Visualização

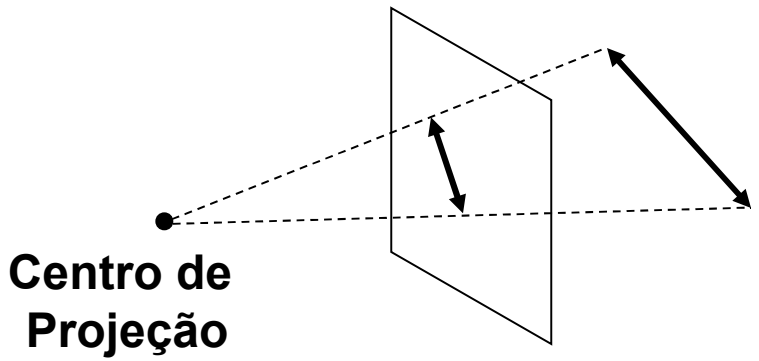
Claudio Esperança
Paulo Roma Cavalcanti

Câmera Virtual

- O processo de visualização do OpenGL define uma câmara virtual.
- Objetos da cena são projetados sobre o plano de projeção (filme virtual) e exibidos na tela.
- Existem vários sistemas de coordenadas envolvidos, de forma a reduzir o problema a uma configuração canônica.

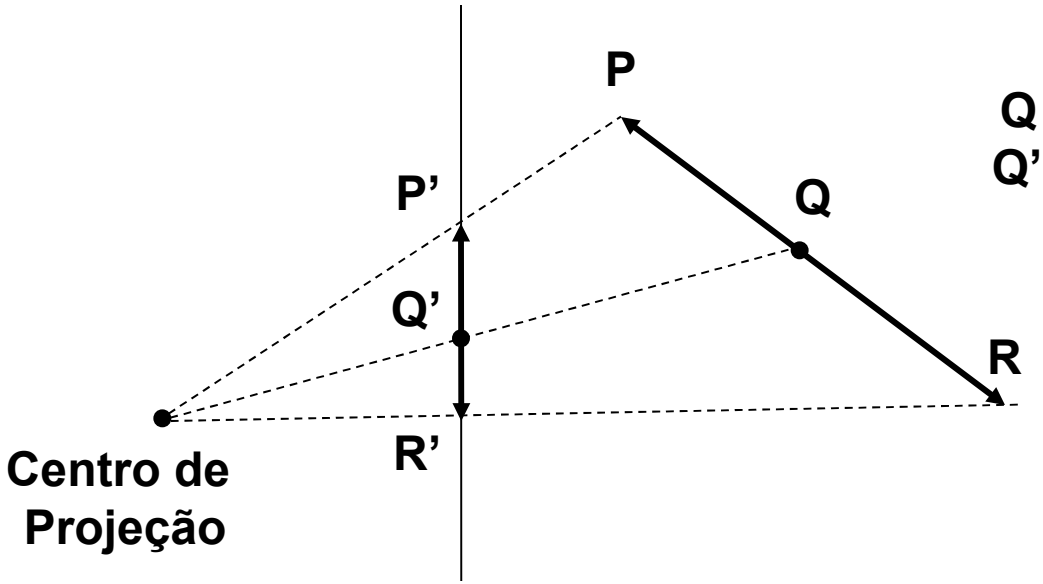
Projeções Planares

- O problema consiste em projetar pontos do espaço *d* dimensional em um espaço de dimensão *d*-1 (plano de projeção).
  - emprega um ponto especial chamado de centro de projeção.



Transformações Projetivas

- Transformações projetivas levam retas em retas, mas não preservam combinações afim.

