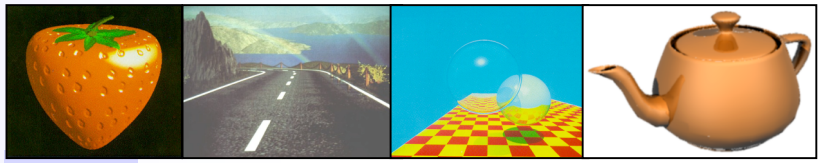


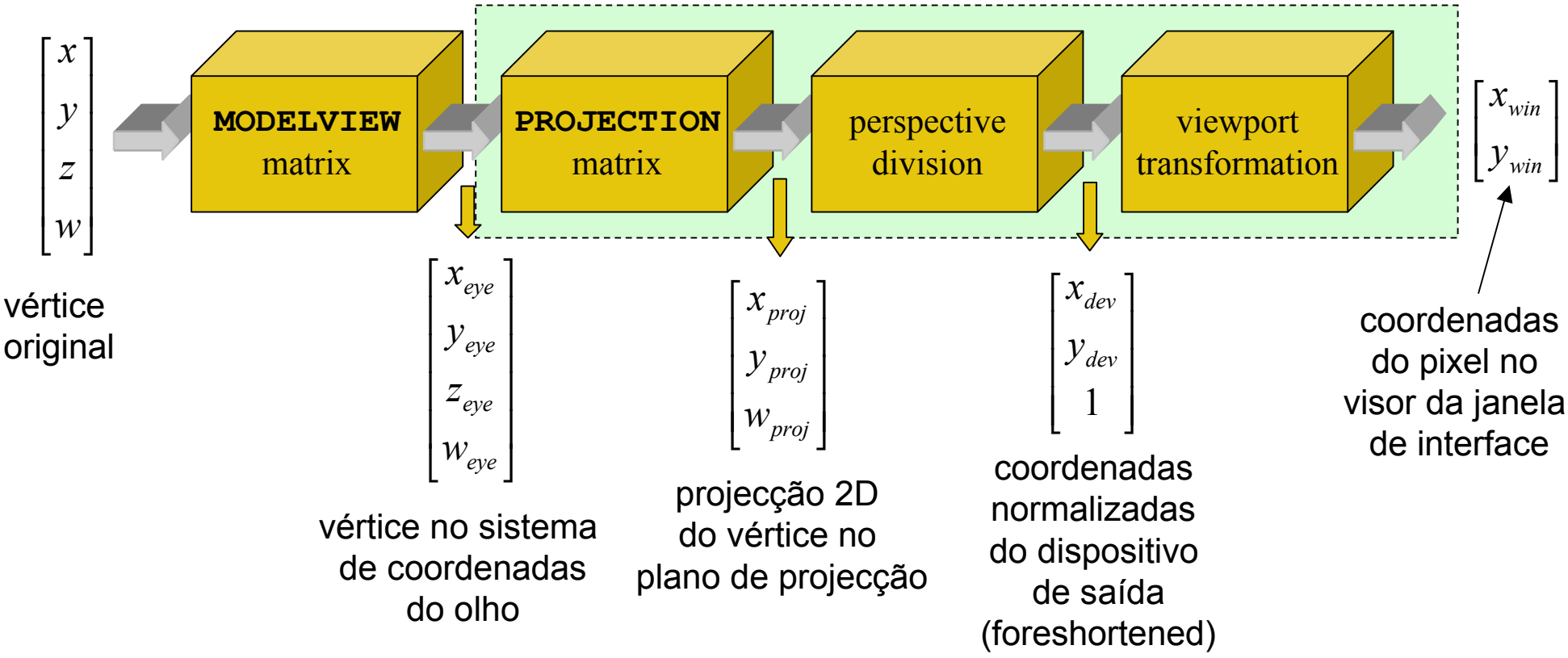
Cap. 4
Projecções e Visualização 3D

Ensino de Informática (3326) - 4º ano, 2º semestre
Matemática (5828) - 2º ano, 2º semestre
Engenharia Informática (5385) - 2º ano, 2º semestre

Bibliografia

- "*Computer graphics: principles & practice*", Foley, vanDam, Feiner, Hughes(tem um apêndice sobre álgebra linear)
- *"Advanced Animation and Rendering Techniques"*, Watt and Watt
- *"The OpenGL Programming Guide"*, Woo, Neider & Davis
- *"Interactive Computer Graphics",* Edward Angel

Pipeline de Renderização em OpenGL®



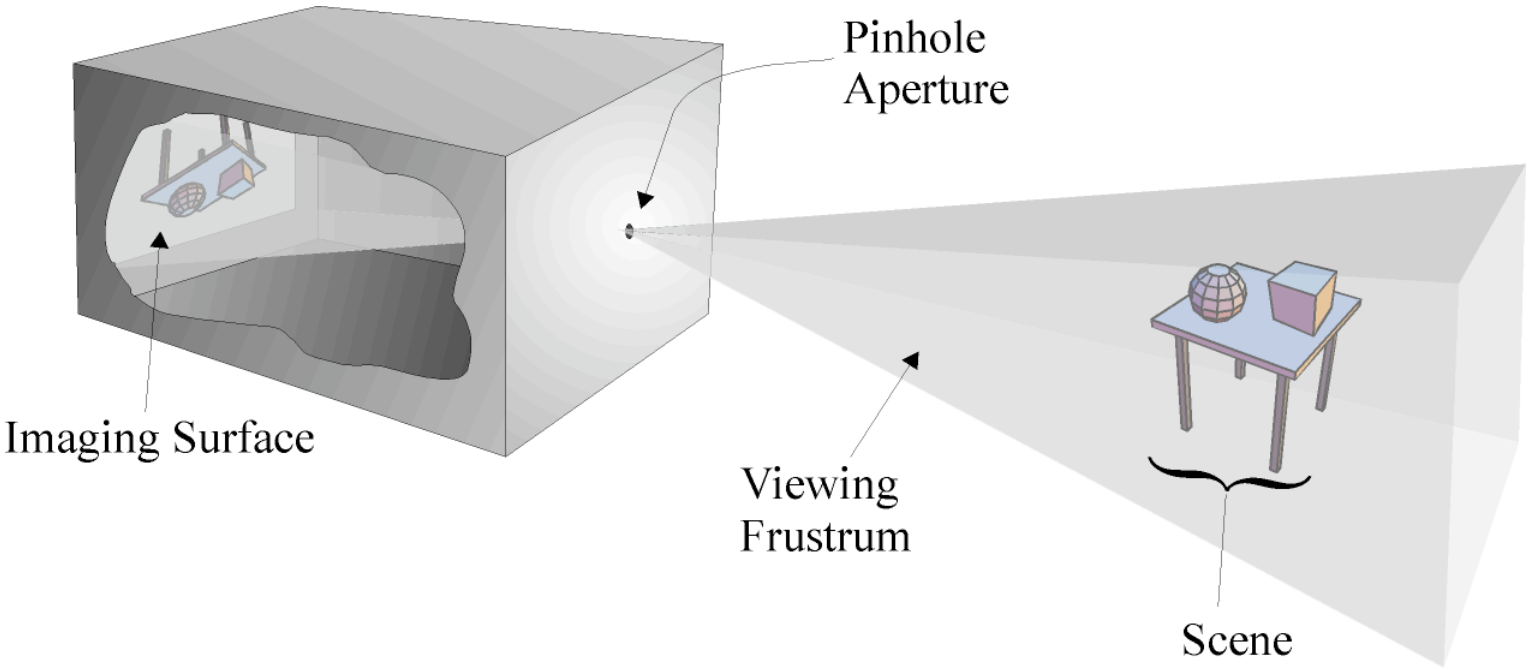
Sistema de câmara

- Para criar uma vista duma cena é necessário:
  - **uma descrição da geometria da cena**
  - **uma câmara ou definição do ponto de vista (ou observador)**
  - **um plano de projecção**
- Por omissão, a câmara OpenGL está localizada na origem e direccionada no sentido do eixo ***z* negativo**.
- A definição da câmara permite a *projecção* da geometria da cena 3D numa superfície 2D para efeitos de saída gráfica.
- Esta projecção pode ser feita de várias maneiras:
  - ***ortogonal* (paralelismo das linhas é preservado)**
  - ***perspectiva*: *1-ponto*, *2-pontos* ou *3-pontos***
  - ***ortogonal oblíqua***

