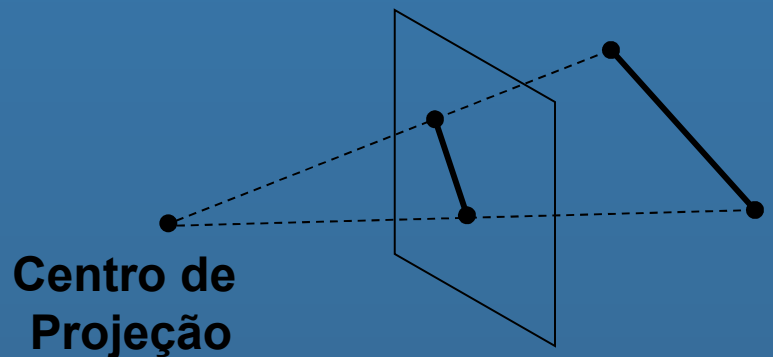
Introdução à Computação Gráfica
Projeções

Jonh Edson R. de Carvalho

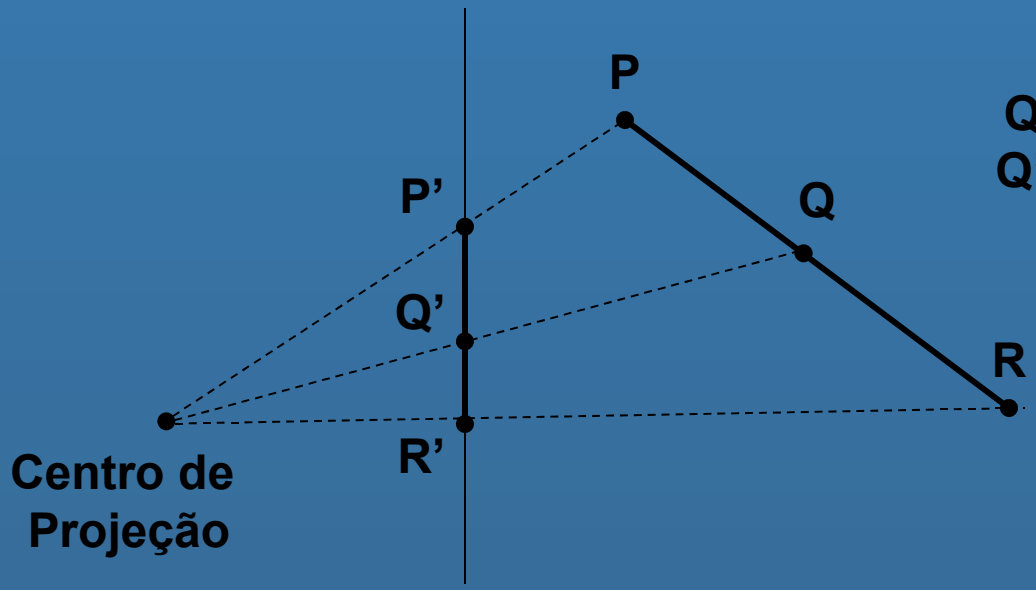
Perspectiva

- É o estudo de transformações projetivas
- O problema consiste em projetar pontos no espaço *d* dimensional no plano *d*-1 dimensional usando um ponto especial chamado de centro de projeção



Transformações Projetivas

- Transformações projetivas transformam retas em retas mas não preservam combinações afim