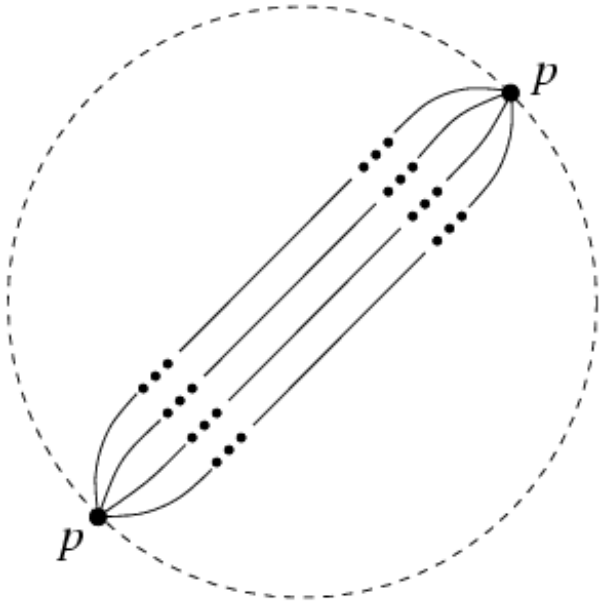


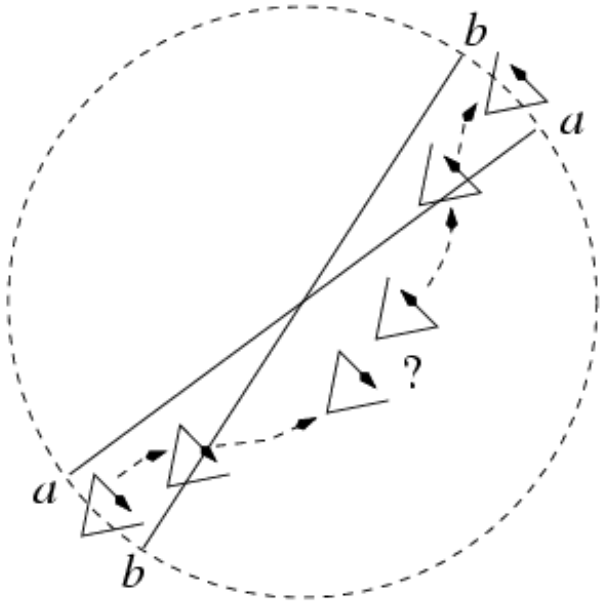
Q é o ponto médio de PR, mas
Q' não é o ponto médio de P'R'

Geometria Projetiva

- Geometria Euclideana: duas retas paralelas não se encontram.
- Geometria Projetiva: não existe paralelismo.
  - Retas paralelas se encontram num ponto ideal (no infinito)
  - 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 (constituída de pontos ideais)
  - Transformações projetivas podem levar pontos ideais em pontos do plano euclidiano e vice-versa
  - Problema: O plano projetivo é uma variedade não orientável

Geometria Projetiva





Coordenadas Homogêneas em Espaço Projetivo

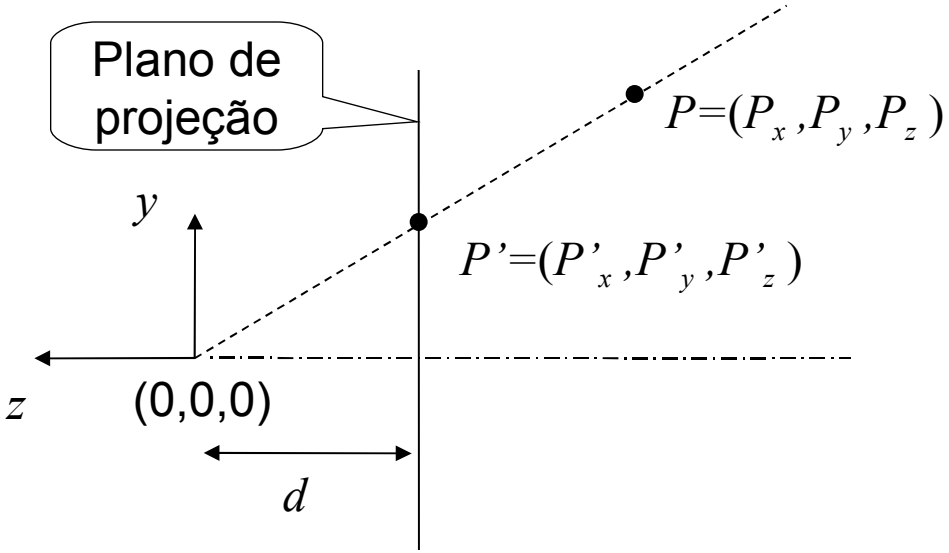
- Não há distinção entre pontos e vetores.
- Em 2D, um ponto (x,y) é representado em c.h. pelo vetor coluna $[x \cdot w \quad y \cdot w \quad w]^T$, para $w \neq 0$
  - Assim, o ponto (4,3) pode ser representado por