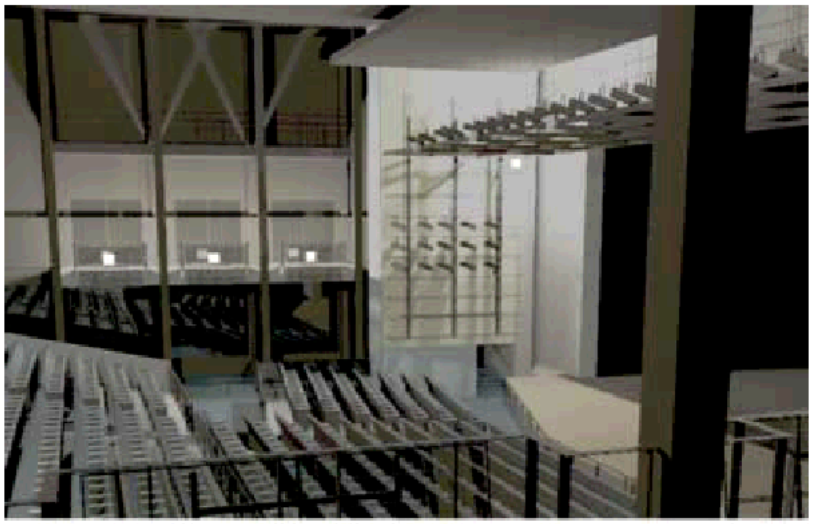
Tipos de câmara

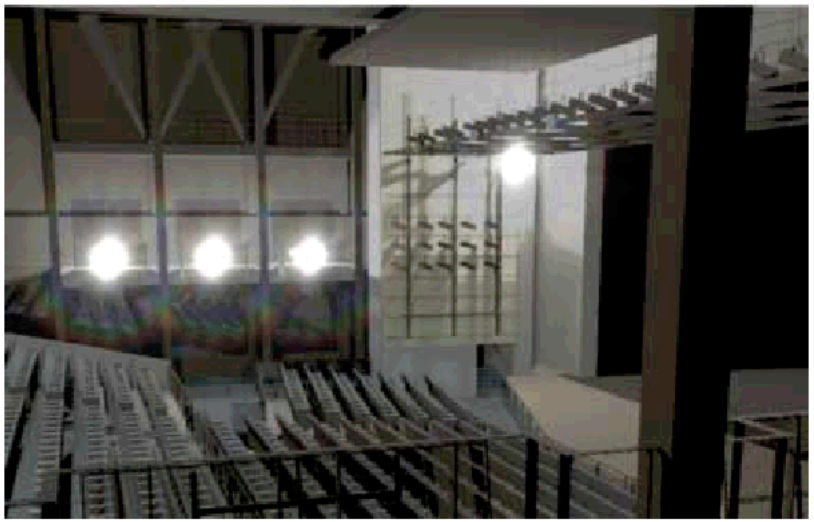
- Antes de gerar uma imagem temos que escolher o observador:
- M*odelo da câmara clássica* (*pinhole camera model*) é o mais usado:
  - profundidade infinita do campo (infinite *depth of field*): tudo é focado
- *Modelo de câmara dos sistemas avançados de renderização*
  - *lentes duplas de Gauss* são usada por muitas câmaras profissionais
  - modela a profundidade de campo e óptica não-linear (incluindo *lens flare*)
- *Sistemas de renderização foto-realística* empregam muitas vezes o modelo físico do olho humano para renderizar imagens
  - modela a resposta dos olhos face aos níveis de *brilho* e *cor*
  - modela a óptica interna do próprio olho (*difracção* pelas fibras da lente etc.)

Modelo da câmara clássica

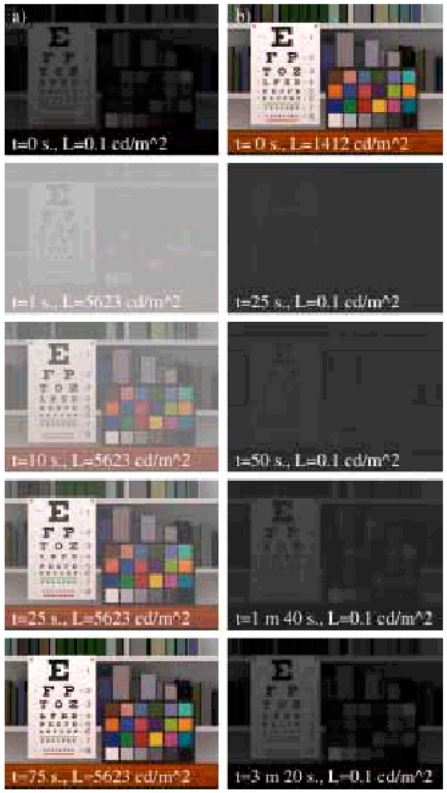


Renderização foto-realística baseada na resposta do olho



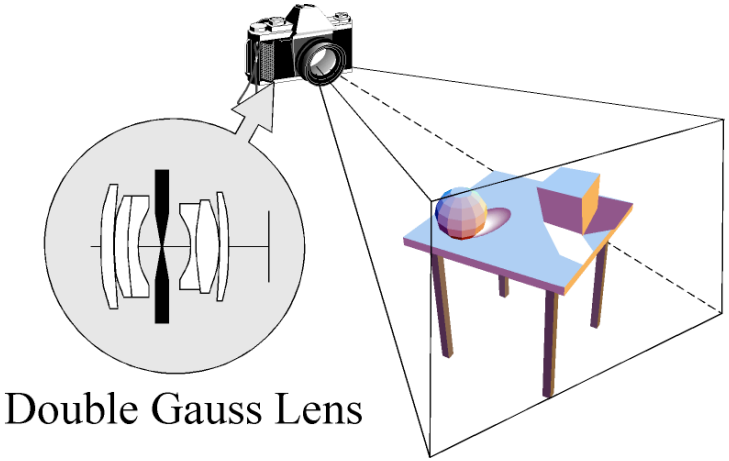


Glare & Difracção



Adaptação

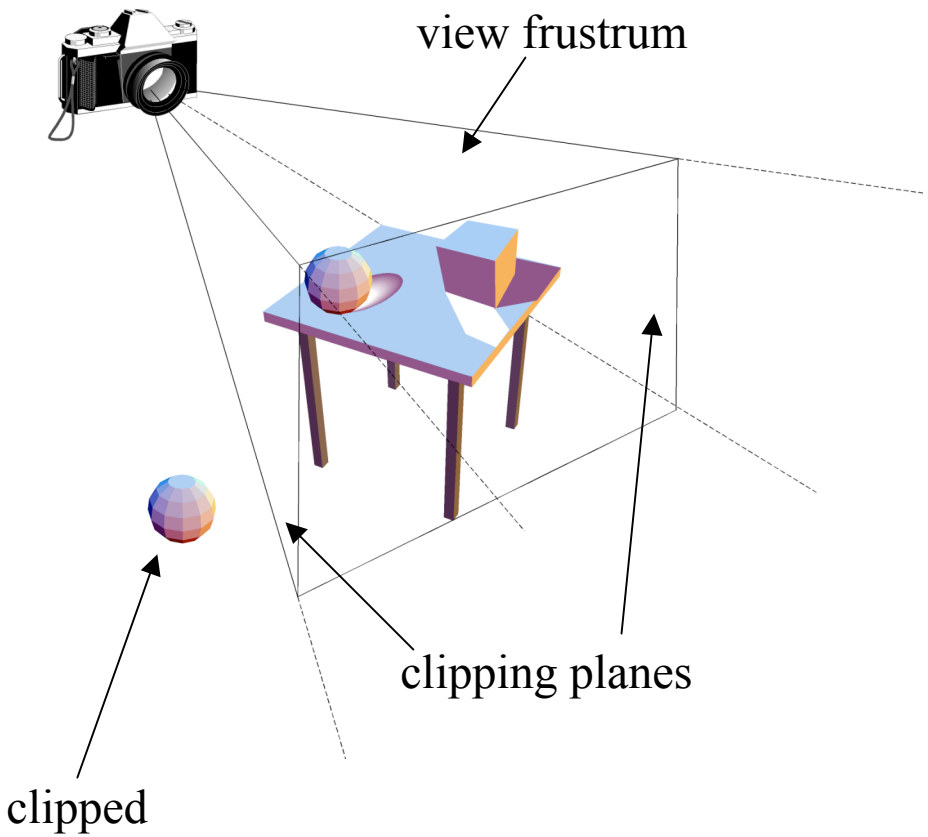
Sistemas baseados na câmara de lentes duplas



Um modelo de câmara implementado na Princeton University (1995)

Sistema de visualização

- Nesta altura só estamos preocupados com a *geometria* da visualização.
- A posição e a orientação da câmara definem um *view-volume* ou *view-frustrum*.
  - **objectos completa ou parcialmente dentro deste volume serão potencialmente visíveis no visor (viewport).**
  - **objectos completamente fora deste volume não podem ser vistos ⇒ *clipped***



Modelos da câmara

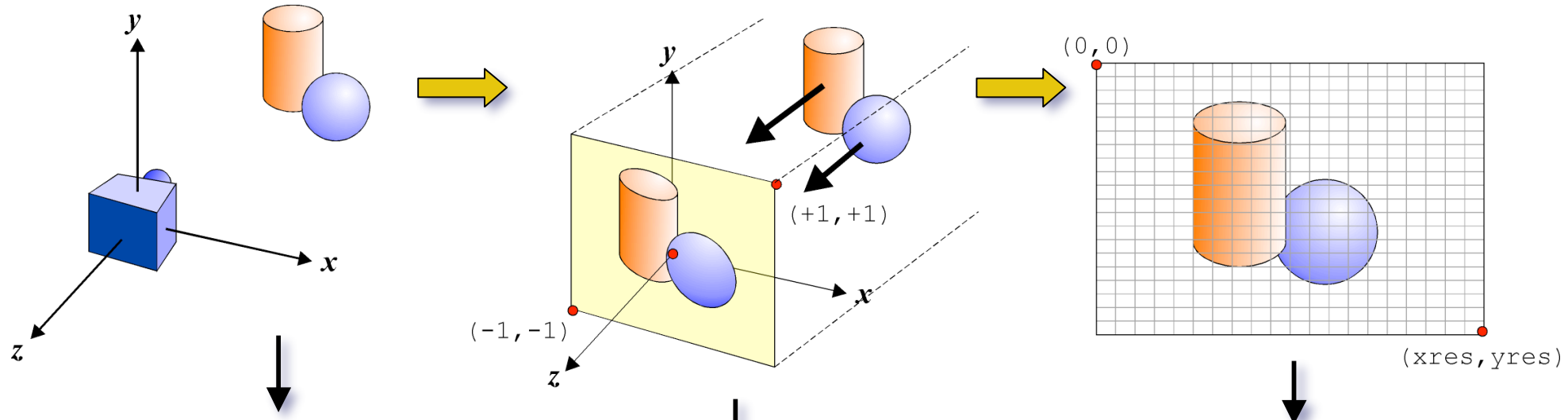
- Cada vértice tem de ser projectado no plano da janela 2D da câmara (plano de projecção) por forma a visualizá-lo no ecrã.
- A *CTM* é empregue para determinar a localização de cada vértice no sistema de coordenadas da câmara:

$$\vec{x}' = \mathbf{M}_{CTM}\,\vec{x}$$

- Depois, aplicamos a matriz de projecção definida por `GL_PROJECTION` para mapear coordenadas da câmara para as coordenadas da janela 2D do plano de projecção.
- Finalmente, estas coordenadas 2D são mapeadas para as coordenadas do dispositivo de saída através da utilização da definição dum visor (viewport) na janela de interface (dado por `glViewport()`).

