MAC420/5744: Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #11: Normalização de projeções

Some slides by Steve Marschner, Cornell CS4620/5620

Visualização

- Abordagem inversa
  - Comeşe com pixels da imagem
  - Construa explicitamente raios para cada pixel
  - Interrogue as partes da cena que projetam em cada pixel
- Abordagem direta
  - Comeşe com pontos em 3D
  - Calcule suas projeções
  - Rasterize seus pixels

Projeções no OpenGL

- No início:
  - Pipeline fixo
  - Transformações Model-View e Projection
    - glMatrixMode(gl_MOVELVIEW);
    - glMatrixMode(gl_PROJECTION);
  - Referências bem definidas: objeto, modelo, câmera, recorte, janela
- Após versão 3.0 (ES 2.0)
  - Pipeline programável
  - Sem transformações pré-definidas
- MV.js reintroduz algumas das funções existentes

Projeções em CG

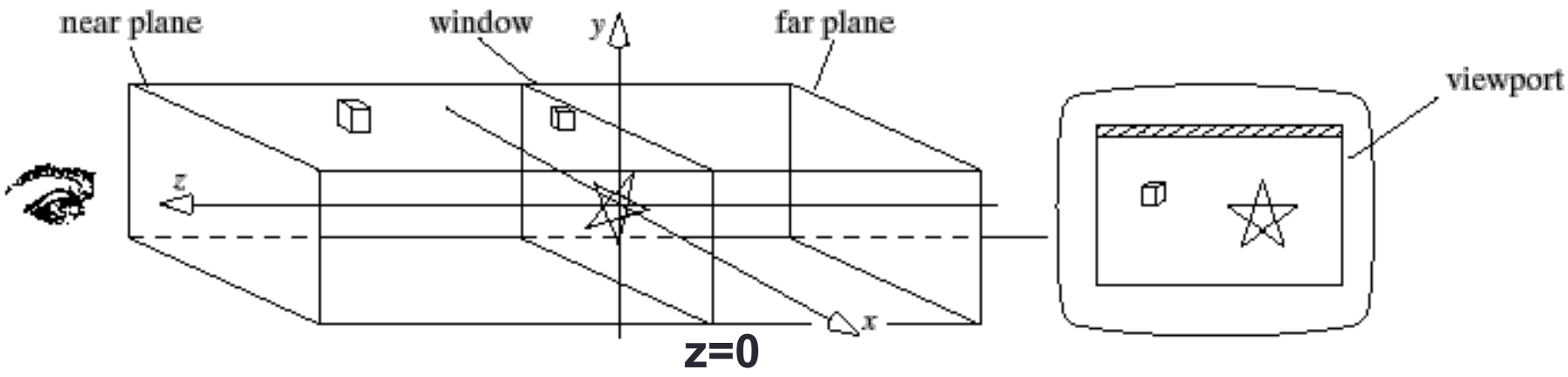
- Três aspectos que temos que levar em consideração, todos eles implementados no pipeline
  - Posicionamento da câmera
    - Manipulação da matriz de model-view
  - Escolha da lente
    - Manipulação da matriz de projeção
  - Recorte (*clipping*)
    - Seleção do volume de visualização

A câmera do WebOpenGL

- Inicialmente o sistema de referência do objeto e câmera são os mesmos
  - Matriz model-view *default* é a identidade
- A câmera está localizada na origem e aponta na direção negativa de *z*
- O Web/OpenGL também especifica um volume de visualização que é um cubo de tamanho 2, centralizado na origem
  - A matriz *default* de projeção é a identidade

Projeção default

A projeção *default* é a ortogonal

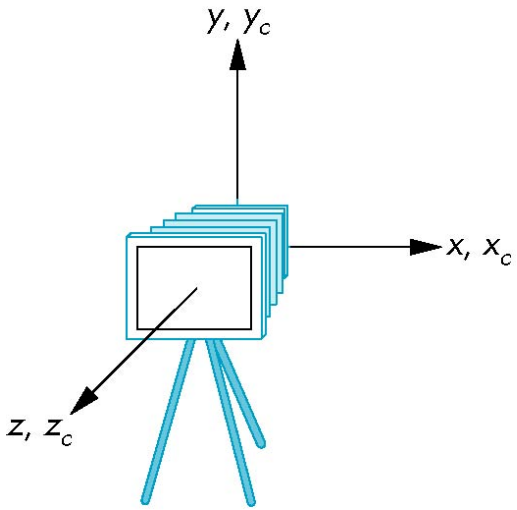


Movendo a câmera

- Para visualizar objetos com valores positivos e negativos de z
  - Movemos a câmera para o lado positivo de z
    - Translação do referencial da câmera
  - Movemos os objetos para o lado negativo de z
    - Translação do referencial do mundo
- Estas duas abordagens são equivalentes e determinadas pela matriz model-view
  - Desejamos uma translação
    - `translate(0.0,0.0,-d), d > 0`

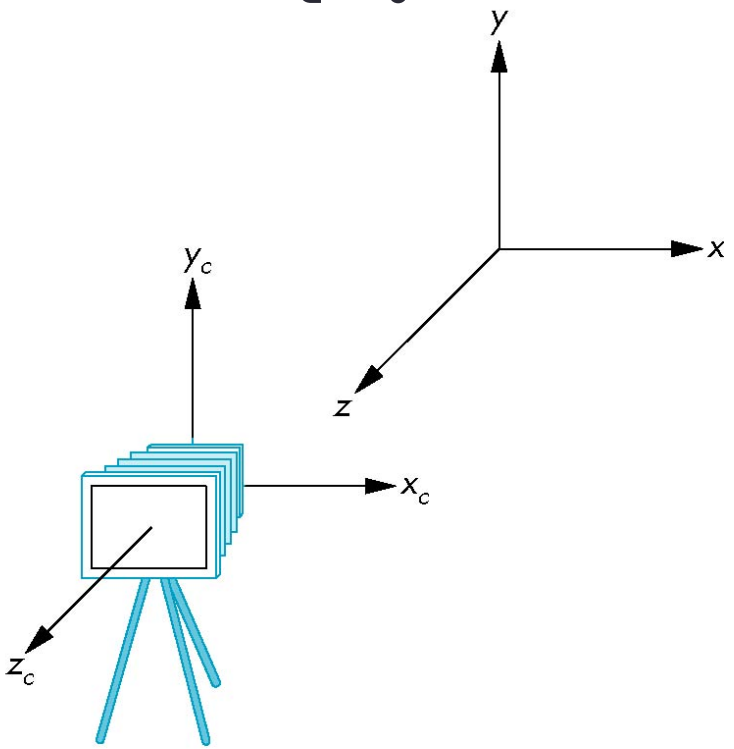
Movendo a câmera para trás da origem

referencial default



(a)

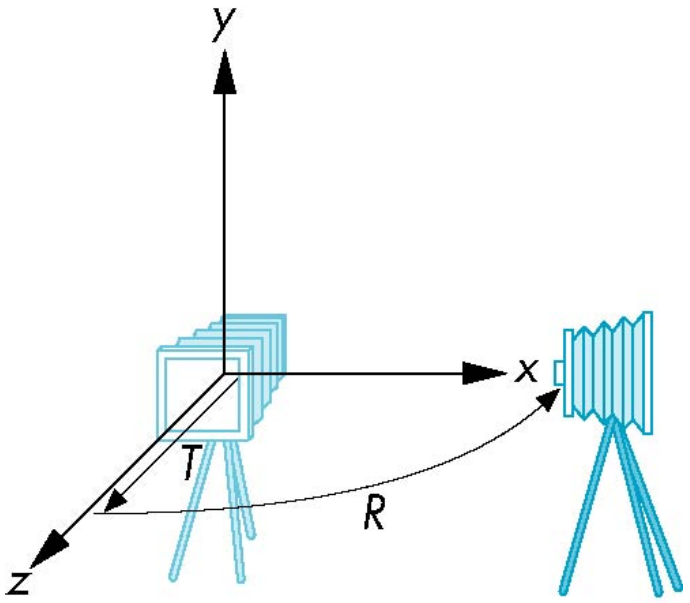
referencial após translação por –d
d > 0



(b)

Movendo a câmera

- Podemos mover a câmera para qualquer posição utilizando uma sequência de rotações e translações
- Exemplo: visão lateral
  - Rotacione a câmera
  - Distancie-se da origem
  - Matriz model-view C=TR



Código WebGL

- Lembre-se que a última transformação especificada é a primeira ser feita

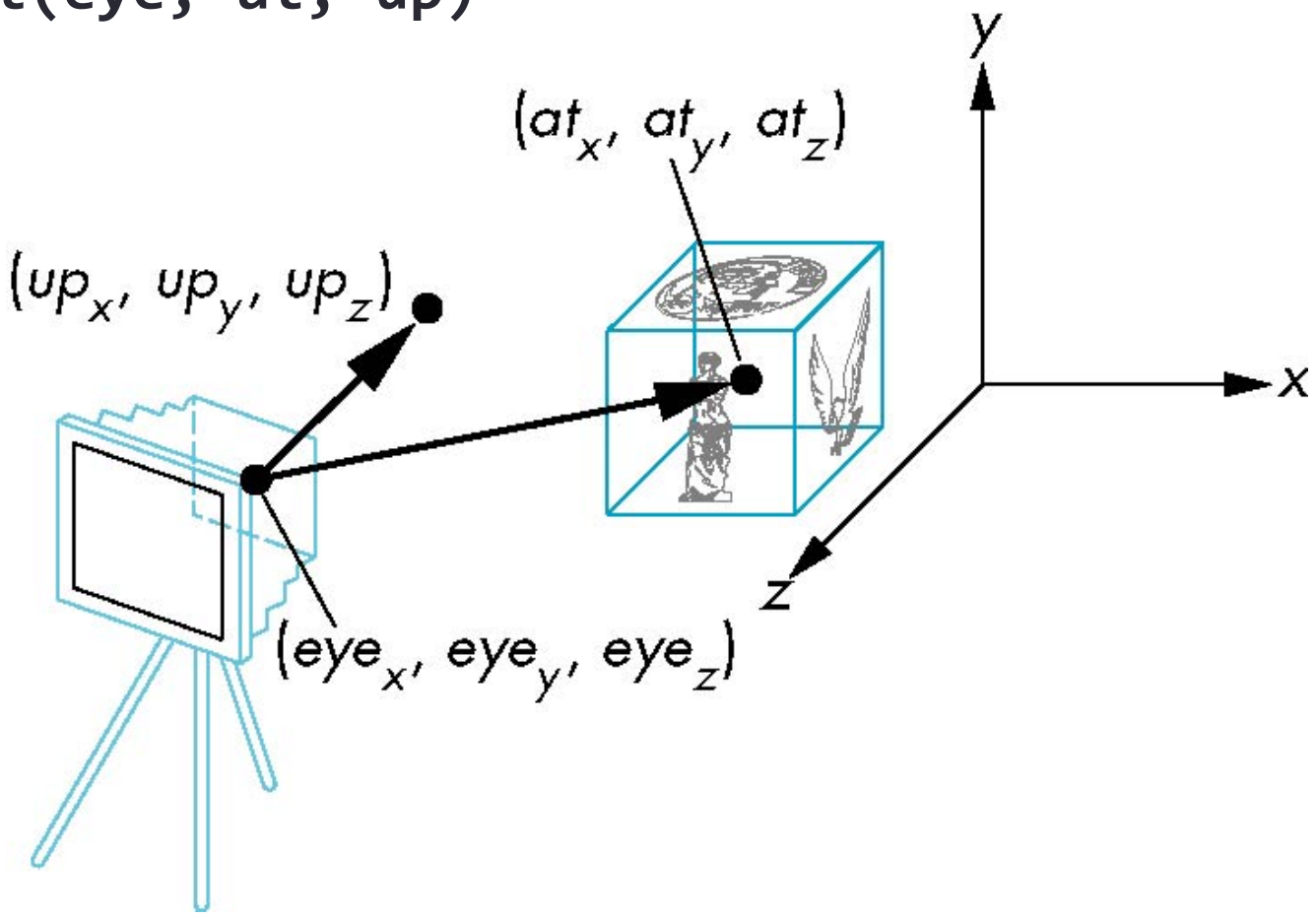
```
// Using MV.js
var t = translate (0.0, 0.0, -d);
var ry = rotateY(90.0);
var m = mult(t, ry);

or

var m = mult(
       translate (0.0, 0.0, -d),
       rotateY(90.0)
);
```