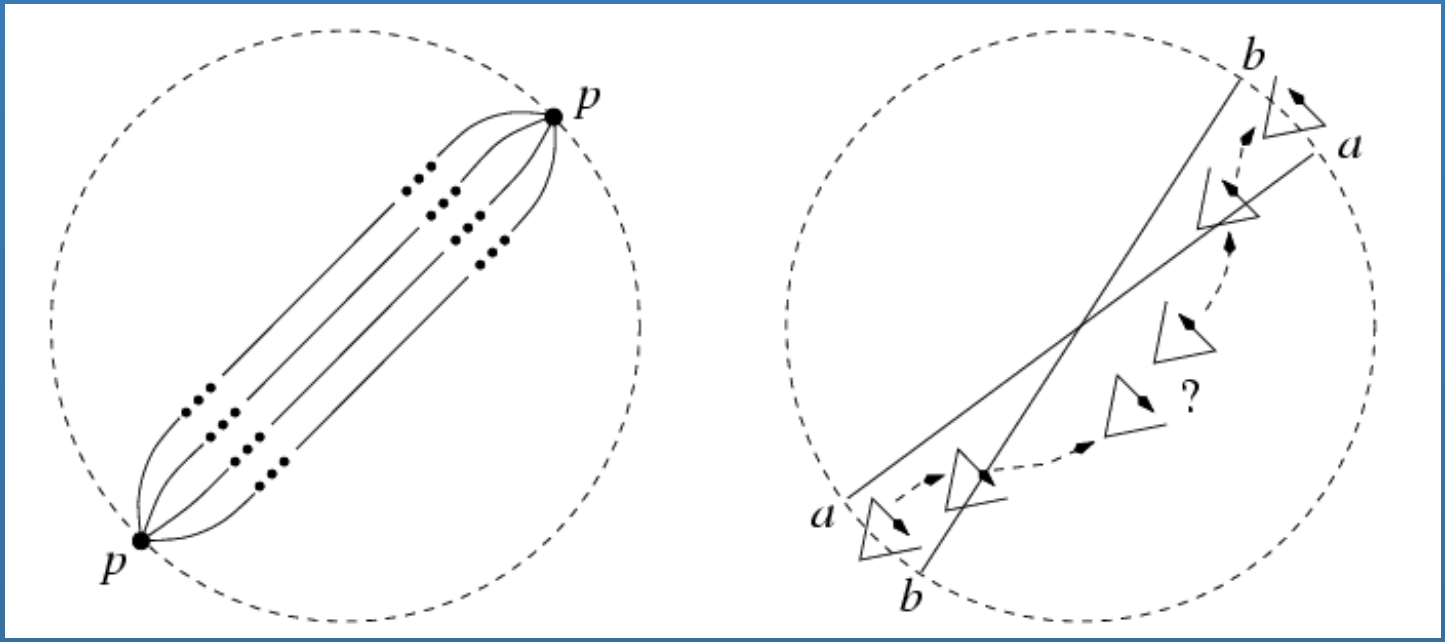


Q é o ponto médio de PR, mas Q' não é o ponto médio de P'R'

Geometria Projetiva

- Geometria euclidiana: duas retas paralelas não se encontram
- Geometria projetiva: assume-se a existência de pontos *ideais* (no infinito)
  - Retas paralelas se encontram num ponto ideal
  - Para não haver mais de um ponto ideal para cada inclinação de reta, assume-se que o plano projetivo se fecha sob si mesmo
  - Em 2D o plano projetivo tem uma borda que é uma reta no infinito (feita de pontos ideais)
  - Transformações projetivas podem levar pontos ideais em pontos do plano euclidiano e vice-versa
  - Problemas: O plano projetivo é uma variedade não orientável
- (Vamos usar geometria projetiva apenas para projetar pontos)