Modelação da câmara em OpenGL ®

sistema de coordenadas da câmara → sistema de coordenadas da janela 2D → sistema de coordenadas do dispositivo/ecrã



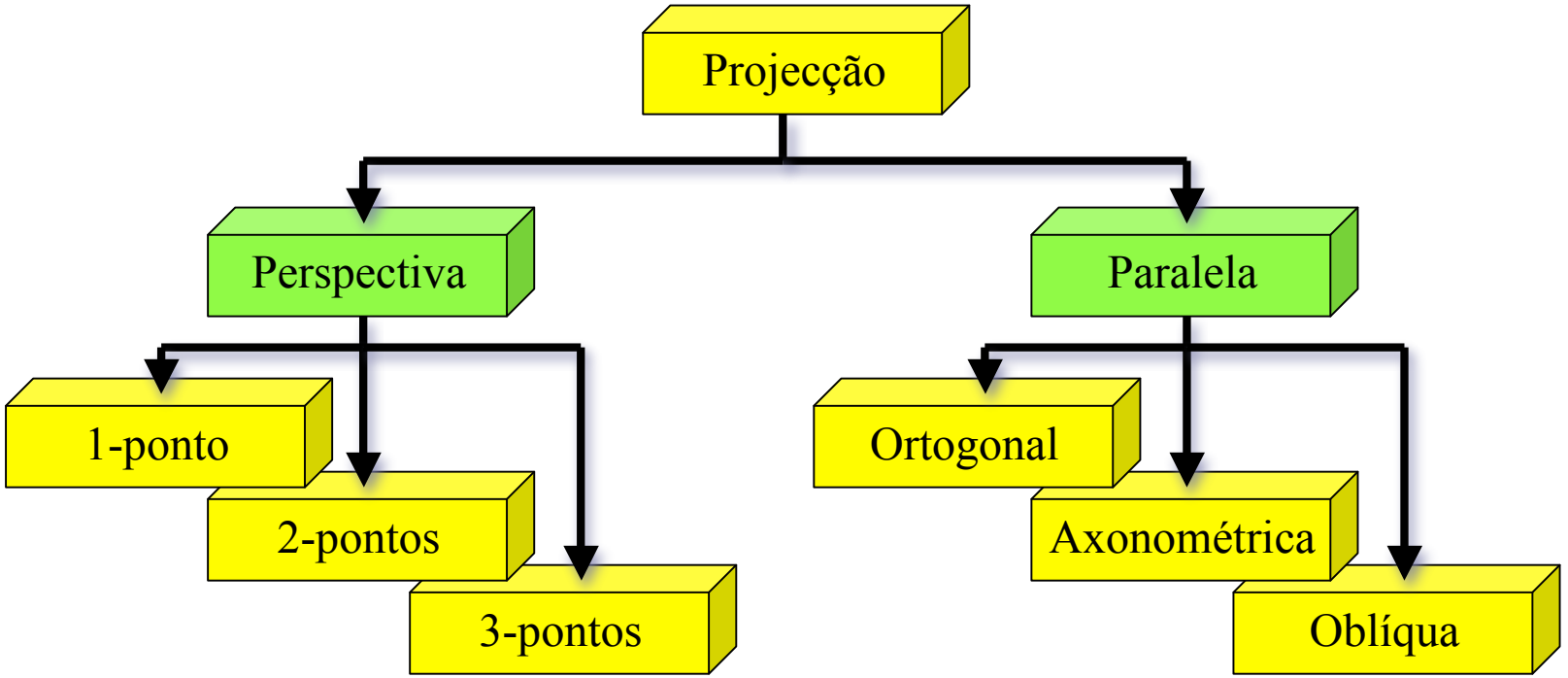

```
glMatrixMode(GL_MODELVIEW)
...
```

```
glMatrixMode(GL_PROJECTION)
...
```

```
glViewport(0,0,xres,yres)
```

Projecção 3D → 2D

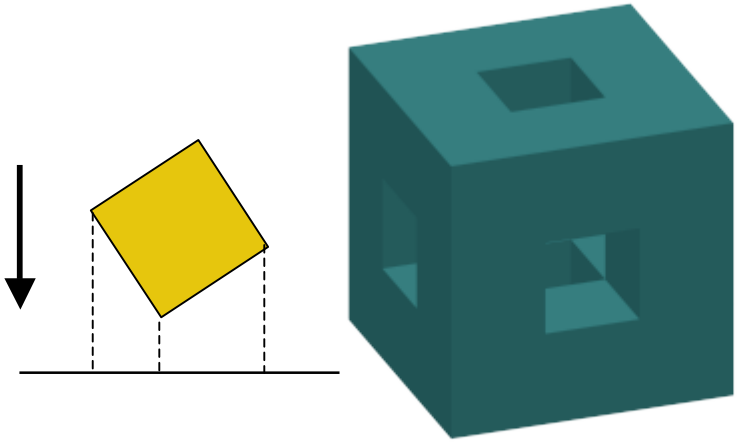
- Tipo de projecção depende dum conjunto de factores:
  - *localização* e *orientação* do plano de projecção (janela de visualização)
  - *direcção da projecção* (descrita por um vector)
  - *tipo de projecção*:



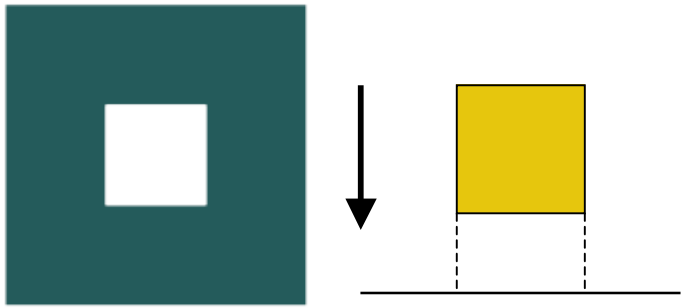
Projecções paralelas

- projectantes são paralelas

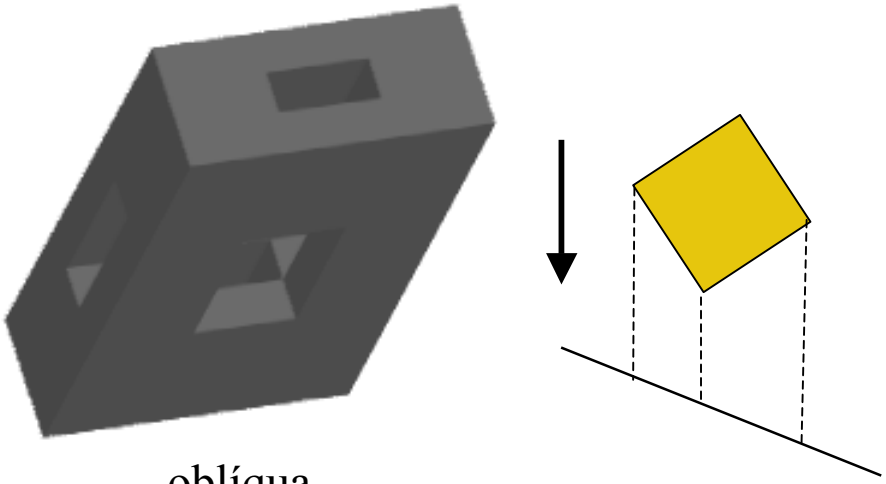
- observador no infinito



axonométrica



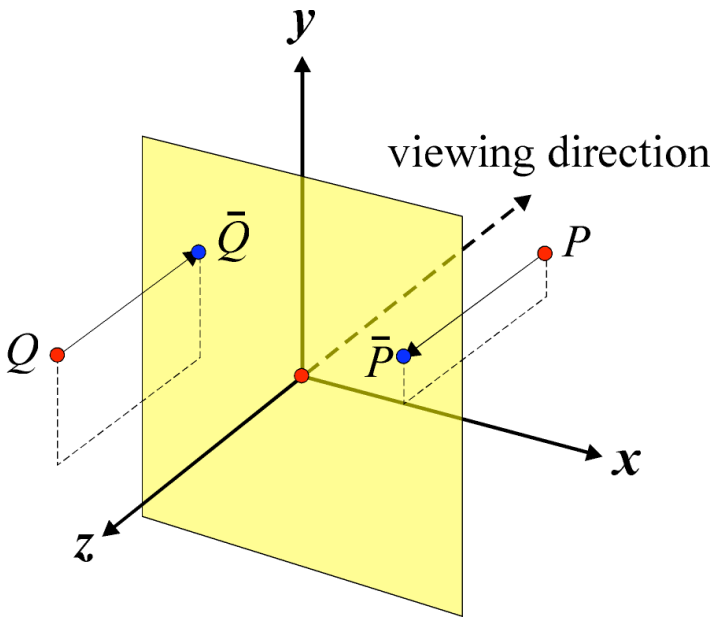
ortogonal



oblíqua

Projecções paralelas ortogonais

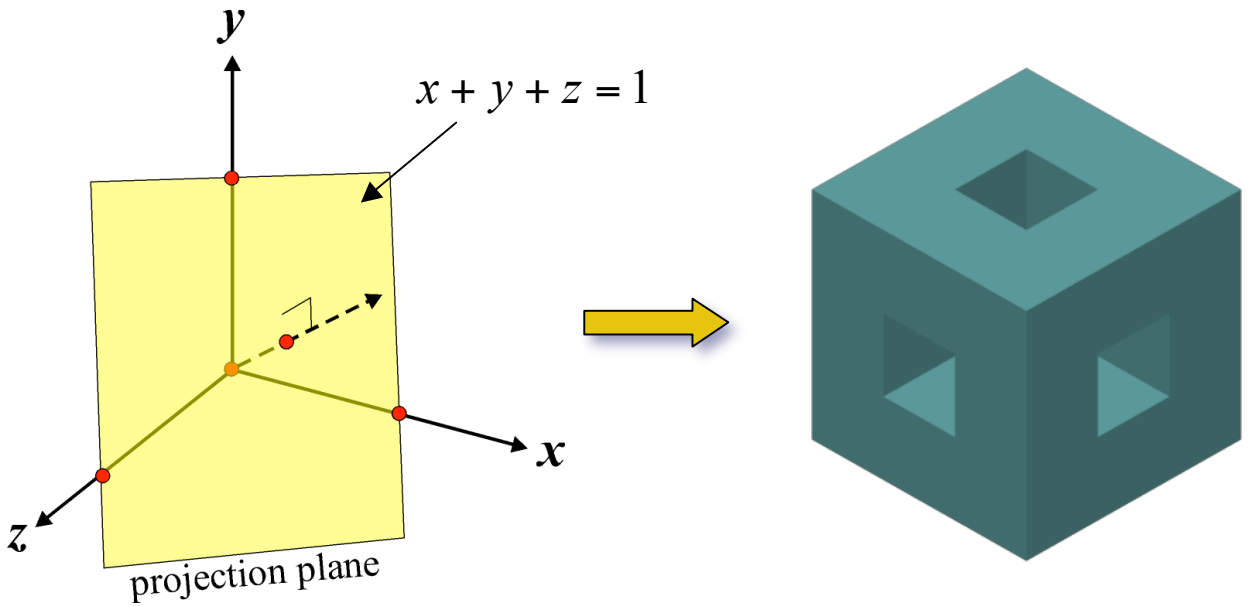
- A mais simples de todas as projecções: as projectantes são perpendiculares ao plano de projecção.
- Normalmente, o plano de projecção está alinhado com os eixos (muitas das vezes em *z=0*)