A função lookAt()

```
lookAt(eye, at, up)
```



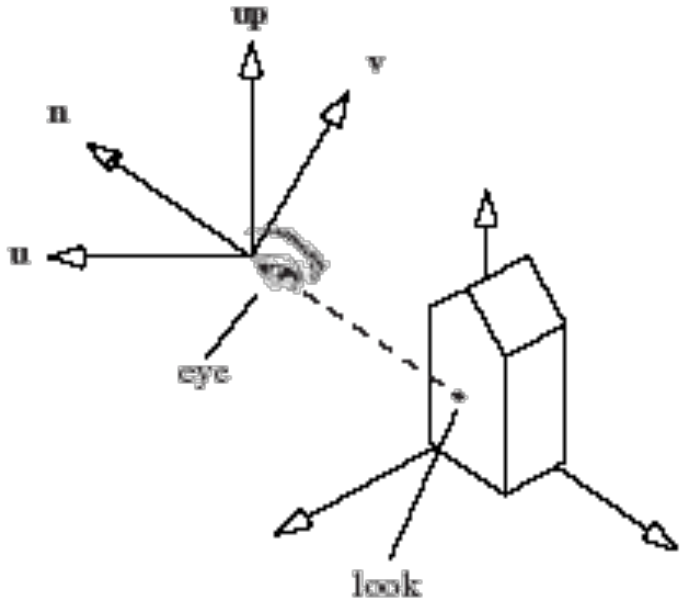
A função lookAt()

- A biblioteca GLU possuia a função gluLookAt() para construir a matriz model-view usando uma interface simples
- Note que precisamos passar um vetor que aponte para indicar o "céu" (cima da câmera)
- Substituída por lookAt() in MV.js
  - Podemos concatenar com as transformações do modelo
- Exemplo: visão isométrica de um cubo alinhado com os eixos

```
var eye = vec3(1.0, 1.0, 1.0);
var at = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);
var mv = lookAt(eye, at, up);
```

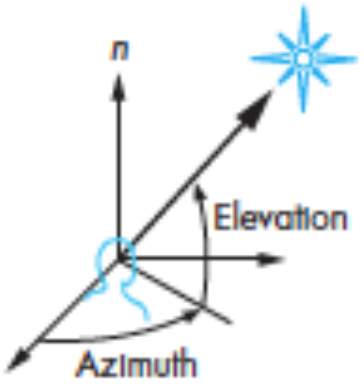
A função lookAt()

- Esta função cria um sistema de coordenadas da câmera que consiste em calcular os vetores ortogonais: **u**, **v**, e **n**
  - **n** = eye – at
  - **u** = **up** x **n**;
  - **v** = **n** x **u**
- Normalizar **n**, **u**, **v**

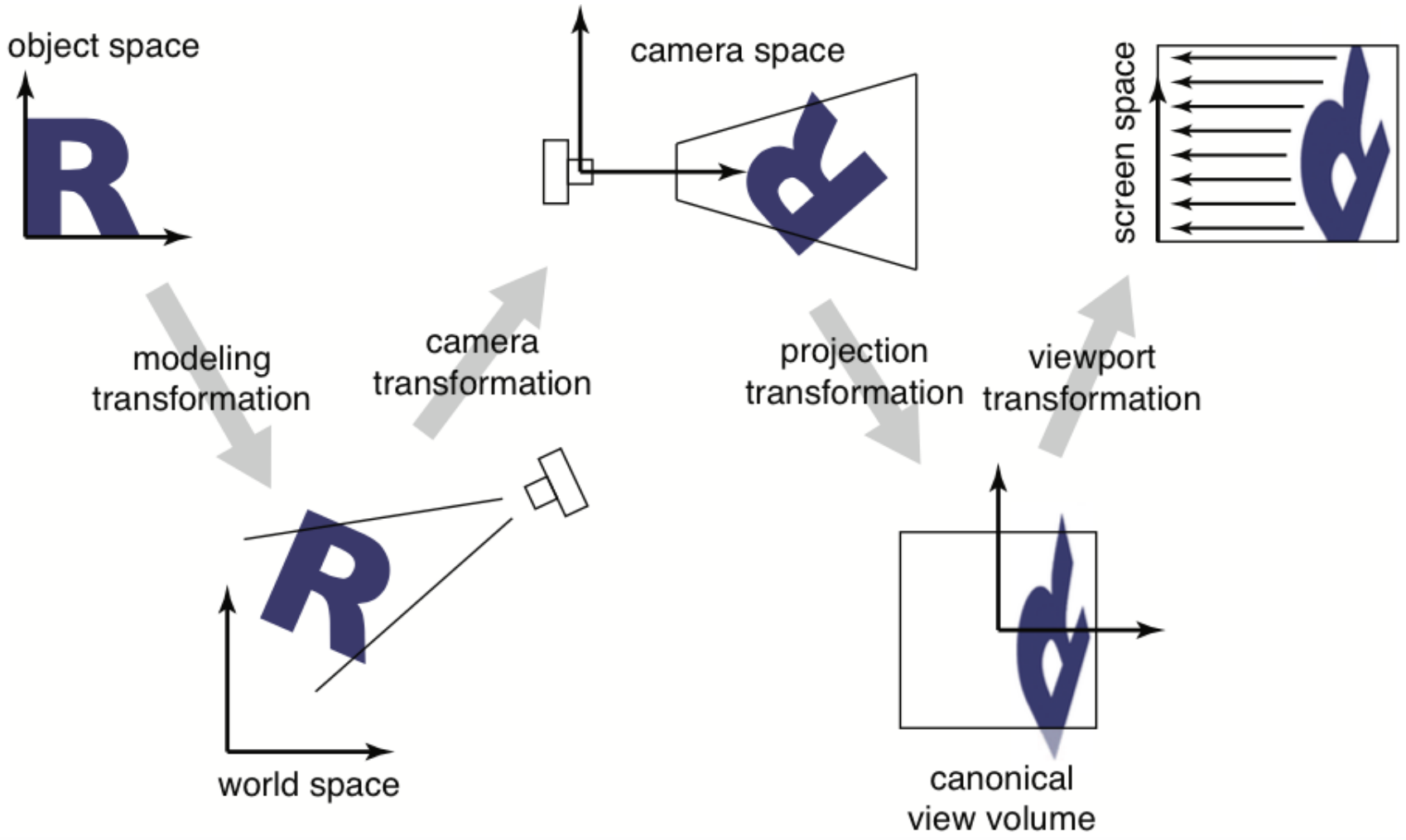


Outras APIs

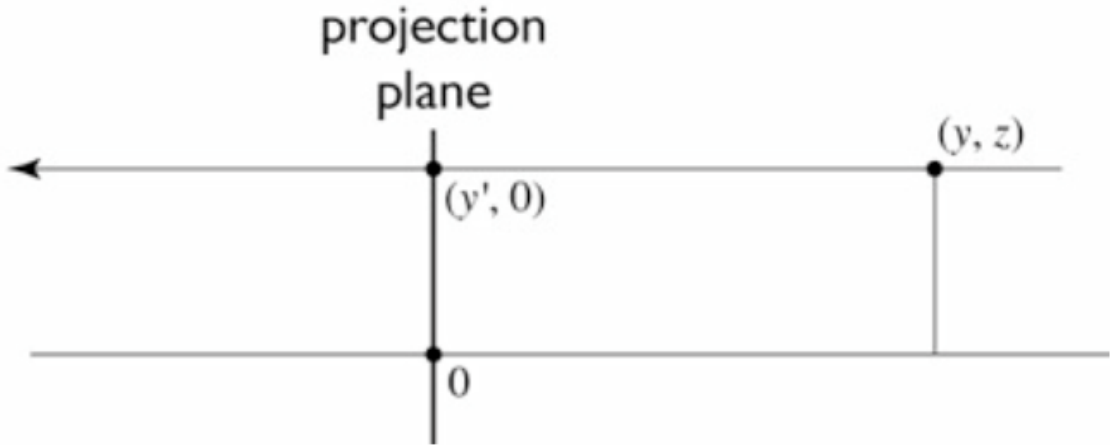
- A função lookAt() é nossa única função para posicionar a câmera
- Outras APIs:
  - Yaw, pitch, roll
  - Elevação, azimute, *ângulo de torsão*
  - Ângulos de Euler, etc



Pipeline de transformações



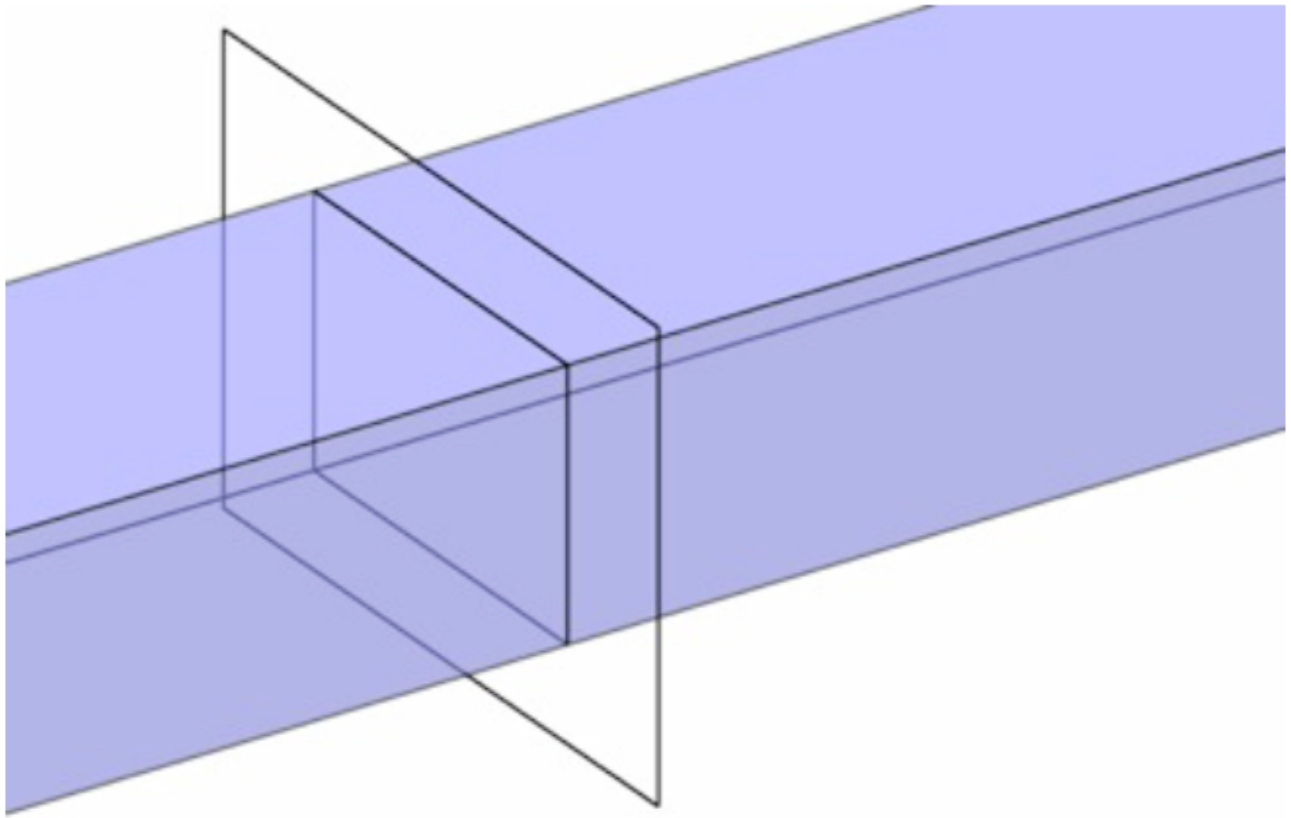
Projeção paralela: ortográfica



to implement orthographic, just toss out z:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Volume de visualização