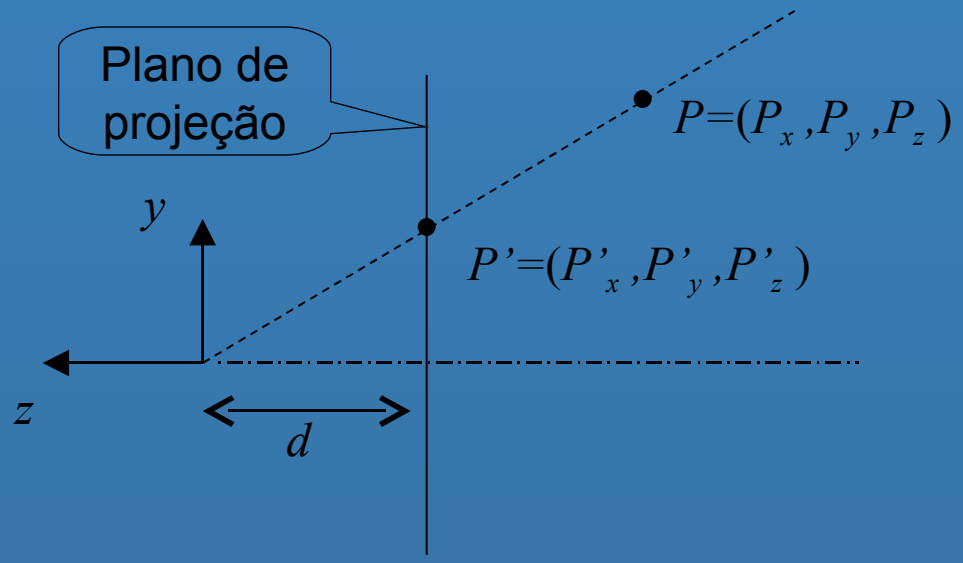
Geometria Projetiva



Coordenadas homogêneas em espaço projetivo

- Representamos apenas pontos (não vetores)
- 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 homogêneas $[x \quad y \quad w]^T$, sua representação canônica é dada por $[x/w \quad y/w \quad 1]^T$. Chamamos a essa operação de *divisão perspectiva*
- Considere os pontos sobre a reta $x=y$: (1,1), (2,2), etc
  - Podemos representá-los em c.h. por $[1\ 1\ 1]^T$, $[1 \quad 1 \quad ½]^T$, etc
  - Claramente, o ponto ideal dessa reta é dado por $[1\ 1\ 0]^T$

Transformações projetivas



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

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

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

Transformação perspectiva em coordenadas homogêneas

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

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

Perspectiva - Sumário

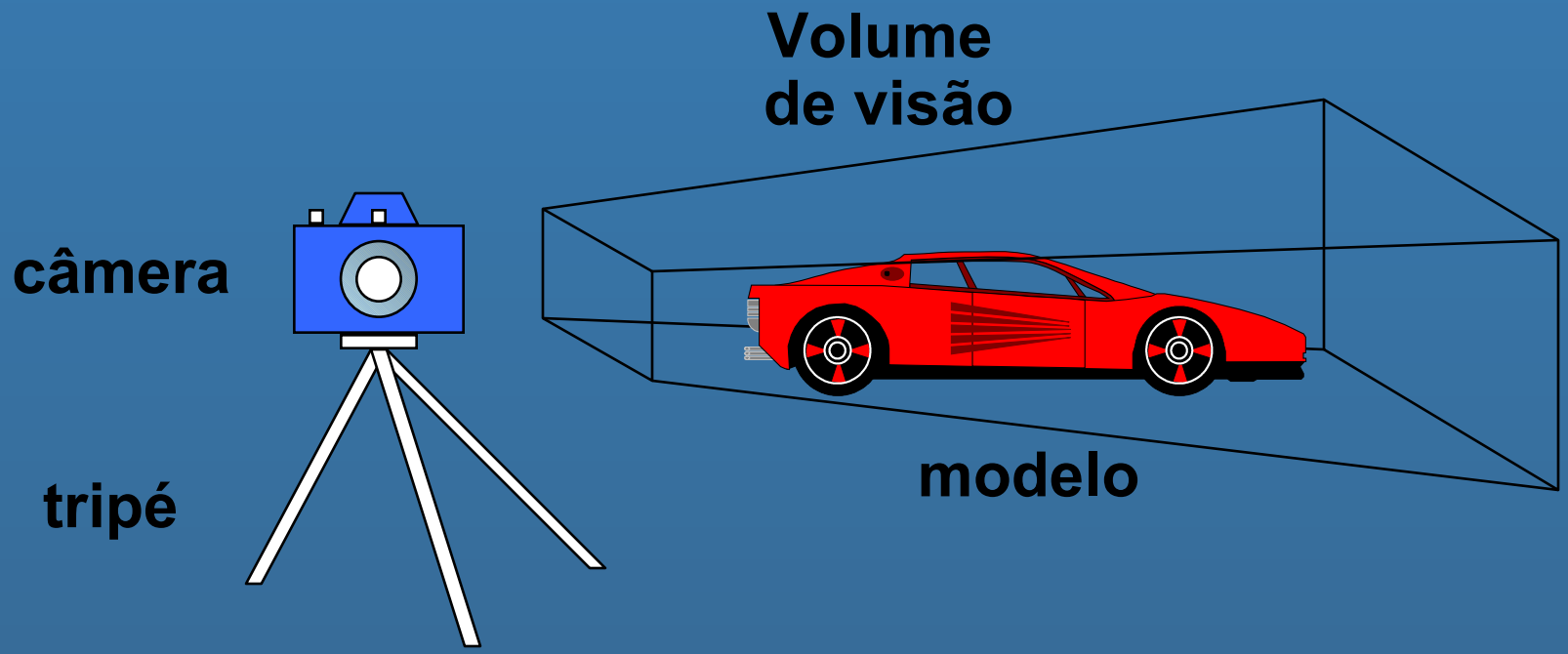
- Para fazer projeção perspectiva de um ponto *P*, segue-se os seguintes passos
  - *P* é levado do espaço euclidiano para o projetivo
    - Trivial – mesmas coordenadas homogêneas
  - *P* é multiplicado pela matriz de transformação perspectiva resultando em *P'*
  - *P'* é levado novamente ao espaço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