$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} \Rightarrow \overline{P} = \mathbf{M}P \text{ where } \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

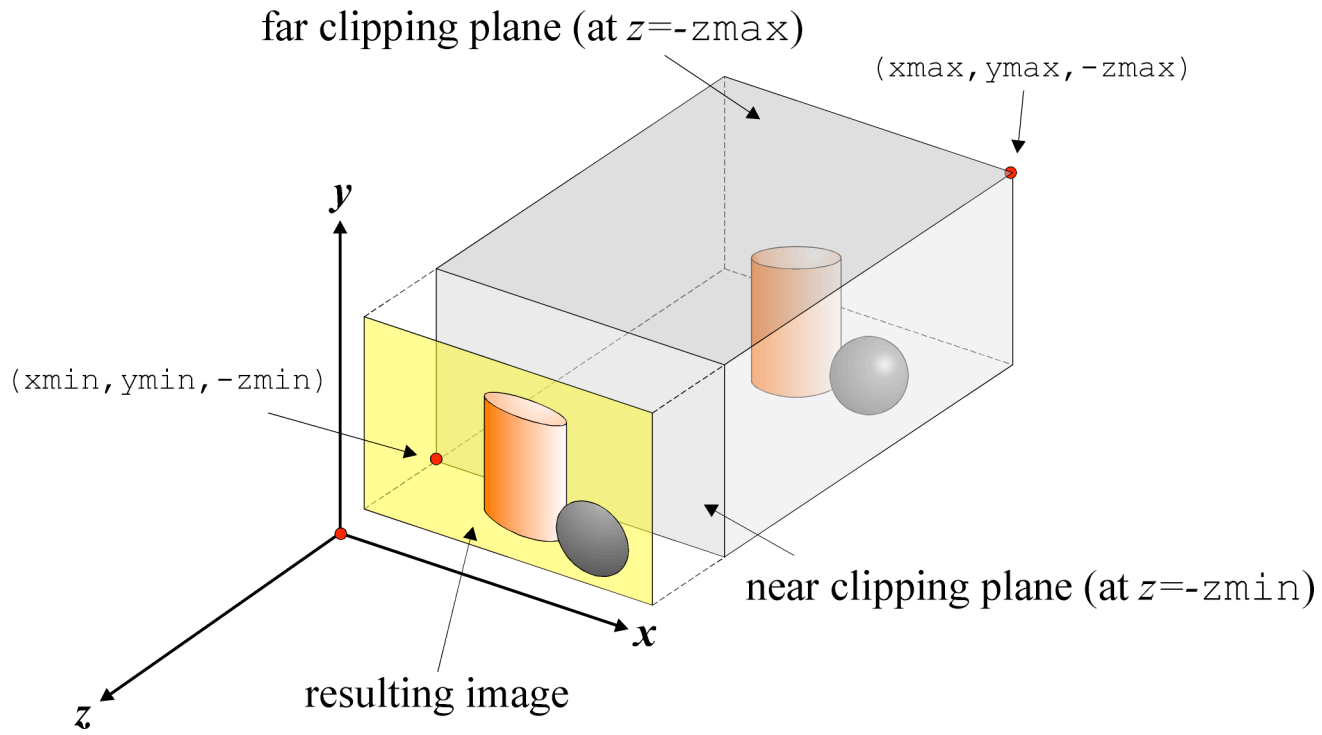
Projecções paralelas axonométricas: isométrica, dimétrica e cavaleira

- Se o objecto está alinhado com os eixos, o resultado é uma projecção ***ortogonal***;
- Caso contrário, é uma projecção ***axonométrica***.
- Se o plano de projecção intersecta os eixos XYZ à mesma distância relativamente à origem, o resultado é uma projecção ***isométrica***.



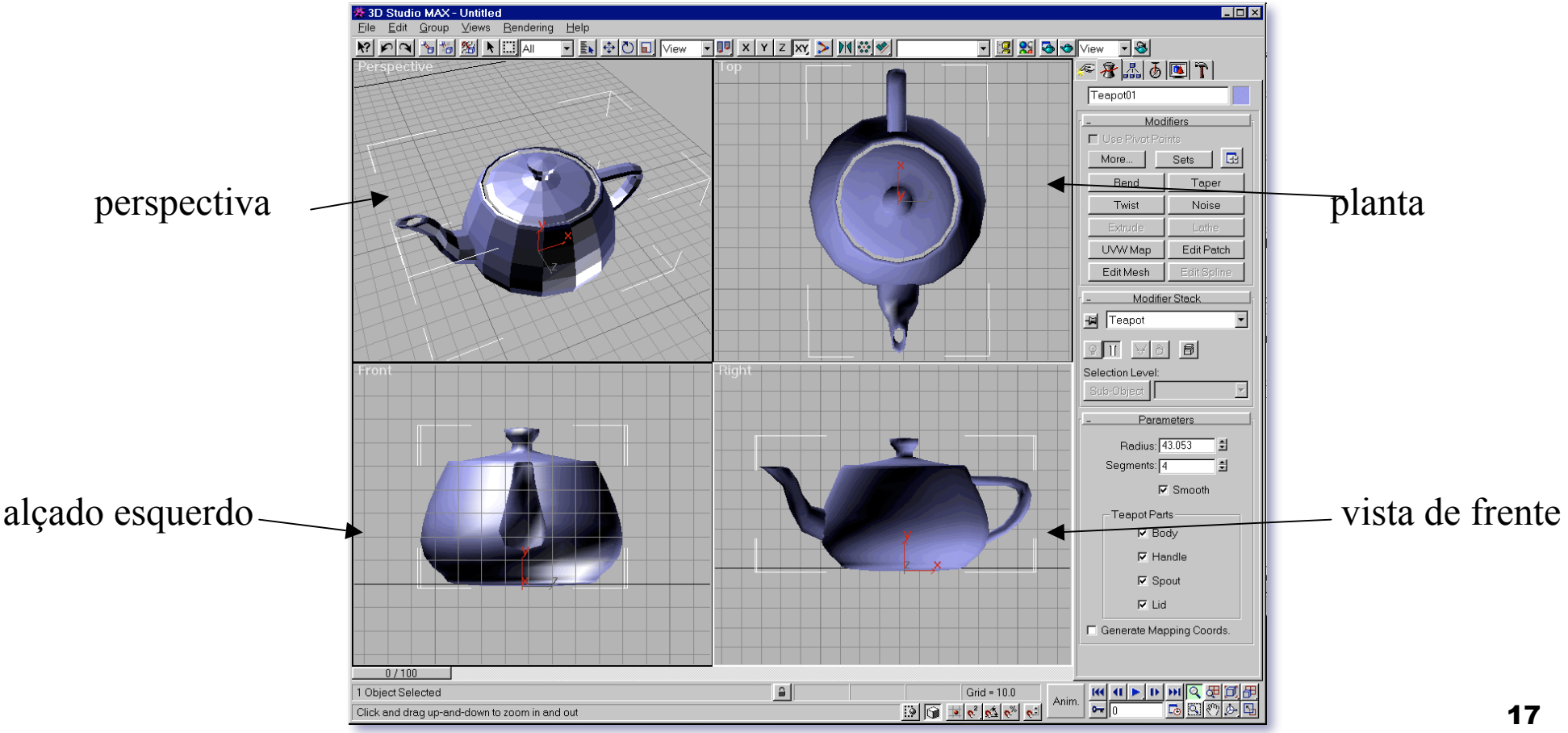
Projecções paralelas em OpenGL®

```
glOrtho(xmin, xmax, ymin, ymax, zmin, zmax);
```



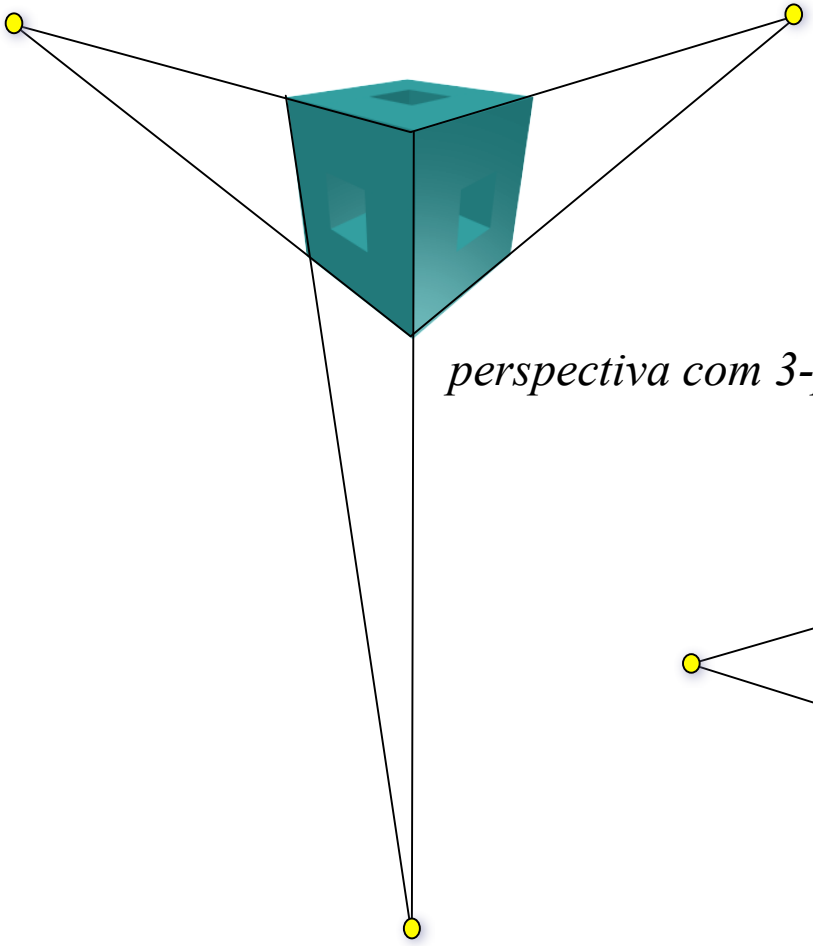
Projecções múltiplas

- Às vezes é útil ter *várias projecções* disponíveis para visualização
  - normalmente: *vista de frente*, *planta* e *alçado esquerdo*