    $[8\ 6\ 2]^T$, $[12\ 9\ 3]^T$, $[-4\ -3\ -1]^T$, etc
- A representação canônica do ponto com coordenadas homogêneas $[x \ \ y \ \ w]^T$, é:
  - $[x/w \ \ y/w \ \ 1]^T$.
  - Chamamos esta operação de *divisão perspectiva.*

Exemplo

- Os pontos sobre a reta $x=y$: (1,1), (2,2), (3,3), ...
  - Podem ser representados em c.h. por:
    - $[1\ 1\ 1]^T$, $[1\ \ 1\ \ ½]^T$, etc
  - O ponto ideal dessa reta é dado por $[1\ 1\ 0]^T$

Transformações Projetivas



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

- A projeção de um ponto P é dada por:

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

- Plano de projeção é:
- perpendicular ao eixo *z*
- está a uma distância *d* do C.P. (0,0,0)
- intercepta o semi eixo z negativo

Transformação Perspectiva em Coordenadas Homogêneas

- Não existe matriz 3x3 capaz de realizar tal transformação em espaços Euclideanos.
  - Porém, no espaço projetivo:

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

Perspectiva - Sumário

- Para fazer projeção perspectiva de um ponto *P*, seguem-se os seguintes passos:
  - *P* é levado do espaço Euclideano para o projetivo.
    - Trivial – mesmas coordenadas homogêneas.
  - *P* é multiplicado pela matriz de transformação perspectiva resultando em *P'*
  - *P'* é levado novamente ao espaço Euclideano
    - Divisão perspectiva (não linear!!).