Projeção genérica

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

Modelo de câmera sintética

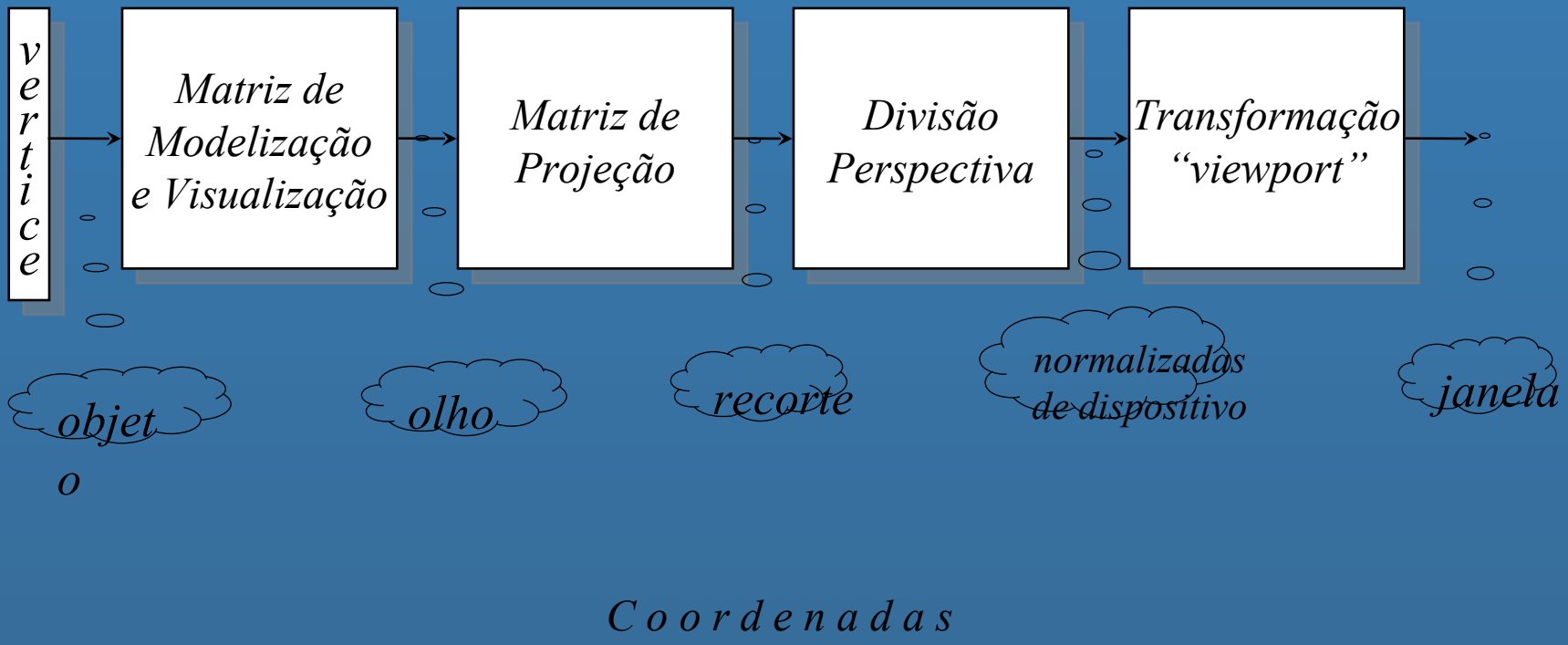
- OpenGL utiliza uma analogia comparando visualização 3D com tirar fotografias com uma câmera