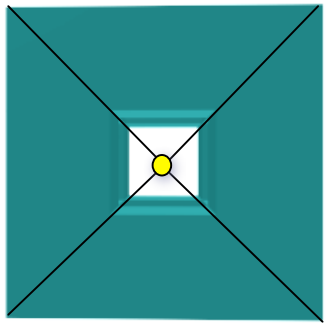


Projecções em perspectiva

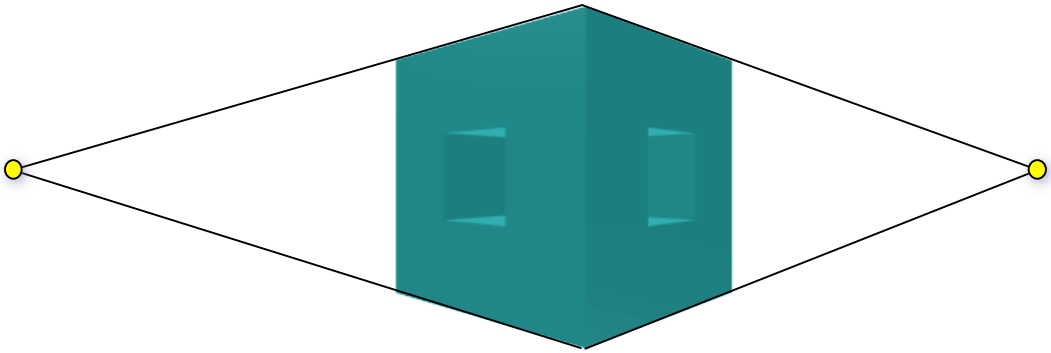
- projectantes não são paralelas
- observador a distância finita infinito



perspectiva com 3-pontos



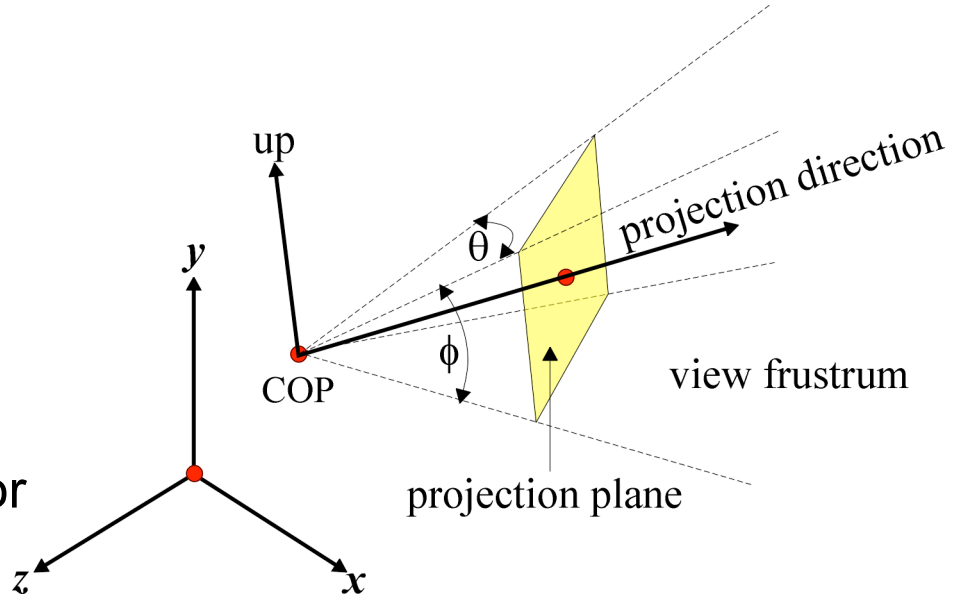
perspectiva com 1-ponto



perspectiva com 2-pontos

Projecções em perspectiva

- Projecções em perspectiva são mais complexas e exibem concorrência das projectantes ou raios visuais (as linhas paralelas parecem convergir para um ponto localizado a uma distância finita).
- Parâmetros:
  - centro de projecção (COP)
  - campo de vista (θ,ϕ)
  - direcção de projecção
  - direcção *up* do eixo da câmara ou do observador



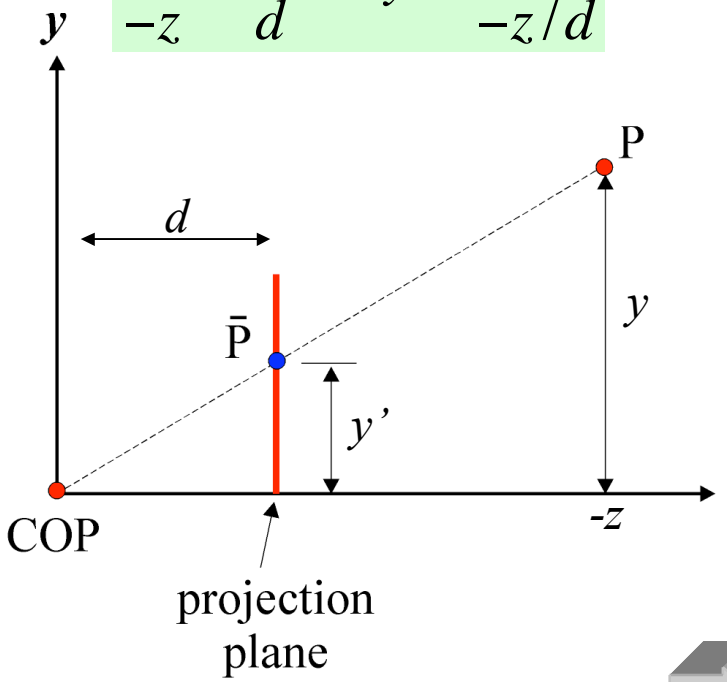
Projecções em perspectiva

Considere uma projecção em perspectiva com o ponto de vista na origem e a direcção de observação orientada ao longo do eixo -***z***, com o plano de projecção localizado em *z* = *-d*

$$\frac{x}{-z} = \frac{x'}{d} \Rightarrow x' = \frac{x}{-z/d}$$

$$\frac{y}{-z} = \frac{y'}{d} \Rightarrow y' = \frac{y}{-z/d}$$

$$\begin{cases} x' = \dfrac{-xd}{z} = \dfrac{x}{-z/d} \\ y' = \dfrac{y}{-z/d} \\ z' = -d \end{cases}$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \dfrac{x}{-z/d} \\ \dfrac{y}{-z/d} \\ -d \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x \\ y \\ z \\ -z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

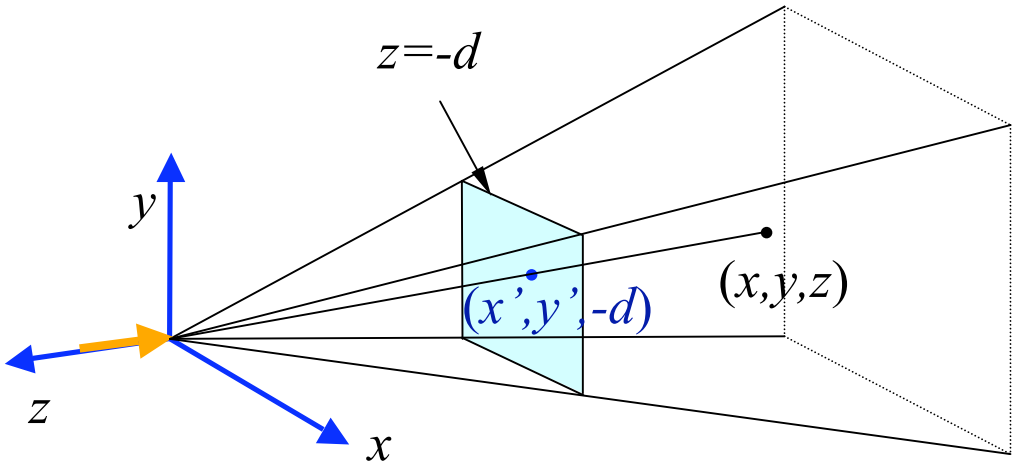
PROJECTION matrix

perspective division

Projecções em perspectiva: alternativa

Objectos dentro do ***frustum*** são projectados no plano paralelo ao plano x-y segundo as seguintes igualdades:

$$\frac{x'}{x} = \frac{y'}{y} = \frac{z'}{z} = \frac{-d}{z}$$



$$\begin{cases} x' = \dfrac{-xd}{z} = \dfrac{x}{-z/d} \\ y' = \dfrac{y}{-z/d} \\ z' = -d \end{cases}$$