Projeção Genérica

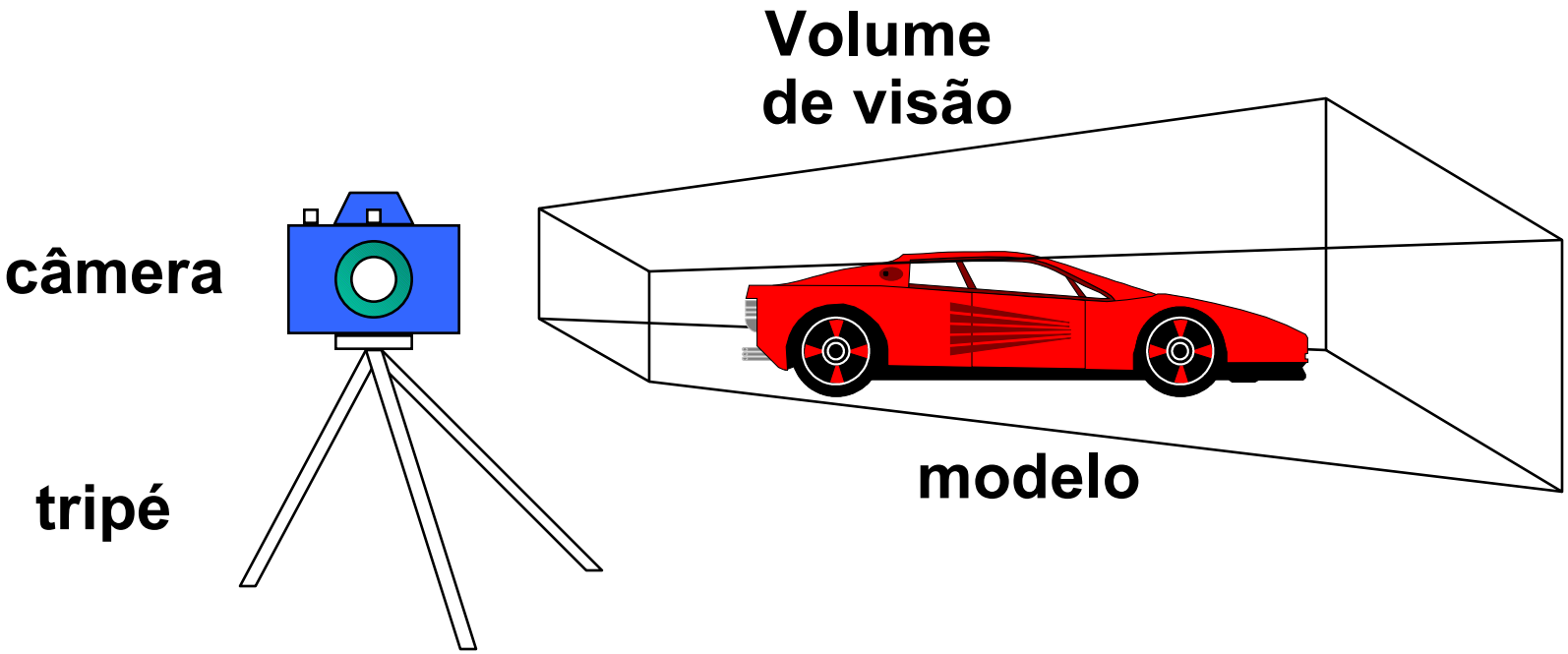
- E se for usado um sistema de coordenadas arbitrário?
  1. Centro de Projeção fora da origem ou
  2. Cena não está posicionada no semi-eixo z negativo.
- Muda-se o sistema de coordenadas.
  - transformações afim posicionam todos os elementos corretamente.
- As maneiras pelas quais essas transformações são executadas caracterizam um dado modelo de projeção.

Espaços de Referência

1. Espaço do objeto.
2. Espaço da cena.
3. Espaço da câmera.
4. Espaço normalizado.
5. Espaço de Ordenação.
6. Espaço da imagem.