Transformações em OpenGL

- Modelagem
  - Mover /deformar os objetos
- Visualização
  - Mover e orientar a câmera
- Projeção
  - Ajuste da lente / objetiva da câmera
- "*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 projeção é paralela sobre o plano *x-y*
    - O mundo visível é restrito ao cubo $-1 \leq x,y,z \leq 1$
  - A transformação "*viewport*" mapeia o quadrado $-1 \leq x,y \leq 1$ (em coordenadas normalizadas de dispositivo) na superfície total da janela

Especificando o *Viewport*

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

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

  - (parâmetros em *pixels,* sendo que (0,0) refere-se 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

Especificando 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*)
- Existem uma série de funções para operar com a pilha corrente, incluindo

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

Transformando objetos

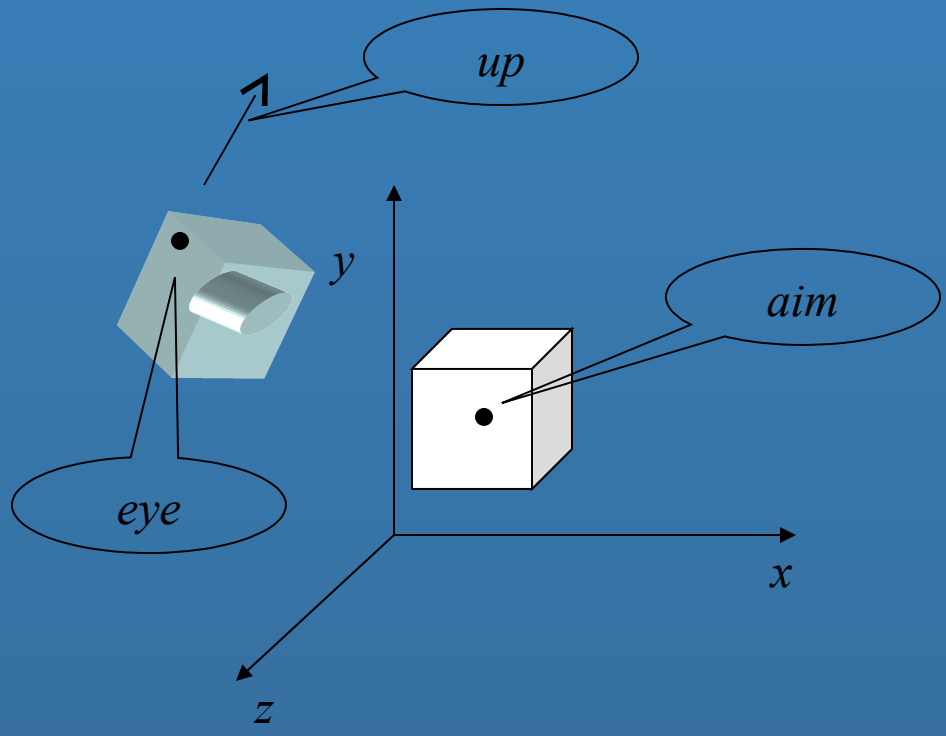
- Usa-se funções para multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros
  - **`glTranslatef ( x, y, z )`**
  - **`glRotatef (ângulo, x, y, z)`**
  - **`glScale ( x, y, z )`**
- Cuidado: ordem é importante:

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

  - **Objeto é rodado e depois transladado!**

Transformações de Visualização

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

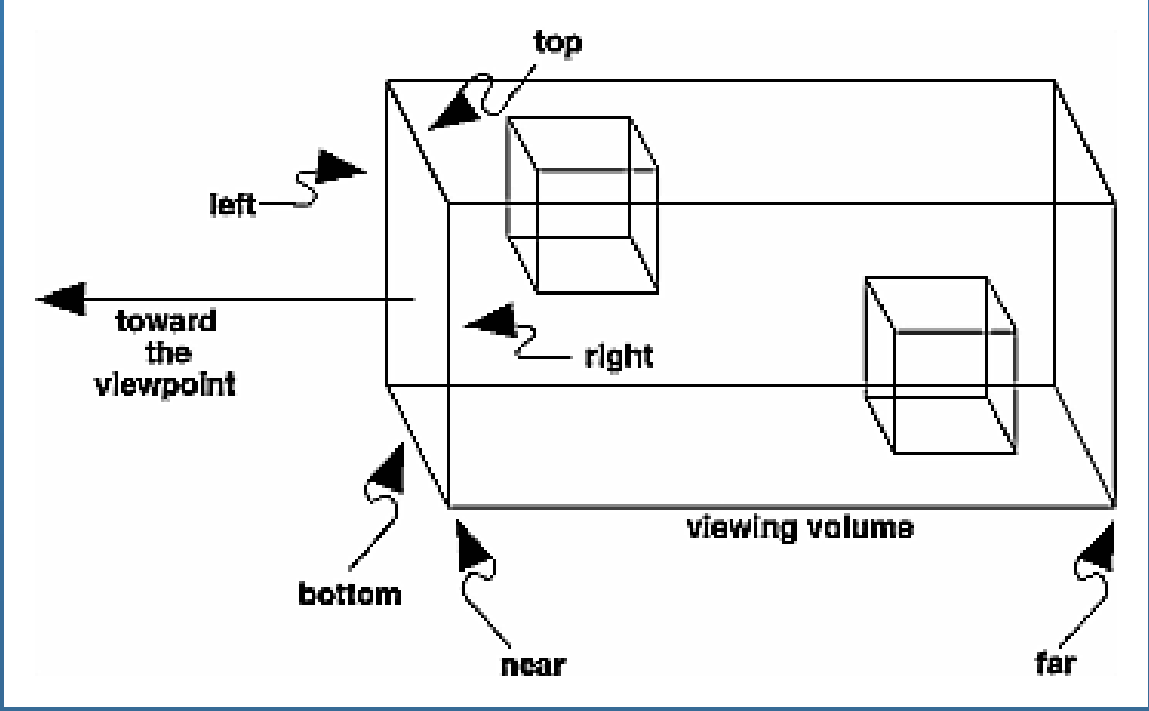


Projeção Paralela

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

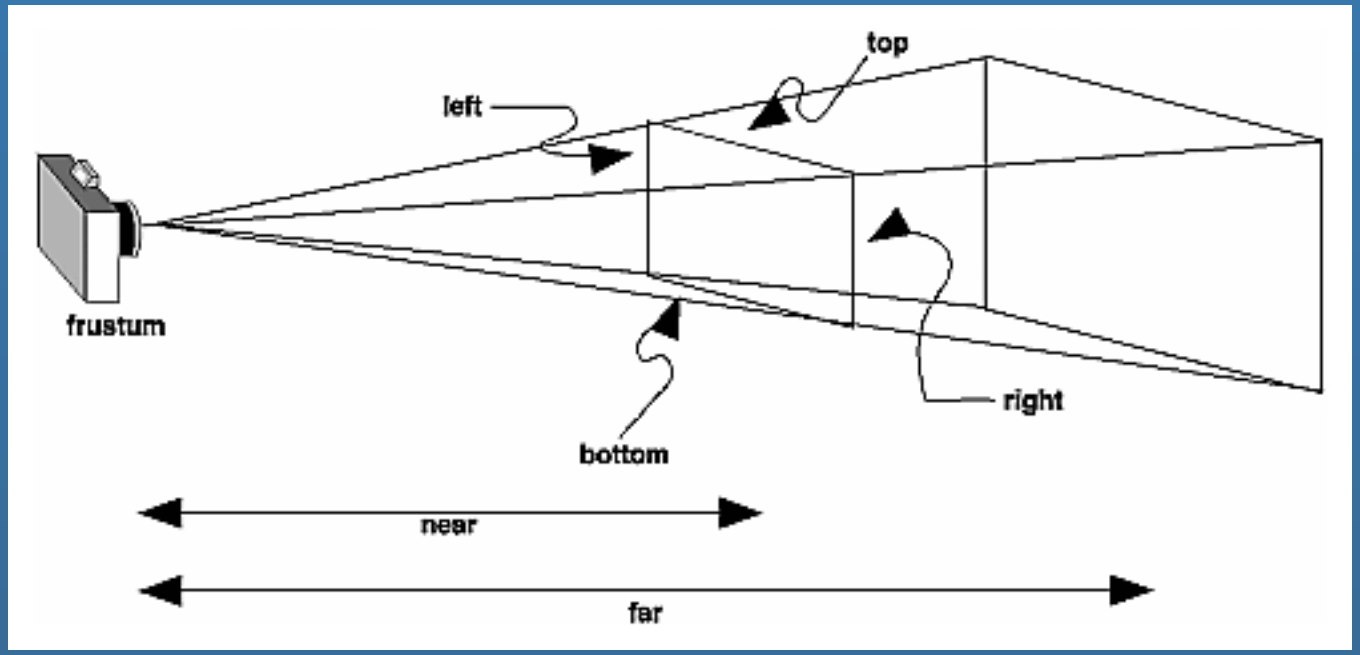
```
glOrtho (left, right,
         bottom, top,
         near, far);
```