Profundidade infinita

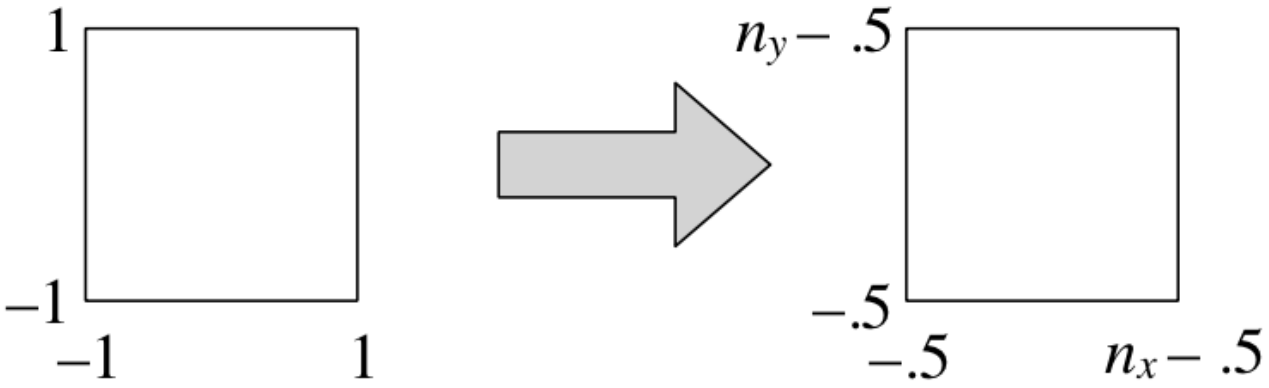
Volume canônico

- Start by looking at a restricted case: the *canonical view volume*
- It is the cube $[-1,1]^3$, viewed from the z direction
- Matrix to project it into a square image in $[-1,1]^2$ is trivial:

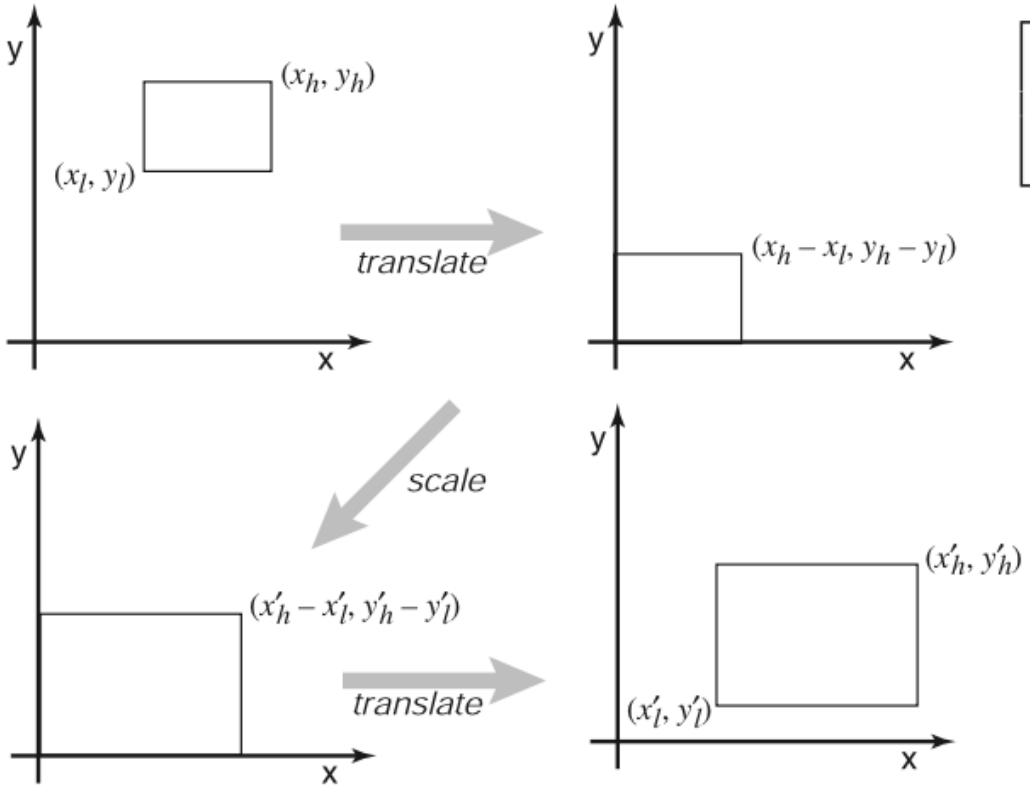
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Volume canônico em pixels

- To draw in image, need coordinates in pixel units, though
- Exactly the opposite of mapping (i,j) to (u,v) in ray generation



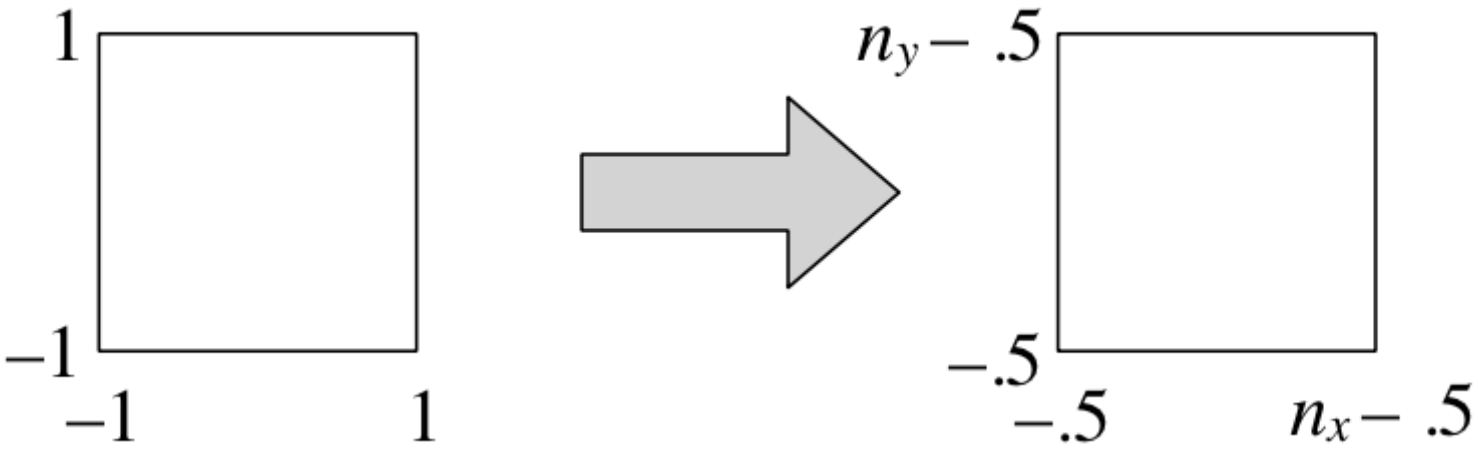
Primeira generalização



$$\begin{bmatrix} 1 & 0 & x'_l \\ 0 & 1 & y'_l \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformação para viewport



$$\begin{bmatrix} x_{\text{screen}} \\ y_{\text{screen}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{canonical}} \\ y_{\text{canonical}} \\ 1 \end{bmatrix}$$

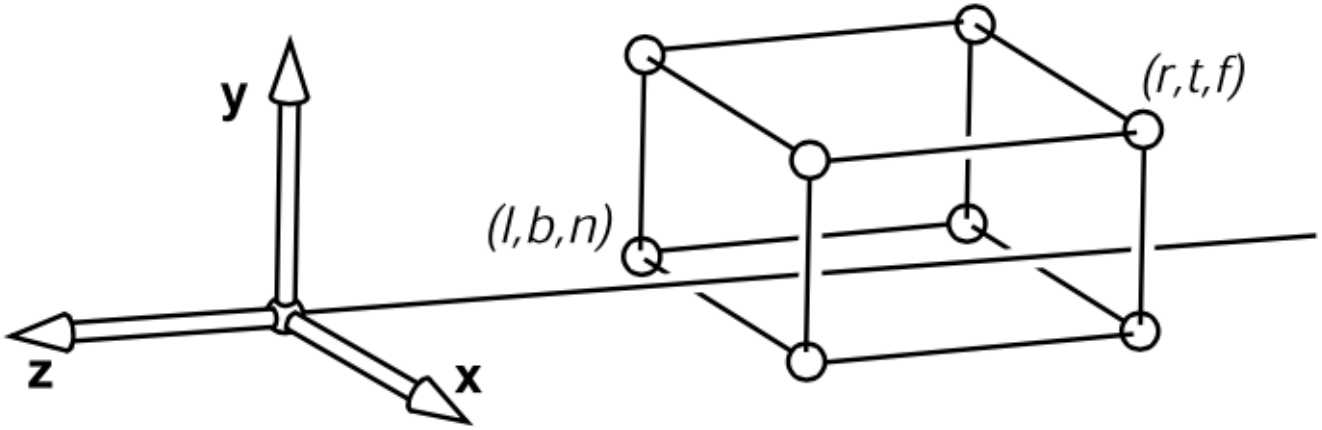
Transformação para viewport

- In 3D, carry along z for the ride
  - one extra row and column

$$\mathbf{M}_{\text{vp}} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Segunda generalização

- First generalization: different view rectangle
  - retain the minus-z view direction



  - specify view by left, right, top, bottom (as in RT)
  - also near, far

Recortes

In object-order systems we always use at least two *clipping planes* that further constrain the view volume

- near plane: parallel to view plane; things between it and the viewpoint will not be rendered
- far plane: also parallel; things behind it will not be rendered

These planes are:

- partly to remove unnecessary stuff (e.g. behind the camera)
- but really to constrain the range of depths