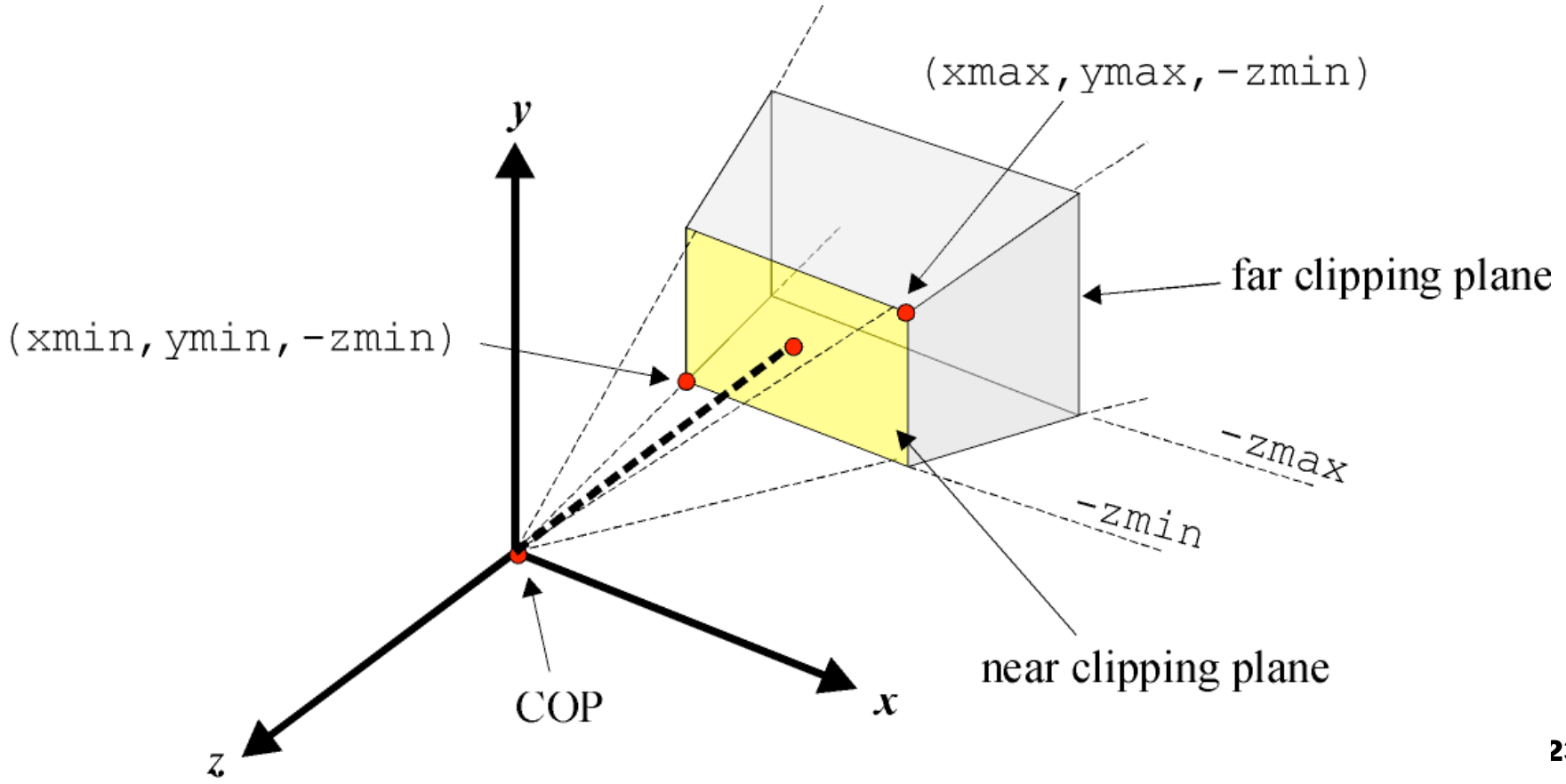
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Projecção em perspectiva

- Dependendo da aplicação, pode usar-se mecanismos diferentes para especificar uma vista em perspectiva. Exemplo: os ângulos do *campo de vista* podem ser inferidos se a distância ao plano de projecção é conhecida.
- Exemplo: a direcção de observação pode ser obtida se, além do COP, se especificar um ponto na cena para onde o observador olha.
- OpenGL suporta estes dois mecanismos de especificar uma vista em perspectiva através de:
  - **`glFrustrum`** ou
  - **`gluPerspective`**
  - A função **`gluLookAt`** permite alterar a posição do observador, a qual é, por omissão, a origem.

Projecções em perspectiva

```
glFrustrum(xmin, xmax, ymin, ymax, zmin, zmax);
```



frustum = pirâmide truncada do campo de vista

glFrustrum

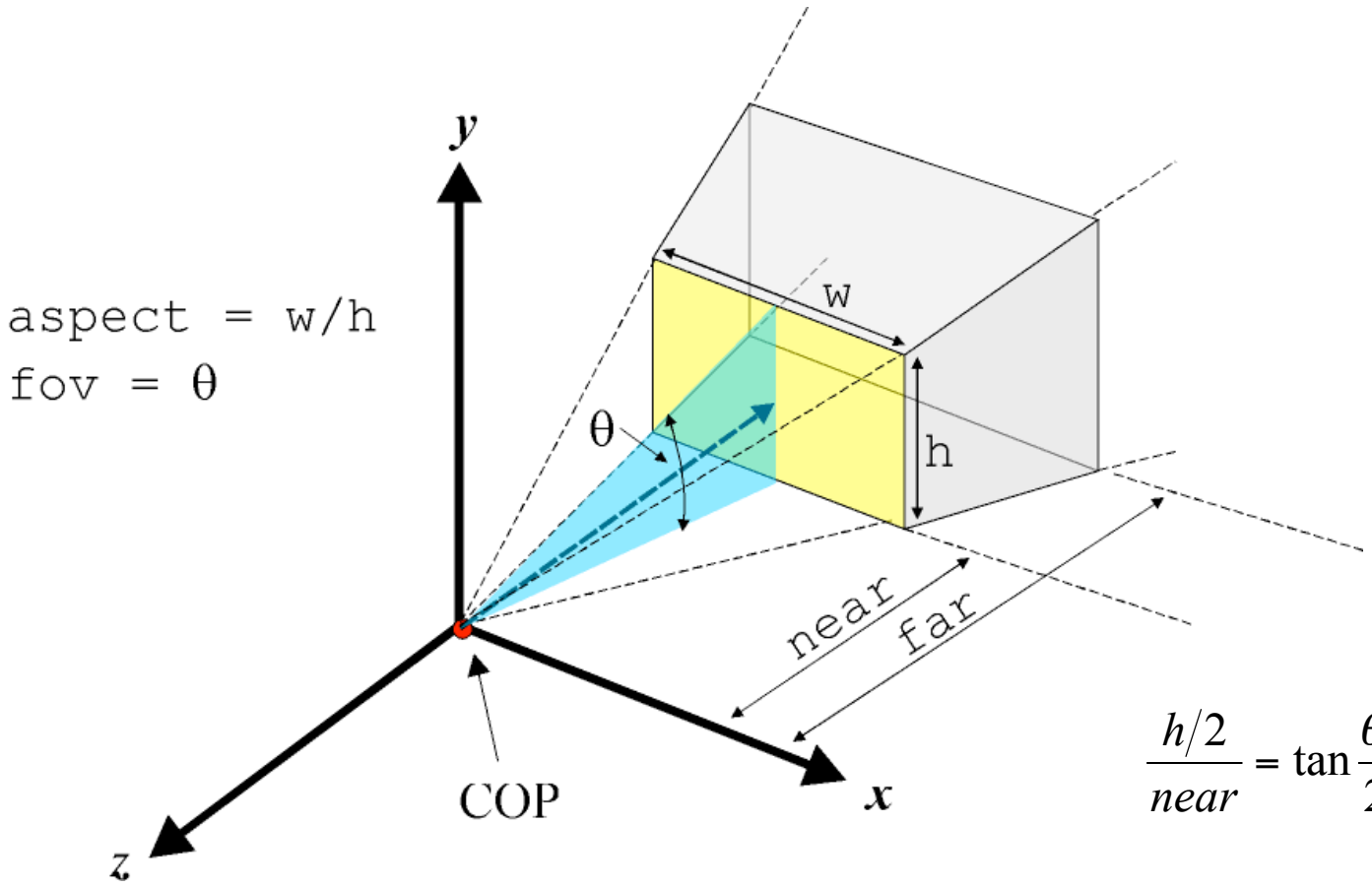
- Note-se que todos os pontos na linha definida pelo `COP` e (`xmin`,`ymin`,`-zmin`) são mapeados para o canto *inferior esquerdo* da janela.
- Também todos os pontos na linha definida pelo `COP` e (`xmax`,`ymax`,`-zmin`) são mapeados para o canto superior direito da janela.
- A direcção de observação é sempre paralela a ***-z***
- Não é necessário ter um *frustrum simétrico* como:

```
glFrustrum(-1.0, 1.0, -1.0, 1.0, 5.0, 50.0);
```

- Um *frustrum* não-simétrico introduz *obliquidade* na projecção.
- `zmin` e `zmax` são especificados como distâncias positivas ao longo de ***-z***