- *Obs.: near e far são valores positivos tipicamente*



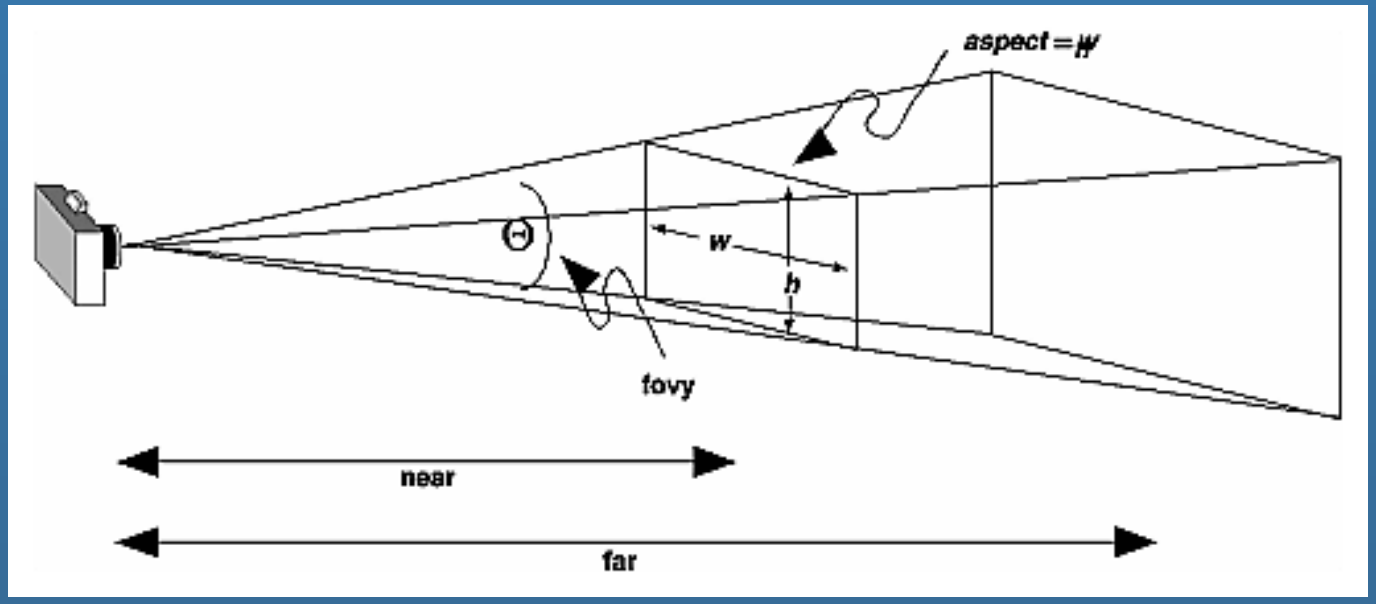
Projeção em Perspectiva

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



Projeção Perspectiva

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



Receita para evitar ‘telas pretas’

- Matriz de projeção especificada com `gluPerspective()`
  - Tentar levar em conta a razão de aspecto da janela (parâmetro `aspect`)
  - Sempre usar `glLoadIdentity()` *antes*
  - Não colocar *nada depois*
- Matriz de visualização especificada com gluLookAt
  - Sempre usar glLoadIdentity () *antes*
  - Outras transformações usadas para mover / instanciar os objetos 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 );
}
```