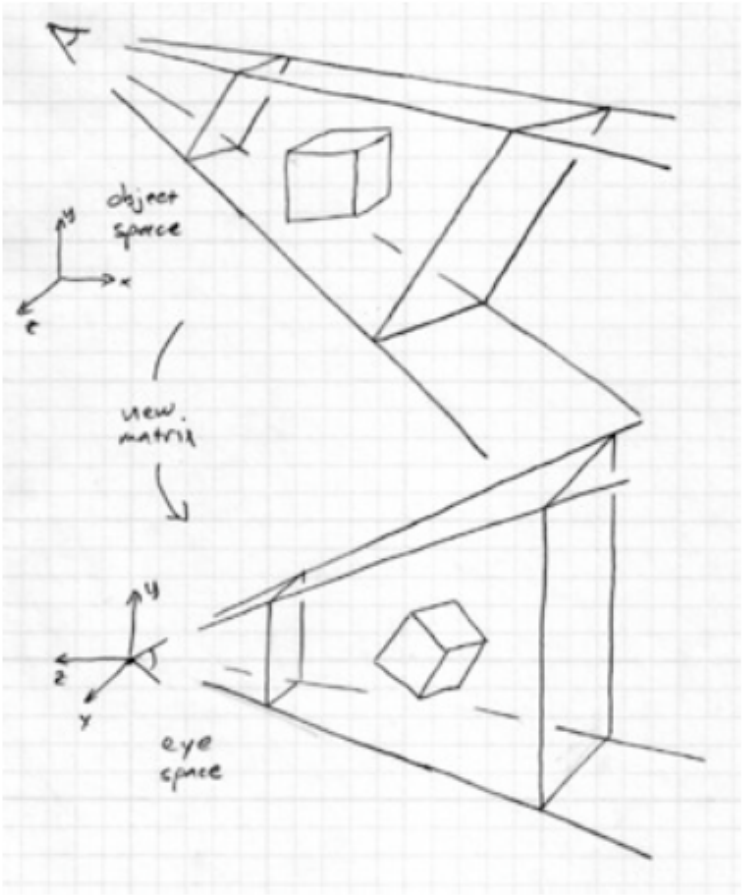
Projeção ortográfica

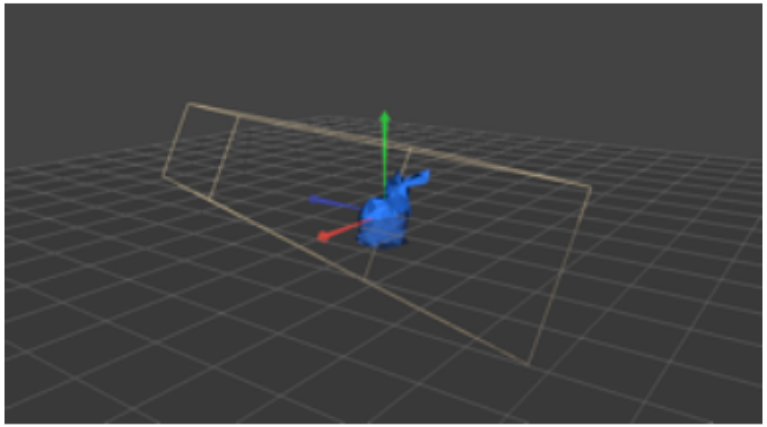
- We can implement this by mapping the view volume to the canonical view volume.
- This is just a 3D windowing transformation!

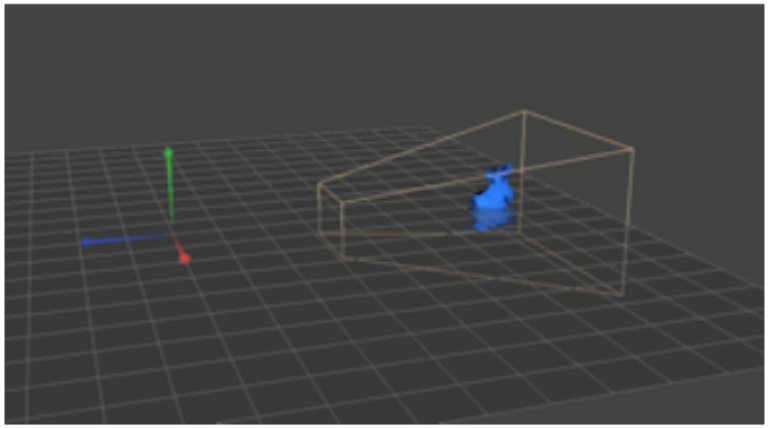
$$\begin{bmatrix} \frac{x'_h - x'_l}{x_h - x_l} & 0 & 0 & \frac{x'_l x_h - x'_h x_l}{x_h - x_l} \\ 0 & \frac{y'_h - y'_l}{y_h - y_l} & 0 & \frac{y'_l y_h - y'_h y_l}{y_h - y_l} \\ 0 & 0 & \frac{z'_h - z'_l}{z_h - z_l} & \frac{z'_l z_h - z'_h z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação da câmera

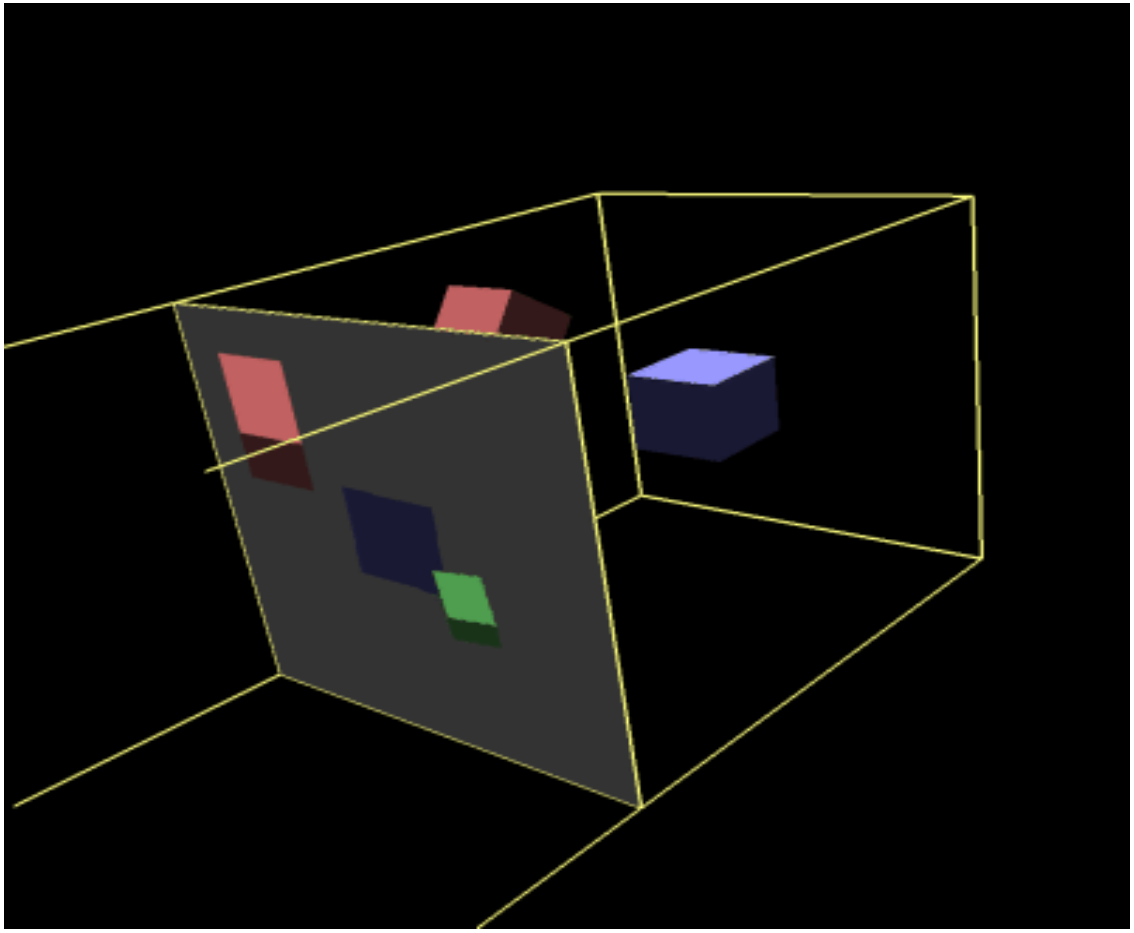






the camera matrix rewrites all coordinates in eye space

Visão ortográfica



Transformação ortográfica

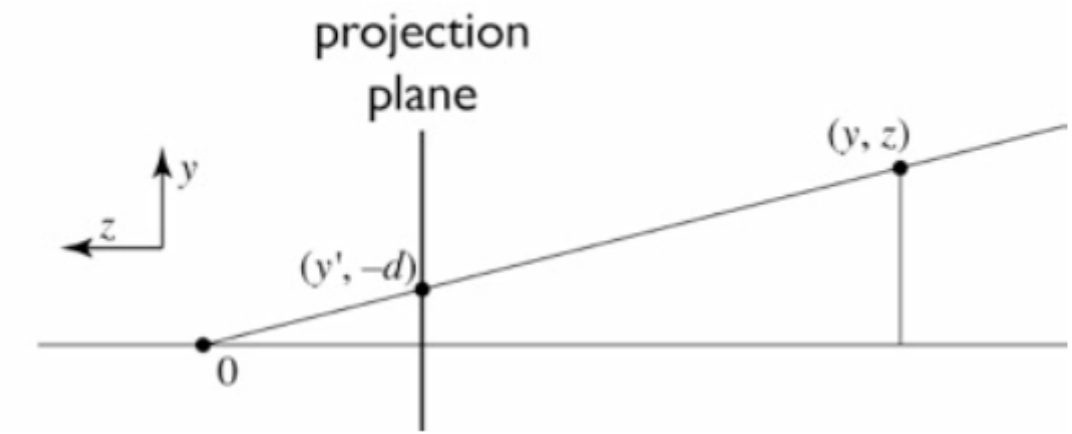
- Start with coordinates in object's local coordinates
- Transform into world coords (modeling transform, M_m)
- Transform into eye coords (camera xf., $M_{cam} = F_c^{-1}$)
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

$$\begin{bmatrix} x_s \\ y_s \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \mathbf{M}_{\text{m}} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$


ar
Ray Verrier

Projeção perspectiva



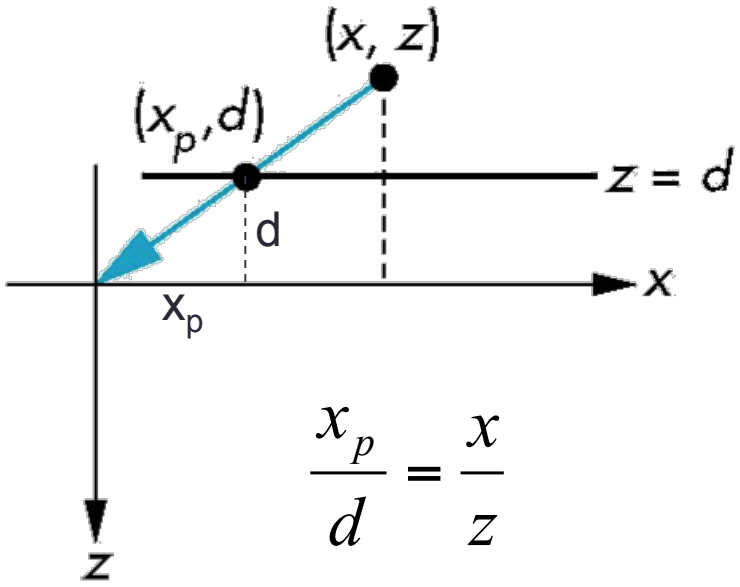
similar triangles:

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

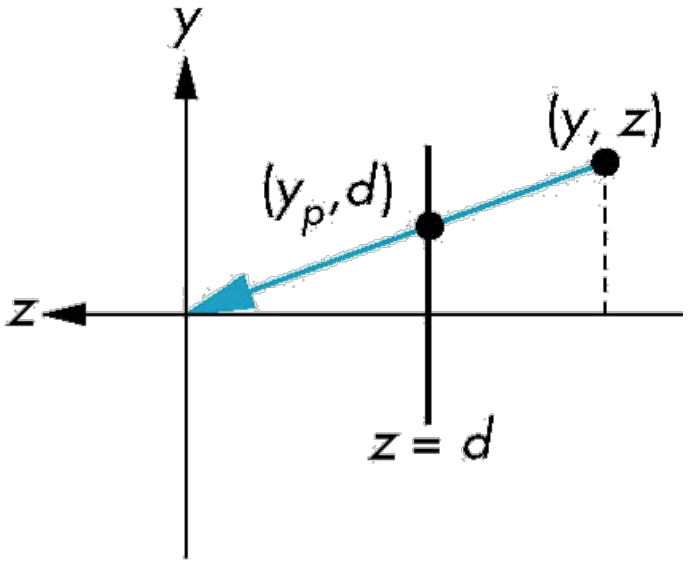
$$y' = -dy/z$$

Equações de perspectiva

Visões superior e lateral



$$\frac{x_p}{d} = \frac{x}{z}$$



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Coordenadas homogêneas

- Perspective requires division
  - that is not part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore not vanishing point
    - therefore no rays converging on viewpoint
- "True" purpose of homogeneous coords: projection

Coordenadas homogêneas

- Introduced $w = 1$ coordinate as a placeholder

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - used as a convenience for unifying translation with linear

- Can also allow arbitrary w

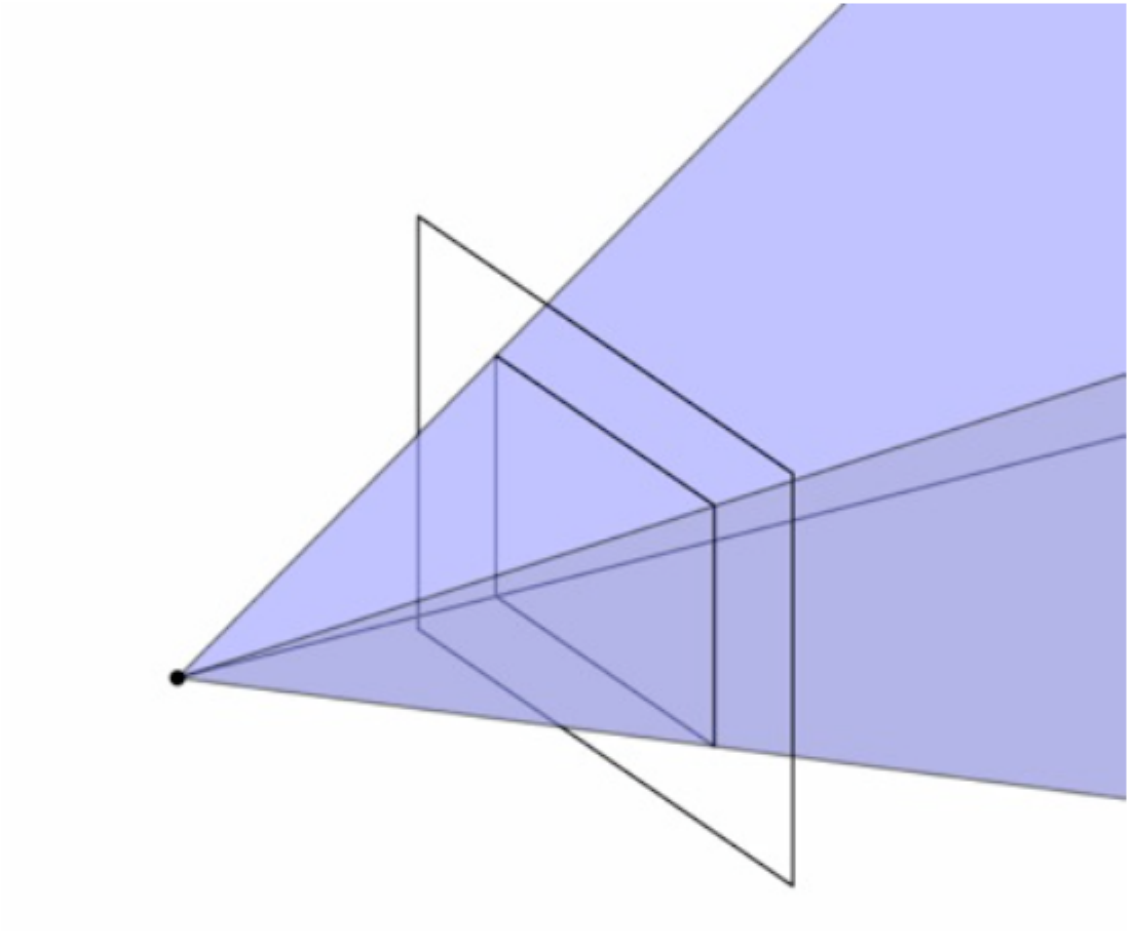
$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

Matriz de projeção perspectiva

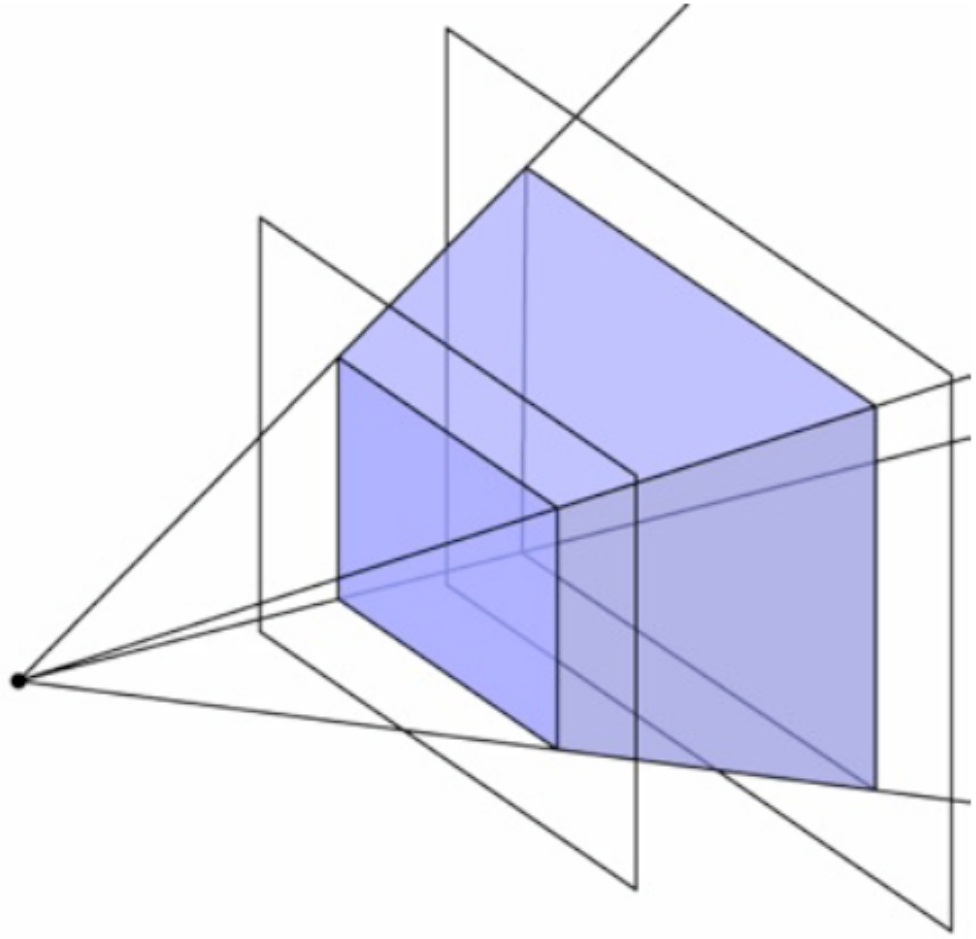
considere $\mathbf{q} = \mathbf{M}\mathbf{p}$ onde $\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \xrightarrow{\text{Divisão de perspectiva}} \mathbf{q}' = \begin{bmatrix} x/z/d \\ y/z/d \\ d \\ 1 \end{bmatrix}$$