Modelo de Câmera Sintética

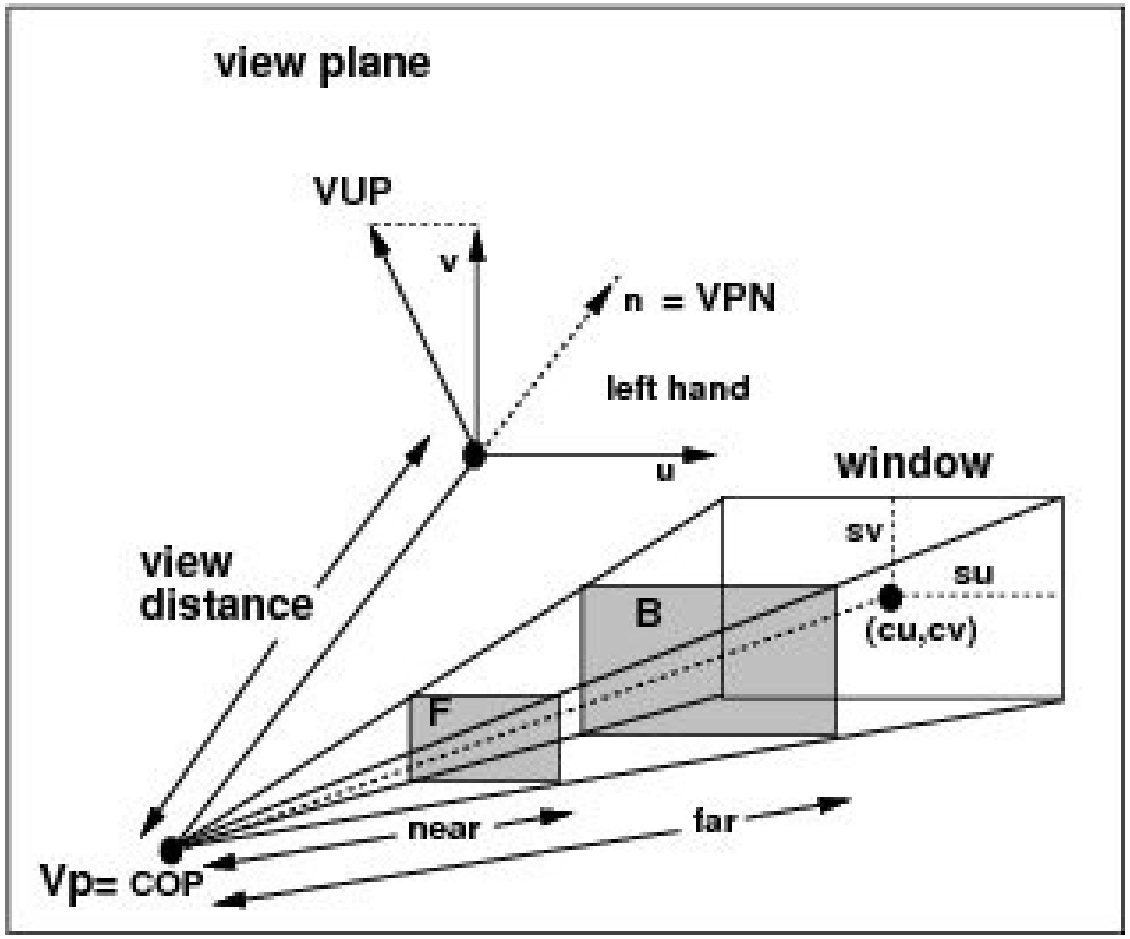
- OpenGL utiliza uma analogia comparando visualização 3D com tirar fotografias com uma câmera