Projecções em perspectiva

```
gluPerspective(fov, aspect, near, far);
```

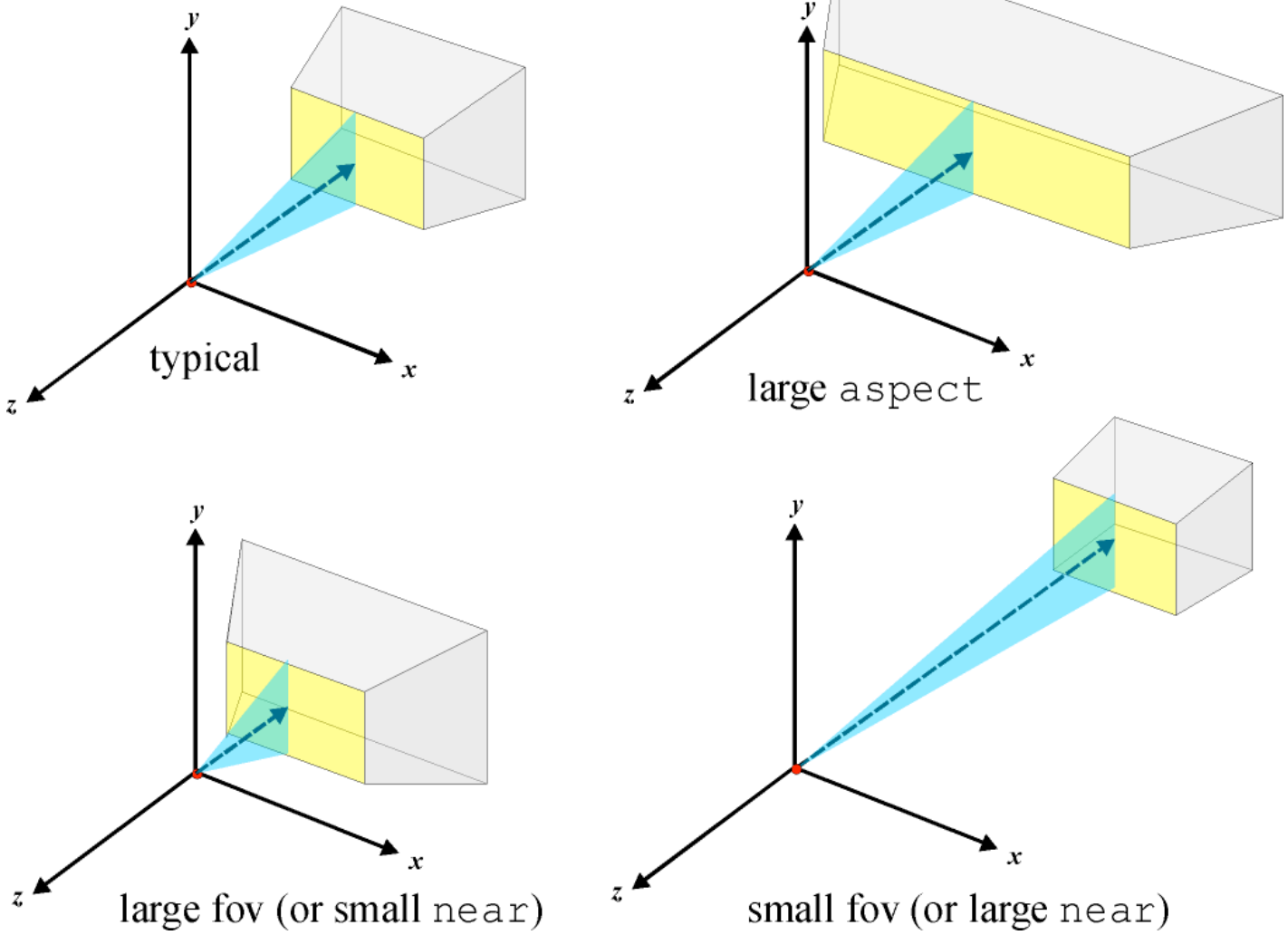


$$\frac{h/2}{near} = \tan\frac{\theta}{2} \Rightarrow h = 2\,near\tan\frac{\theta}{2}$$

`gluPerspective`

- Uma função para simplificar a especificação de vistas ou projecções em perspectiva.
- Só permite a criação de *frustra simétricos*.
- O ponto de vista (do observador) está na origem e a direcção de observação é o eixo **-z**.
- O ângulo do *campo de vista*, `fov`, tem de pertencer ao intervalo [0,180].
- `aspect` permite a criação dum *frustrum* com a mesma *razão de aspecto* do visor (*viewport*) por forma a eliminar distorção.

Projecções em perspectiva

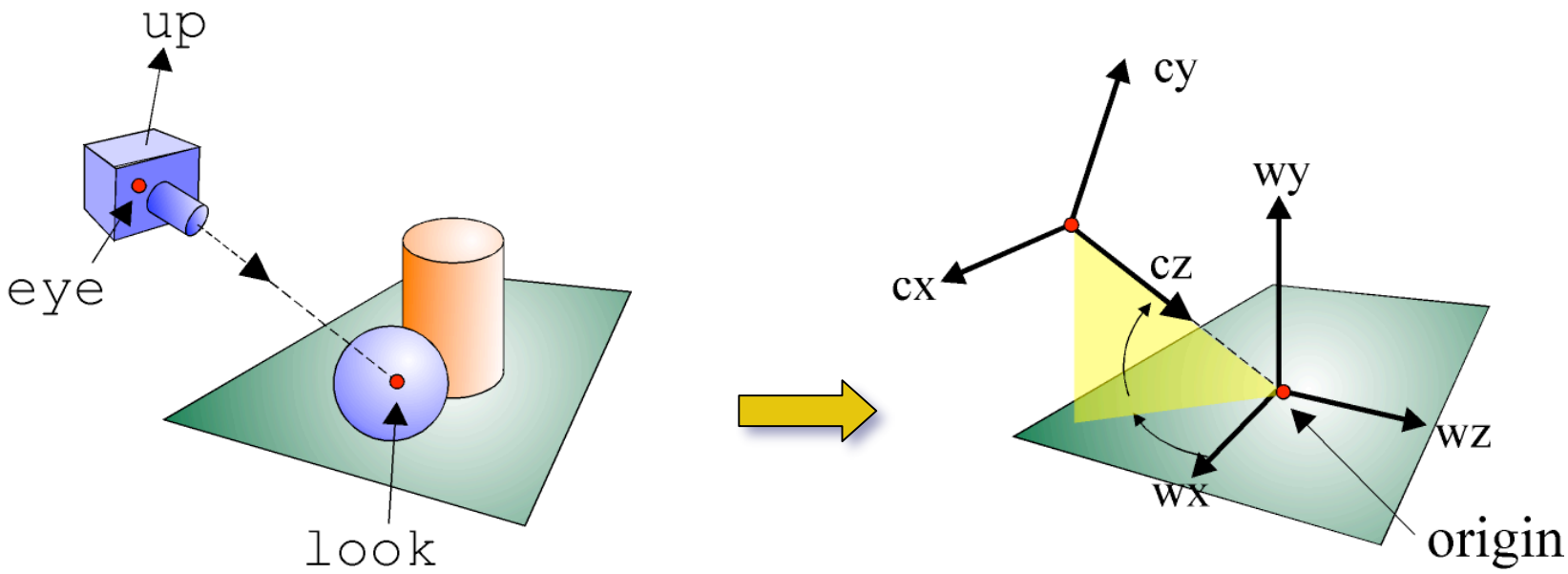


Posicionamento da câmara

- As projecções anteriores têm limitações:
  - COP fixo e direcção de projecção (ou observação) fixa
- Para obter uma posição e orientação arbitrárias da câmara temos de manipular a matriz `MODELVIEW` antes da criação dos modelos. Desta forma, posiciona-se a câmara relativamente aos objectos da cena.
- Por exemplo, há duas possibilidades para posicionar a câmara em (10, 2, 10) em relação ao referencial do domínio da cena:
  - mudar o referencial do domínio da cena antes de criar os objectos usando `translatef` e `rotatef`: **`glTranslatef(-10,-2,-10);`**
  - usar `gluLookAt` para posicionar a câmara relativamente ao referencial do domínio da cena: **`gluLookAt(10, 2, 10, … );`**
- Estas duas possibilidades são *equivalentes*.

Posicionamento da câmara

```
gluLookAt(eyex, eyey, eyez, lookx, looky, lookz, upx, upy, upz);
```

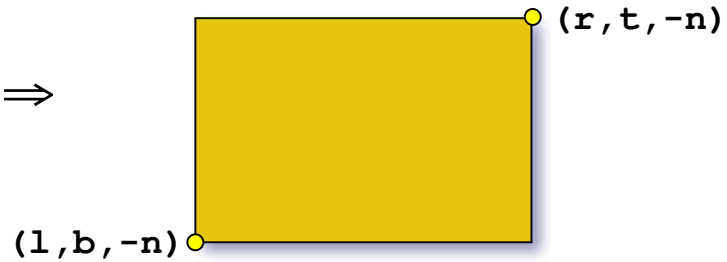


equivalente a:

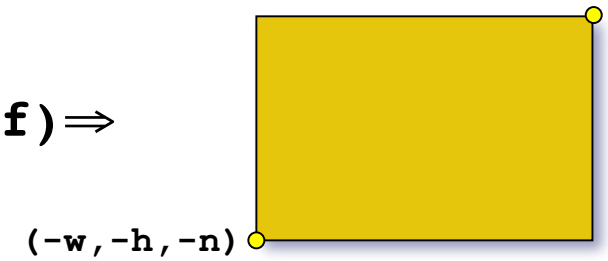
```
glTranslatef(-eyex, -eyey, -eyez);
glRotatef(theta, 1.0, 0.0, 0.0);
glRotatef(phi, 0.0, 1.0, 0.0);
```

Janela de projecção

- A matriz de projecção define a transformação de coordenadas 3D do domínio da cena numa janela 2D que pertence ao plano de projecção.
- As dimensões da janela de projecção são definidas como parâmetros da projecção:
  - **`glFrustrum(l,r,b,t,n,f)`**⇒