Volume de visualização

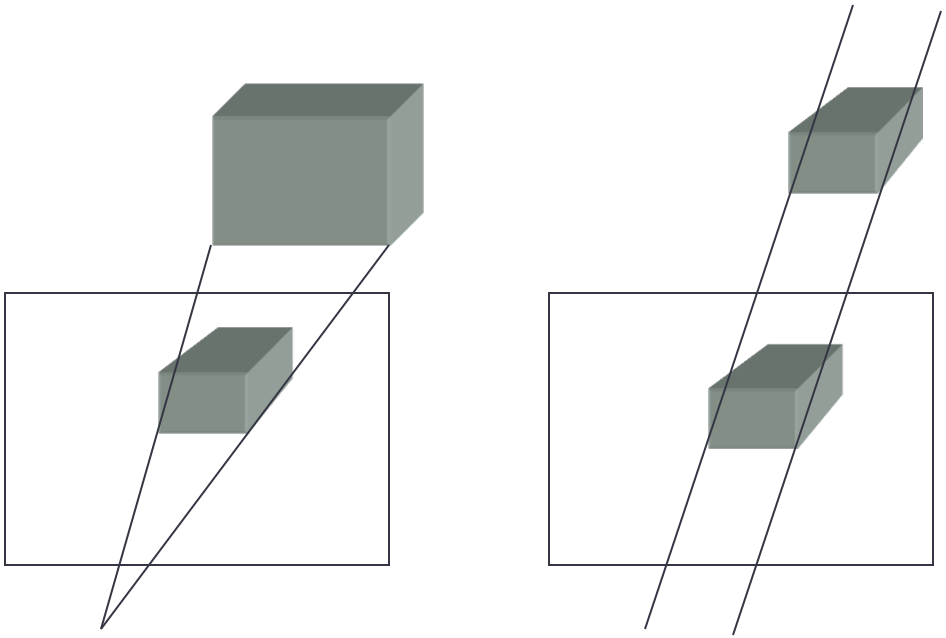


Pirâmide de visualização

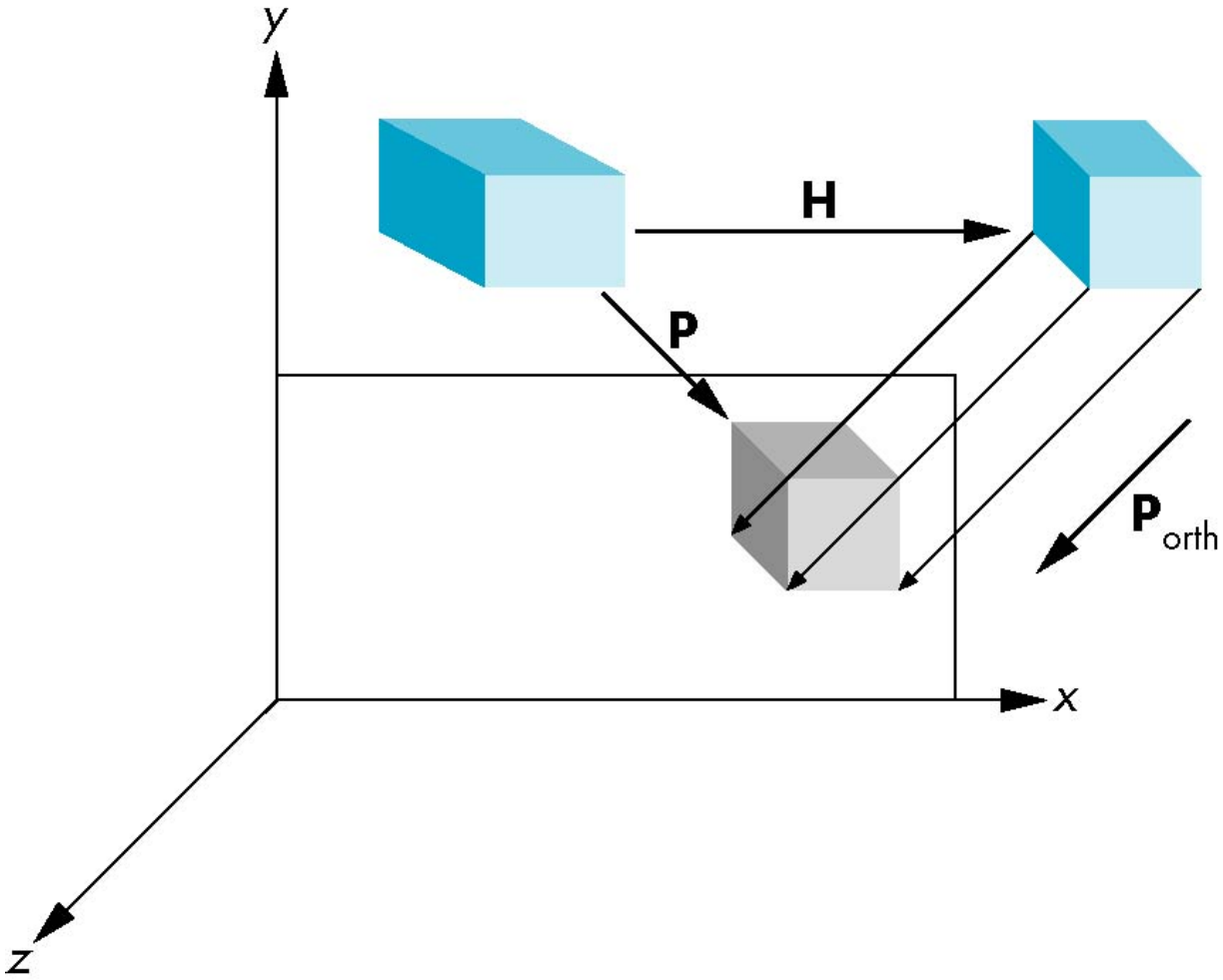


Normalização de projeção

- Primeiro, distorce a cena através de uma transformação afim
- A projeção ortográfica da cena distorcida é a mesma que a projeção original



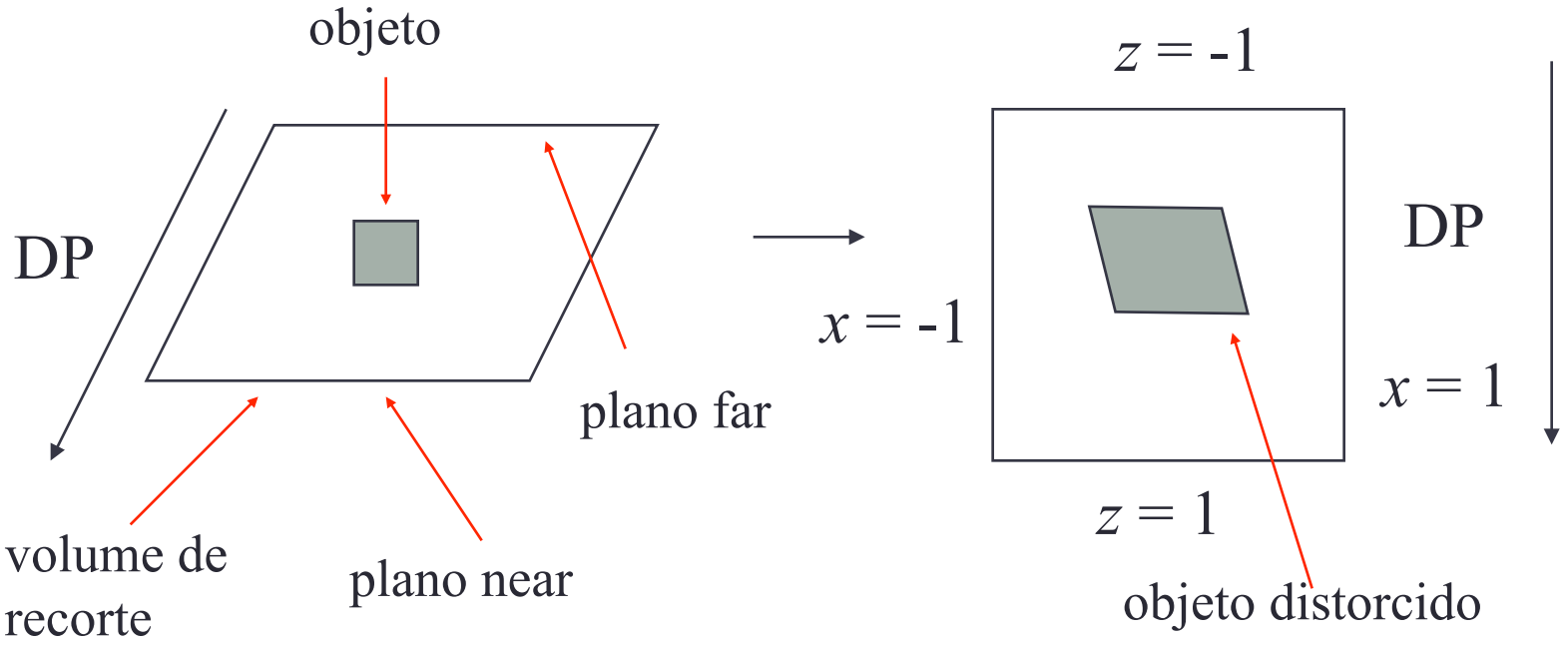
Equivalência



Projeção original = Cisalhamento + Projeção Ortogonal

Efeito no recorte

- A matriz de projeção **P = STH** transforma o volume de recorte original para o volume default



Matriz de perspectiva

Matriz em coordenadas homogêneas

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Note que esta matriz é independente do plano de recorte de trás

Generalização

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

após a divisão de perspectiva, o ponto (x, y, z, 1) traduz-se em

$$x'' = -x/z$$
$$y'' = -y/z$$
$$z'' = -(\alpha+\beta/z)$$

que projeta ortogonalmente para o ponto desejado independente dos valores de α e β

Normalização de perspectiva

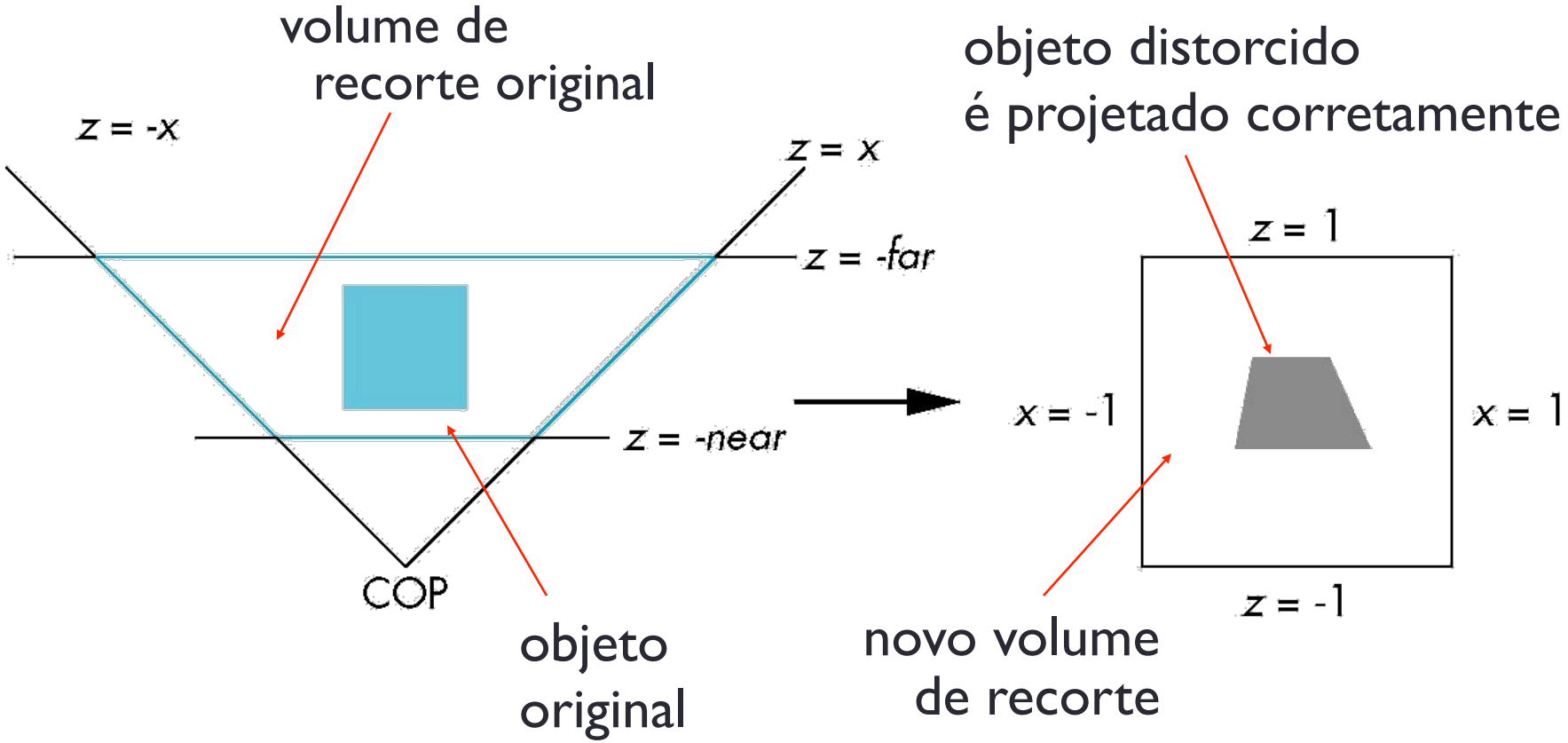
- Se transformarmos pontos através de N, e depois fizermos uma projeção ortogonal, obtemos o mesmo resultado de uma projeção perspectiva.
- Análogo às projeções oblíquas, onde fizemos um cisalhamento e então uma projeção ortogonal.