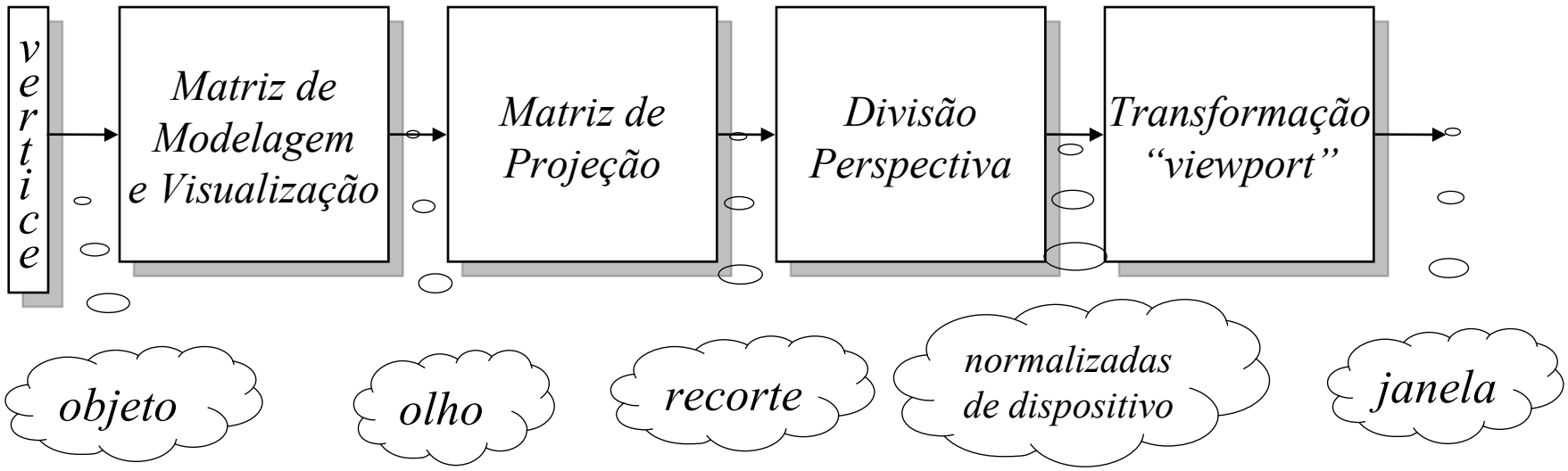
Transformações em OpenGL

- Modelagem
  - Mover /deformar os objetos
- Visualização
  - Mover e orientar a câmera
- Projeção
  - Ajustar a lente / objetiva da câmera
- *"Viewport"*
  - Aumentar ou reduzir a fotografia

Sistema de Coordenadas da Câmera



Pipeline de Transformações do OpenGL



Estado Inicial do *Pipeline*

- As matrizes "*modelview*" e "*projection*" são matrizes identidade:
  - vértices não são transformados
  - 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 a *Viewport*

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

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

  - parâmetros em *pixels*
  - (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* se situam no topo de duas pilhas que são usadas para fazer operações com matrizes.
- Para selecionar a pilha:

  glMatrixMode(*GL_MODELVIEW* ou *GL_PROJECTION*)