  - **`gluPerspective(f,a,n,f)`**⇒

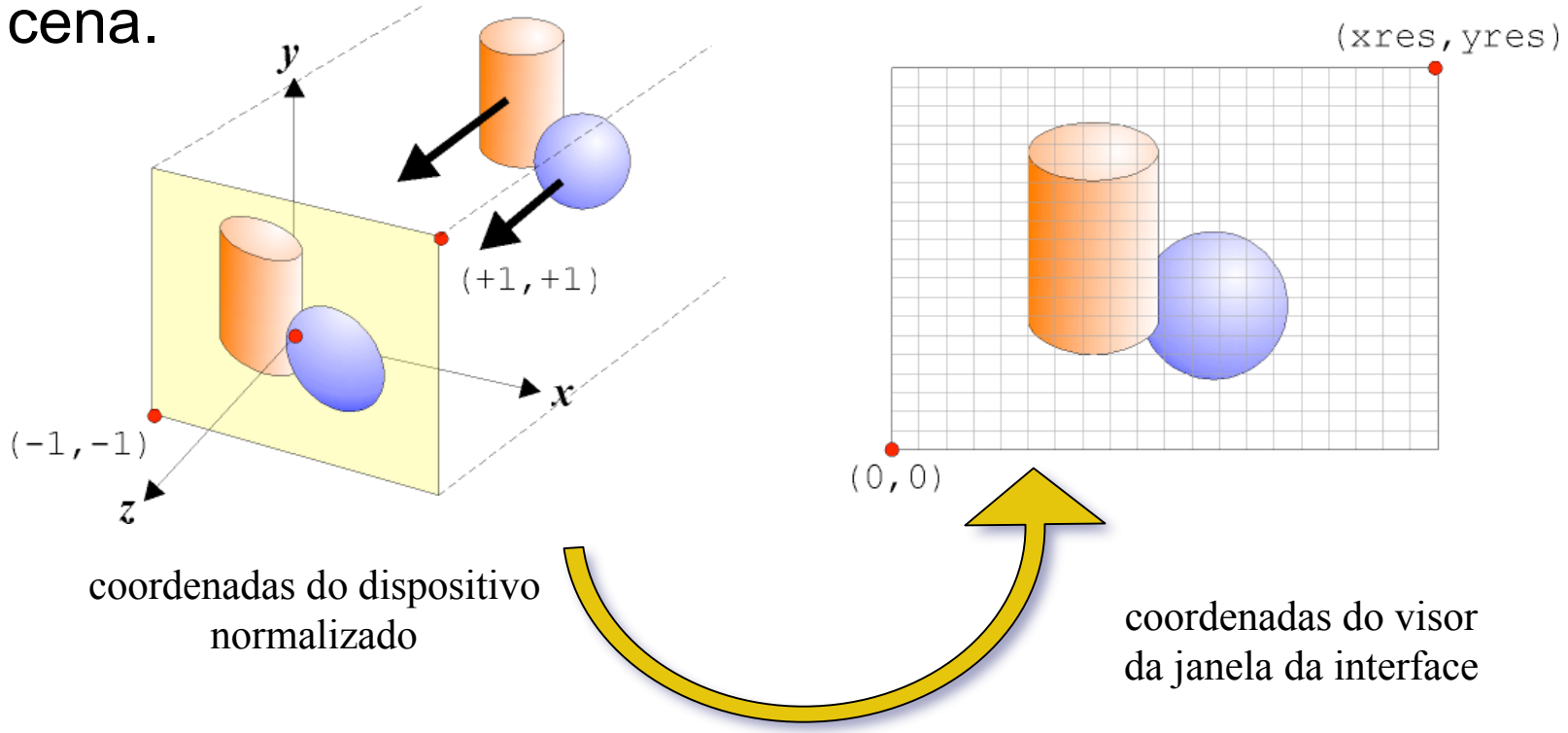


$$\mathbf{h} = \mathbf{n} \cdot \tan\frac{\mathbf{f}}{2}$$

$$\mathbf{w} = \mathbf{h} \cdot \mathbf{a}$$

Transformação janela-visor: revisão

- Como vimos no capítulo anterior, é preciso mapear os pontos do sistema de coordenadas da janela de projecção para os pixéis do sistema de coordenadas do visor da janela da interface, por forma a determinar o pixel associado a cada vértice dos objectos da cena.



Transformação janela-visor: revisão

- Uma transformação *afim* planar é usada.
- Após projecção no plano de vista, todos ospontos são transformados em coordenadas do *dispositivo normalizado*: `[-1,-1]×[+1,+1]`.

$$x_n = 2\left(\frac{x_p - x_{\min}}{x_{\max} - x_{\min}}\right) - 1$$

$$y_n = 2\left(\frac{y_p - y_{\min}}{y_{\max} - y_{\min}}\right) - 1$$

- `glViewport` é usado para relacionar os dois sistemas de coordenadas:

```
glViewport(int x, int y, int width, int height);
```

Transformação janela-visor: revisão

- (`x`,`y`) = posição do canto inferior esquerdo do visor dentro da janela da interface
- `width`,`height` = dimensões do visor em pixéis ⇒

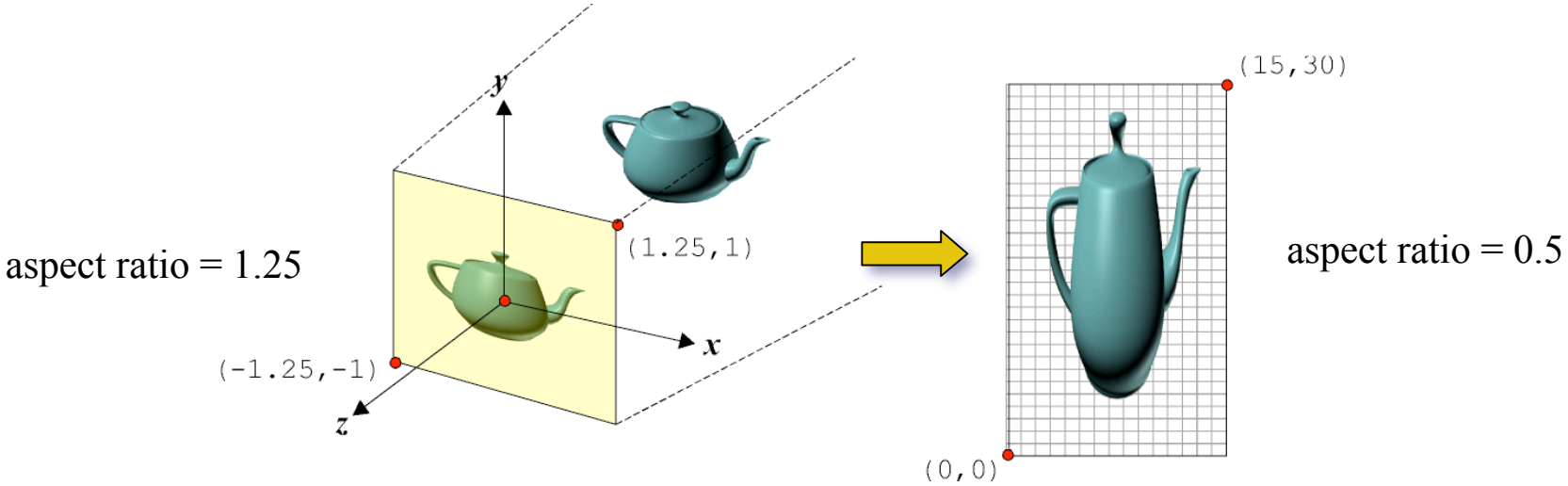
$$x_w = (x_n + 1)\left(\frac{\texttt{width}}{2}\right) + \texttt{x} \qquad y_w = (y_n + 1)\left(\frac{\texttt{height}}{2}\right) + \texttt{y}$$

- Normalmente, recria-se a janela após o evento *resize* da janela da interface assegurar o mapeamento correcto entre as dimensões do visor e da janela:

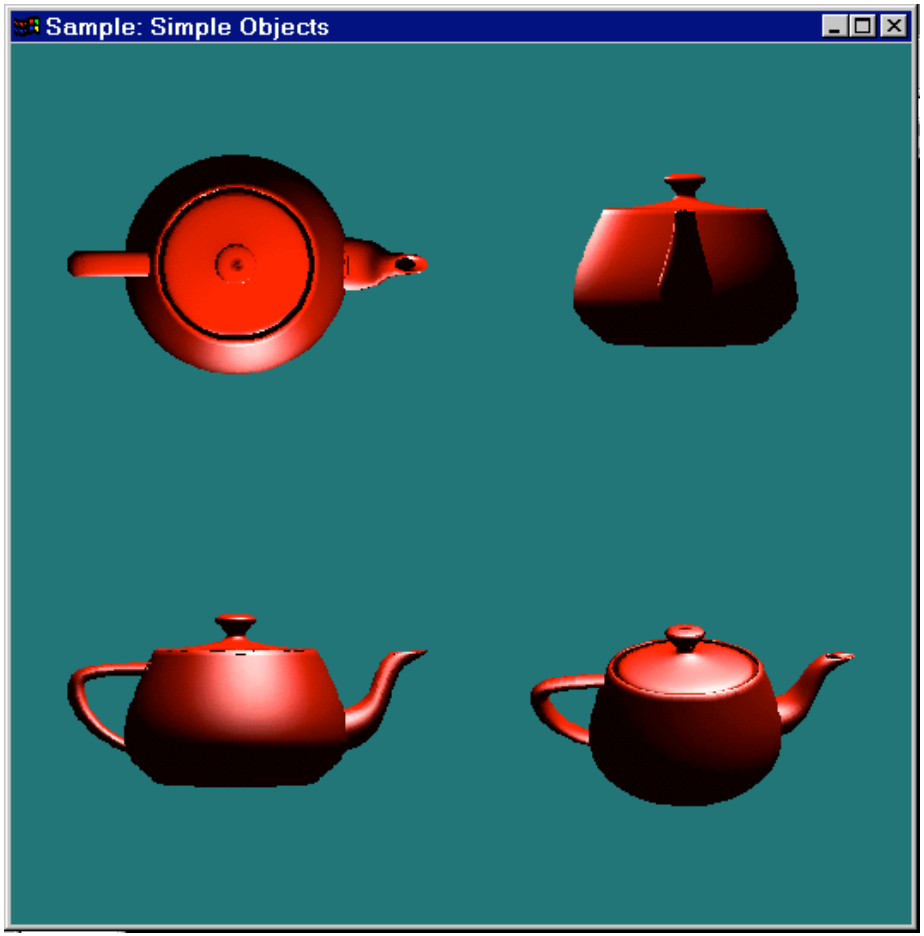
```
static void reshape(int width, int height)
{
        glViewport(0, 0, width, height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(85.0, 1.0, 5, 50);
}
```

Razão de aspecto: revisão

- A razão de aspect (*aspect ratio*) define a relação entre a largura (width) e a altura (height) da imagem.
- A razão de aspecto da janela de projecção é explicitamente fornecida através da função `gluPerspective`.
- A razão de aspecto do visor deve ser a mesma para evitar distorção de imagem:



Aplicação com 4 visores numa janela da interface




```
// top left: top view
glViewport(0, win_height/2, win_width/2, win_height/2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 50.0);
gluLookAt(0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glCallList(object);

// top right: right view
glViewport(win_width/2, win_height/2, win_width/2, win_height/2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 50.0);
gluLookAt(5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glCallList(object);

// bottom left: front view
glViewport(0, 0, win_width/2, win_height/2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-3.0, 3.0, -3.0, 3.0, 1.0, 50.0);
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glCallList(object);

// bottom right: rotating perspective view
glViewport(win_width/2, 0, win_width/2, win_height/2);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(70.0, 1.0, 1, 50);
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(30.0, 1.0, 0.0, 0.0);
glRotatef(Angle, 0.0, 1.0, 0.0);
glCallList(object);
```