$$\mathbf{M}_{\text{orth}}\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{M}_{\text{orth}}\mathbf{N}\mathbf{p} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix}$$

$$x'' = -x/z$$
$$y'' = -y/z$$

Transformação da normalização



Valores para α e β

Se selecionarmos

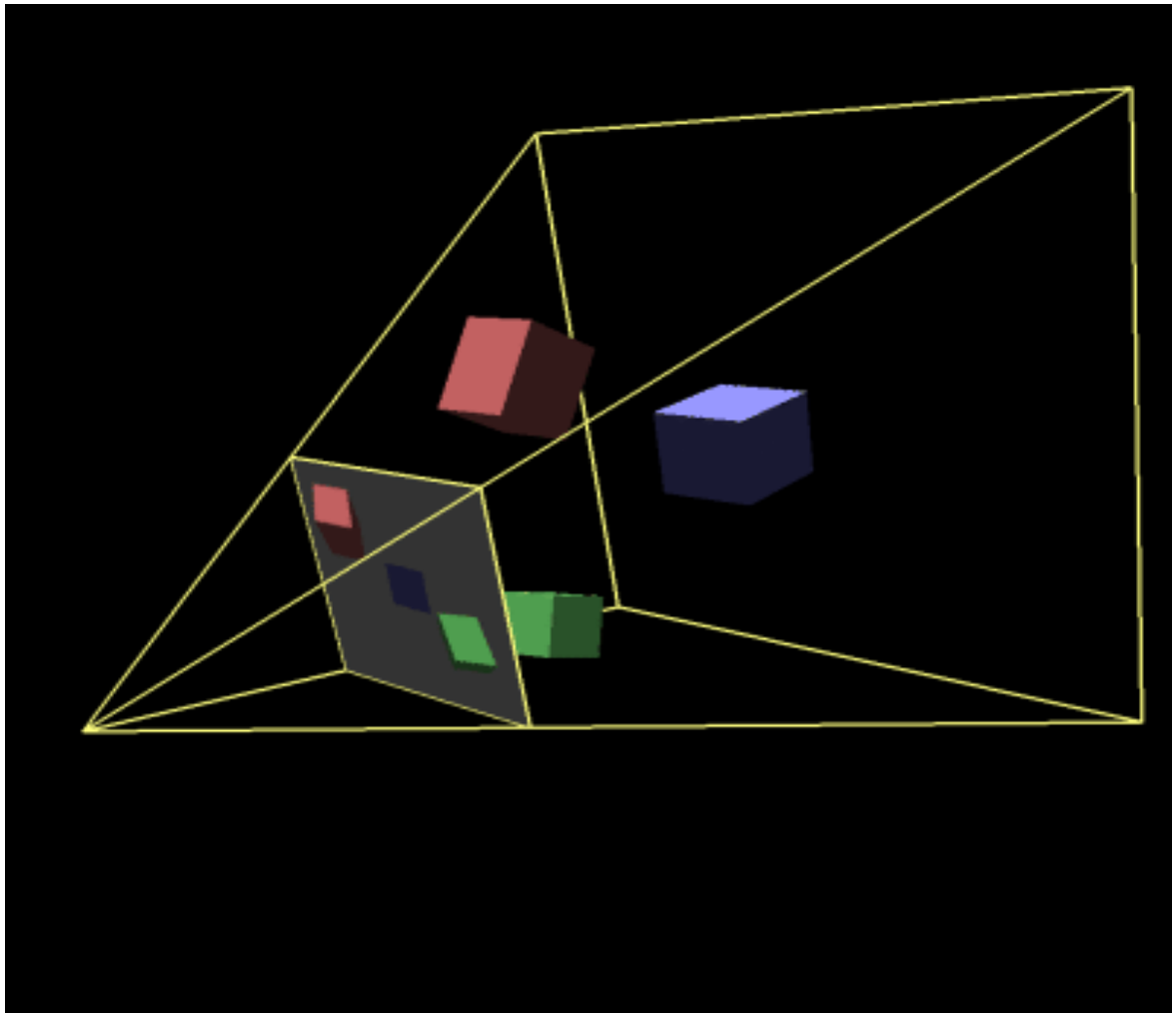
$$\alpha = -\frac{\text{near} + \text{far}}{\text{far} - \text{near}}$$

$$\beta = -\frac{2 \cdot \text{near} \cdot \text{far}}{\text{near} - \text{far}}$$

o plano -near é mapeado em $z = -1$
o plano -far é mapeado em $z = 1$
e os lados são mapeados em $x = \pm 1, y = \pm 1$

Assim o novo volume de recorte se torna o volume de canônico (volume default)

Visão perspectiva



Transformação perspectiva

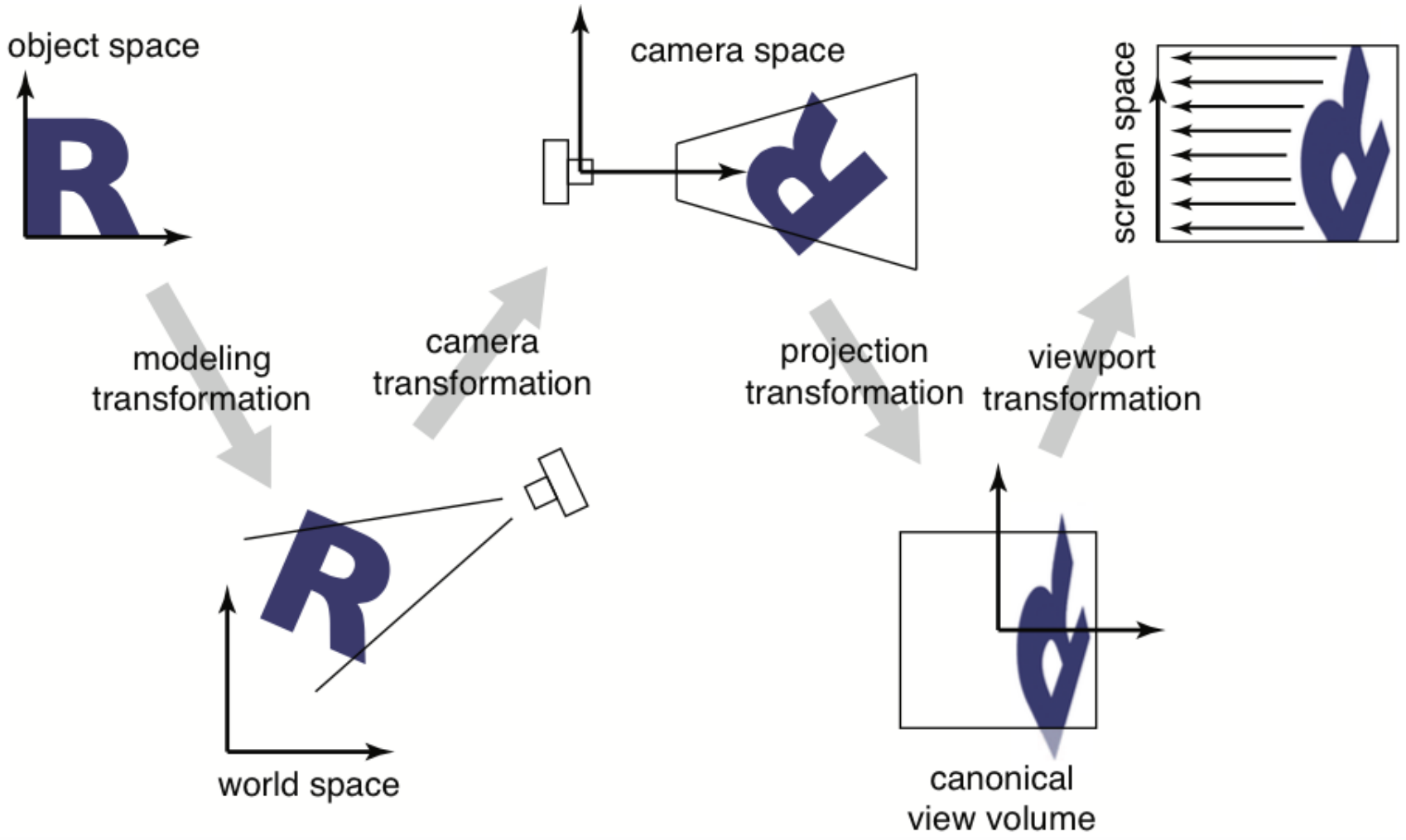
- Transform into world coords (modeling transform, M_m)
- Transform into eye coords (camera xf., $M_{cam} = F_c^{-1}$)
- Perspective matrix, P
- Orthographic projection, M_{orth}
- Viewport transform, M_{vp}

$$\mathbf{p}_s = \mathbf{M}_{\text{vp}}\mathbf{M}_{\text{orth}}\mathbf{P}\mathbf{M}_{\text{cam}}\mathbf{M}_{\text{m}}\mathbf{p}_o$$

Vantagens

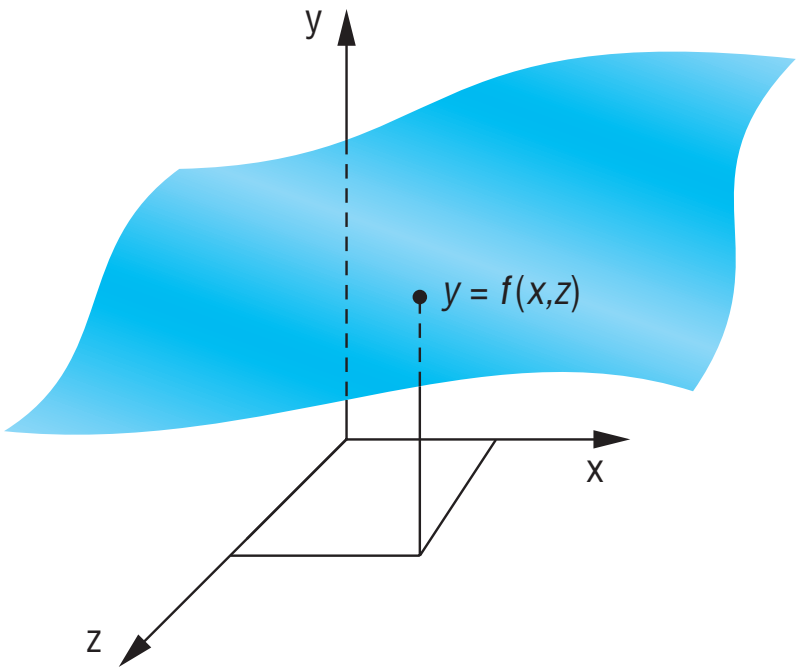
- A normalização nos permite usar um único pipeline tanto para a visualização perspectiva e ortogonal
- Permanecemos em 4D até o mais tardar possível para reter informação em 3D necessária para remoção de superfícies escondidas e tonalização
- Simplificação de recortes