- Existe uma série de funções que operam com a pilha corrente, incluindo

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

Transformando Objetos

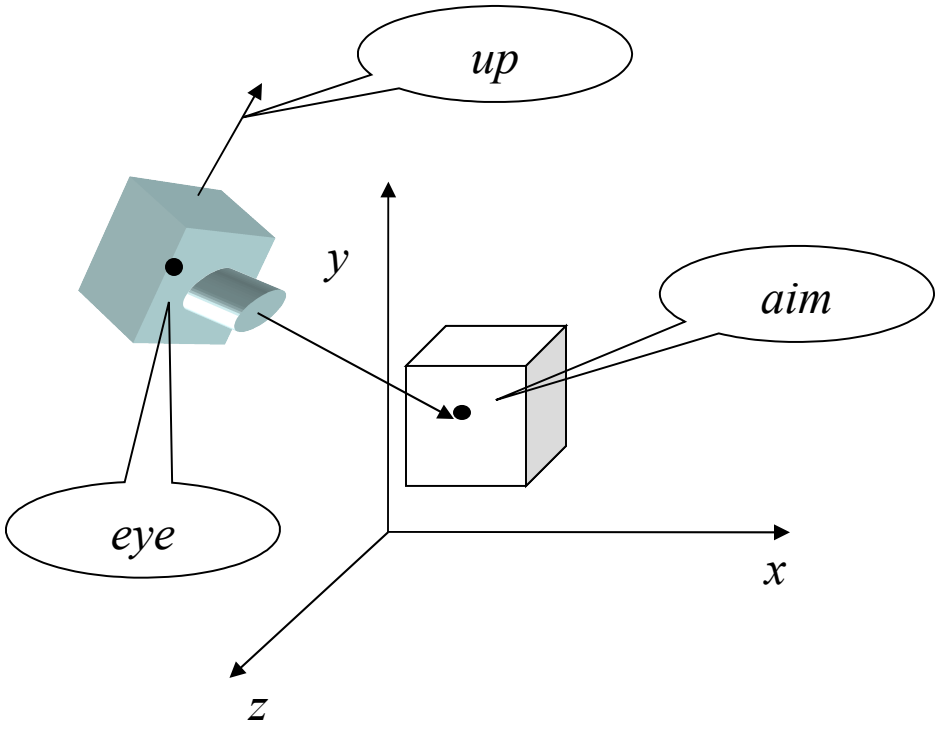
- Para para multiplicar o topo da pilha de matrizes por transformações especificadas por parâmetros:
  - **glTranslatef (*x, y, z*)**
  - **glRotatef (*ângulo, x, y, z*)**
  - **glScale (x, y, z)**
  - Cuidado: ordem é importante:

```
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 = posição da câmera
  - aim = ponto que define a direção de visão
  - up = direção "vertical" da câmera
  - *Cuidado com casos degenerados*

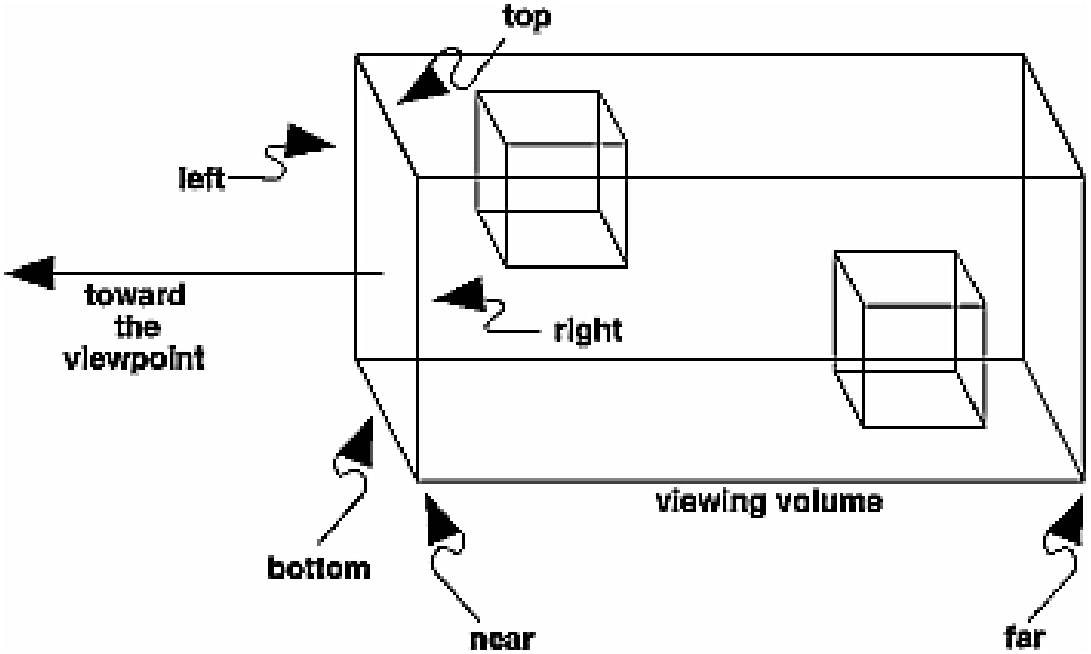


Projeção Paralela

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

