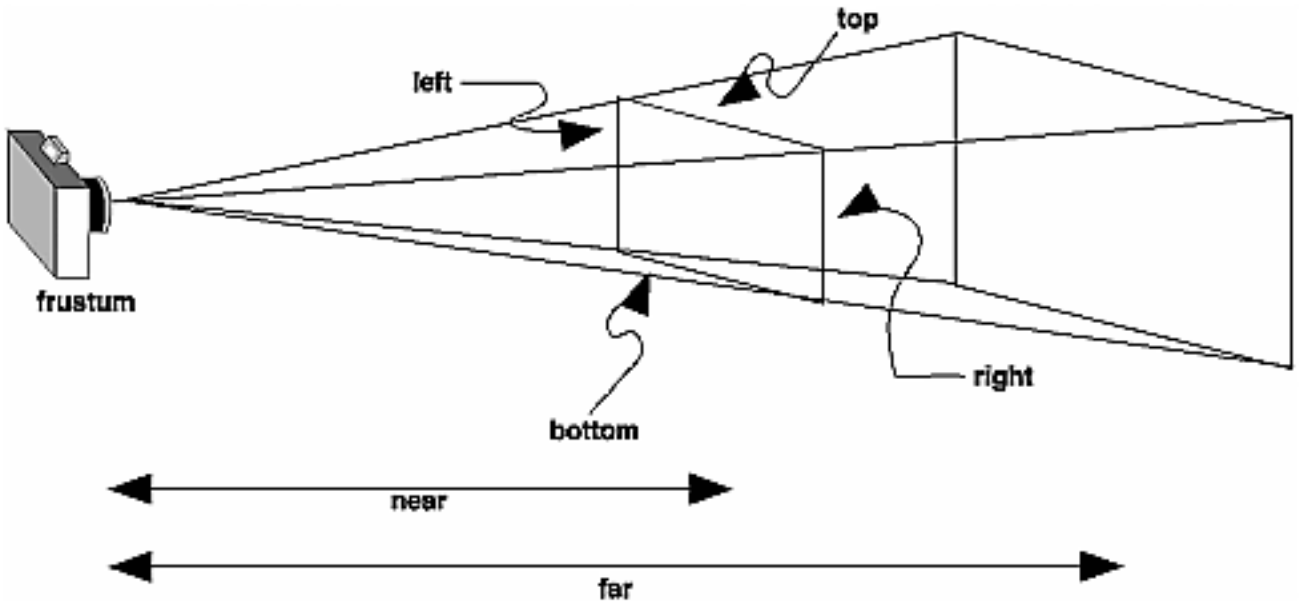
```
glOrtho (left, right,
         bottom, top,
         near, far);
```

  - *Obs.: near e far são valores <u>positivos</u> tipicamente*



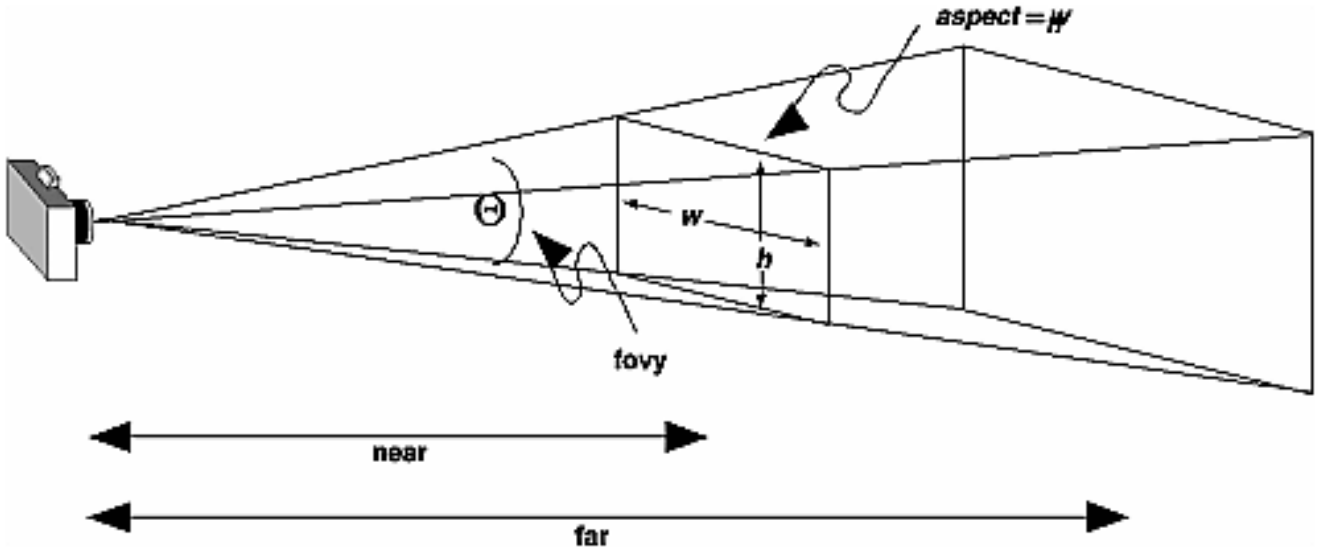
Projecção em Perspectiva

- Volume de visualização especificado com `glFrustum(left,right,bottom,top,near,far);`
- *Não gera necessariamente um v.v. simétrico*



Projecção Perspectiva

- Alternativamente, pode-se usar a rotina `gluPerspective (fovy, aspect, near, far);`
- Gera volume de visualização simétrico centrado sobre o eixo z



Receita para evitar “ecrãs pretos”

- Matriz de projecção especificada com `gluPerspective()`
  - Tentar levar em conta a razão de aspecto da janela (parâmetro `aspect`)
  - Usar sempre `glLoadIdentity()` *antes*
  - Não colocar *nada depois*
- Matriz de visualização especificada com gluLookAt
  - Usar sempre glLoadIdentity () *antes*
  - Outras transformações usadas para mover / instanciar os objectos aparecem *depois*

Exemplo

```
void resize( int w, int h )
{
   glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   gluPerspective( 65.0, (GLdouble) w / h,
                   1.0, 100.0 );
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   gluLookAt( 0.0, 0.0, 5.0,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0 );
}
```