Pipeline de transformações

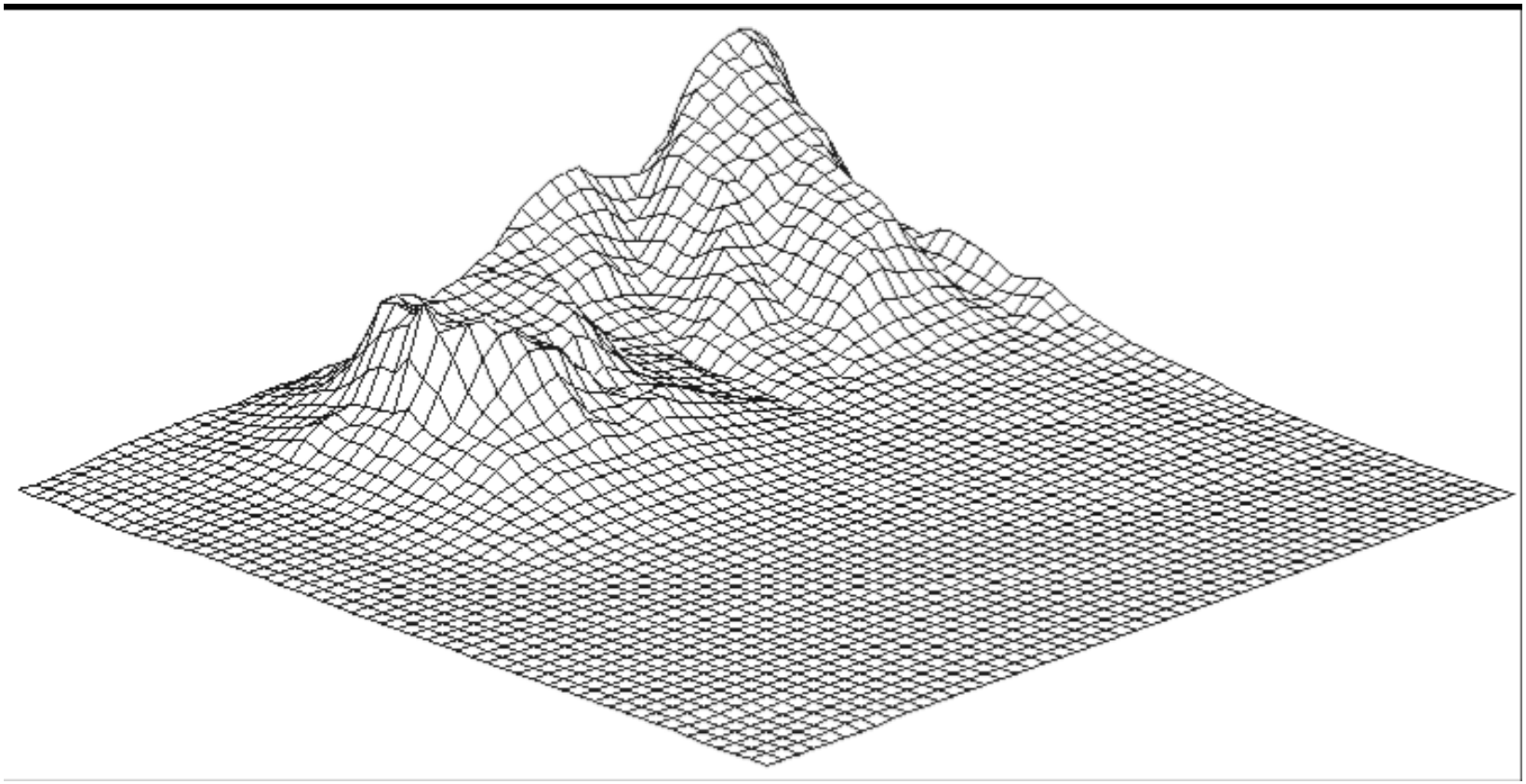


Campos de altura (*height fields*)

- Para cada tupla (x, z) existe um único y
- Amostragem resulta em um *array* de valores y
- Podemos visualizá-la usando malhas triangulares

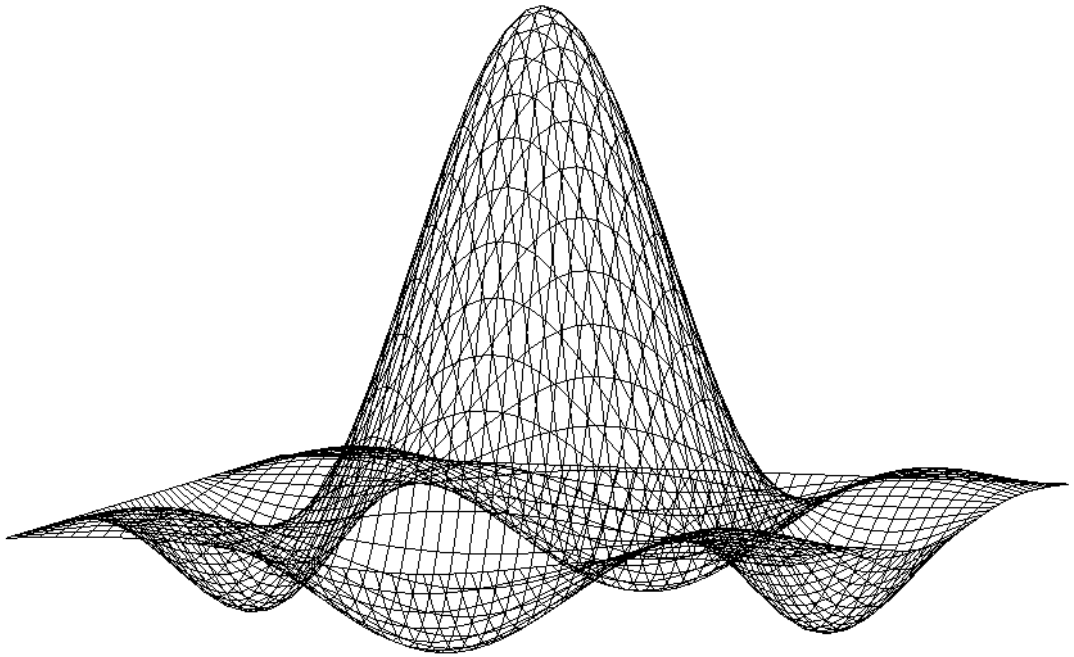


Line Strips



Remoção de linhas escondidas

sombrero or Mexican hat function (sin πr)/(πr)

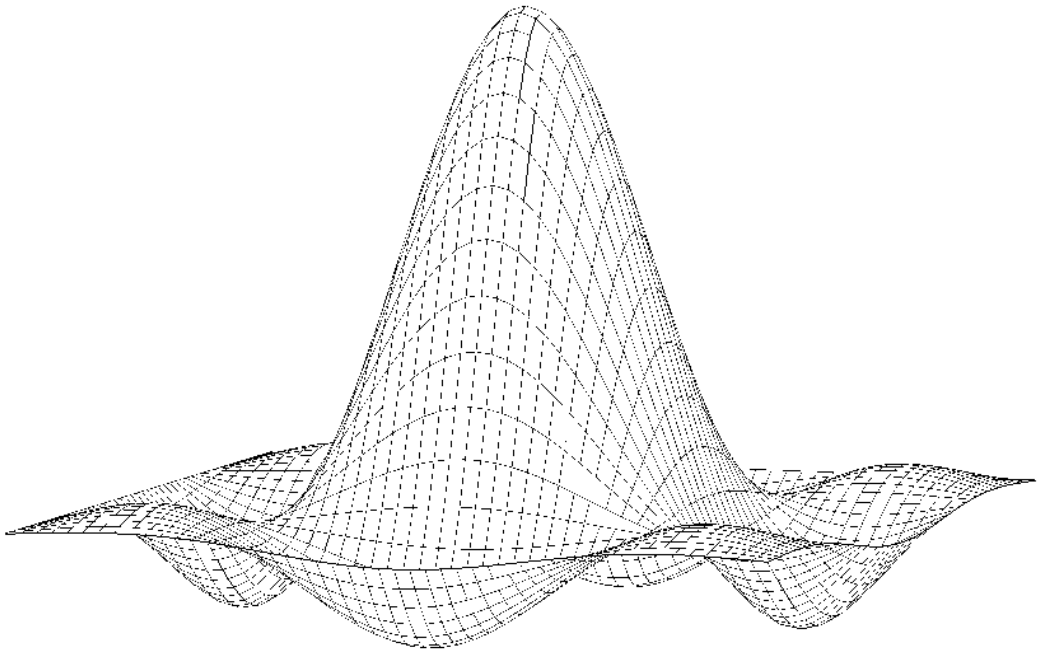


```
glDrawArrays(GL_LINE_STRIP, 0, N*M);
```

Utilizando polígonos

- Podemos utilizar quatro pontos adjacentes para formar um quadrilátero
  - Assim dois triângulos compartilharão uma aresta
- Mas, e se quisermos visualizar a grade ?
- Podemos renderizar a grade duas vezes
  - Primeiro como triângulos (com a cor de fundo)
  - Segundo com line loops (em preto)

Visualização com triângulos e linhas

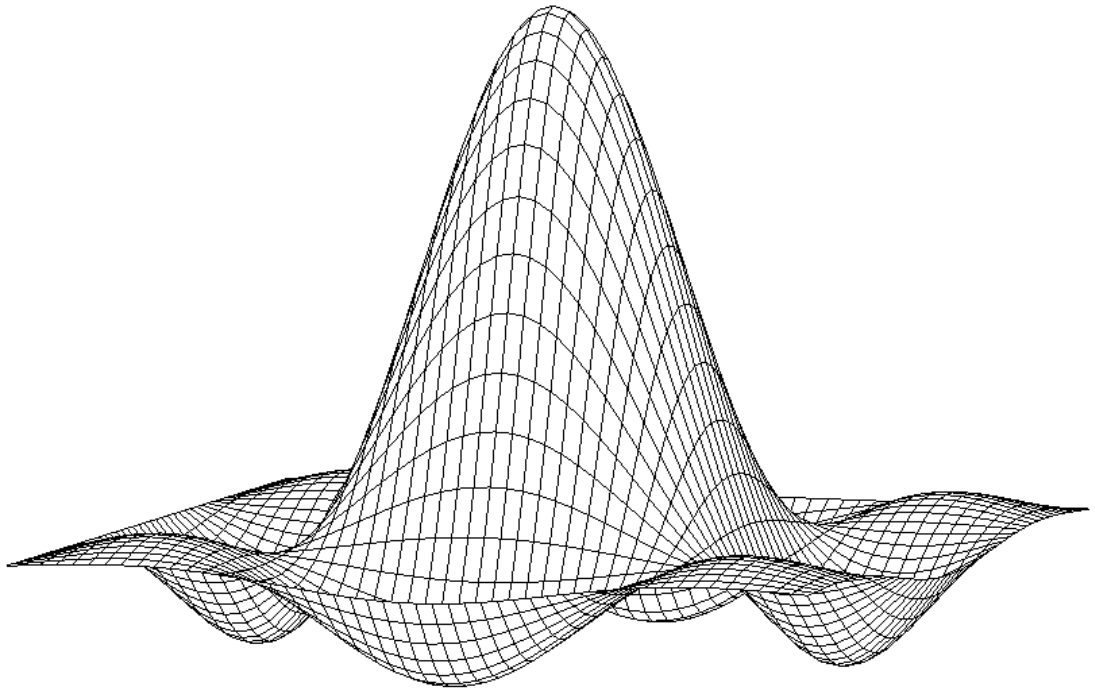


Polygon Offset

- Mesmo que desenhássemos o polígono primeiro com triângulos e depois com linhas, erros numéricos farão com que fragmentos das linhas fiquem ocultos
- Um truque é utilizar o que é chamado de *polygon offset*, que faz com que fragmentos afastem-se sutilmente do observador
- Aplica-se à renderização das faces

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0, 1.0);
```

Visualização com *Polygon Offset*



Tarefa de casa

- Leitura livro-texto
  - Shirley and Marschner. Fundamentals of Computer Graphics, CRC Press, 3rd Ed. 2010
  - Capítulo 7