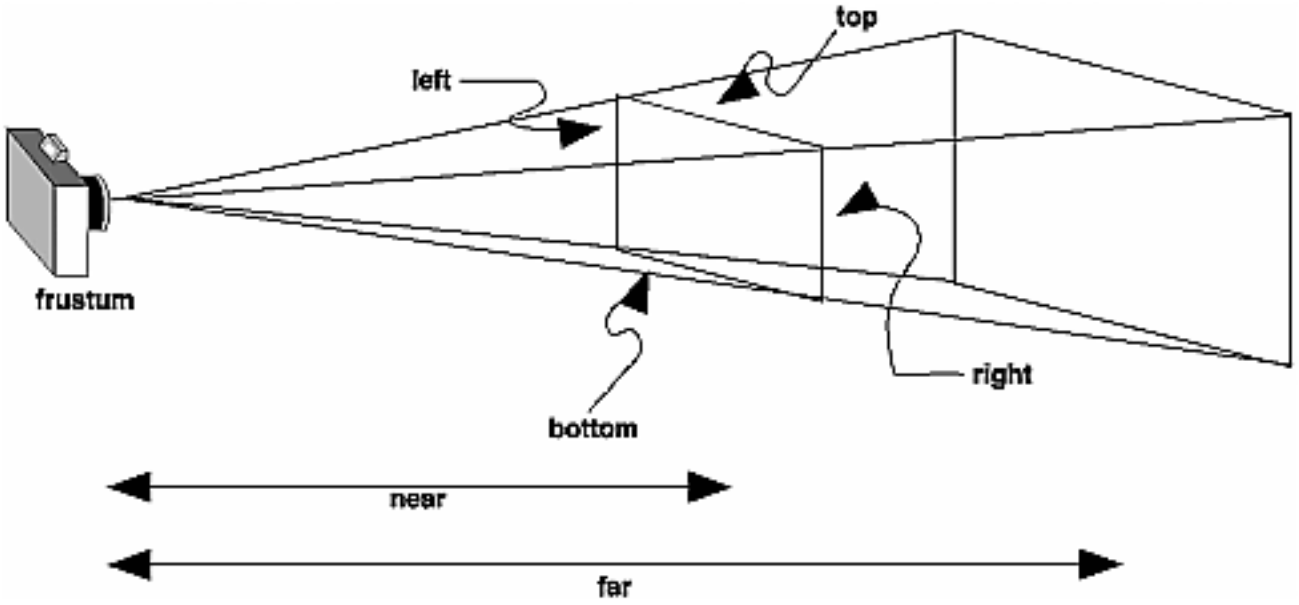
```
glOrtho (left, right,
         bottom, top,
         near, far);
```

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



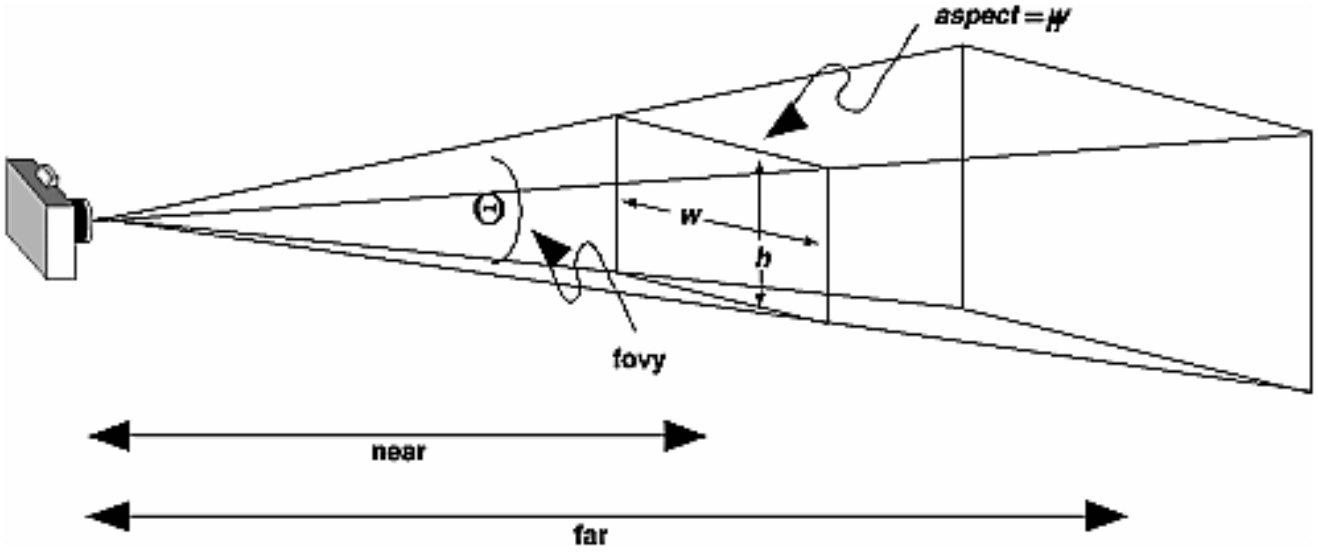
Projeção em Perspectiva

- 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 um volume de visão <u>simétrico</u> (direção de visão perpendicular ao plano de projeção).



Receita Para Evitar ‘telas pretas’

- Especificar Matriz de projeção com `gluPerspective()`
- Tentar levar em conta a razão de aspecto da janela (parâmetro `aspect`)
  - Sempre usar `glLoadIdentity()` *antes*
  - Não colocar *nada depois*
- Especificar Matriz de visualização 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 );
}
```