Computação Gráfica
Visualização 3D

www.dca.ufrn.br/~lmarcos/courses/compgraf

Visualização 3D

- De onde veio a geometria?
- Modelo de Câmera Pin-hole
- Transformações de Projeção (perspectiva e ortogonal)
- Transformação de Visualização
- Cortes de partes não visíveis

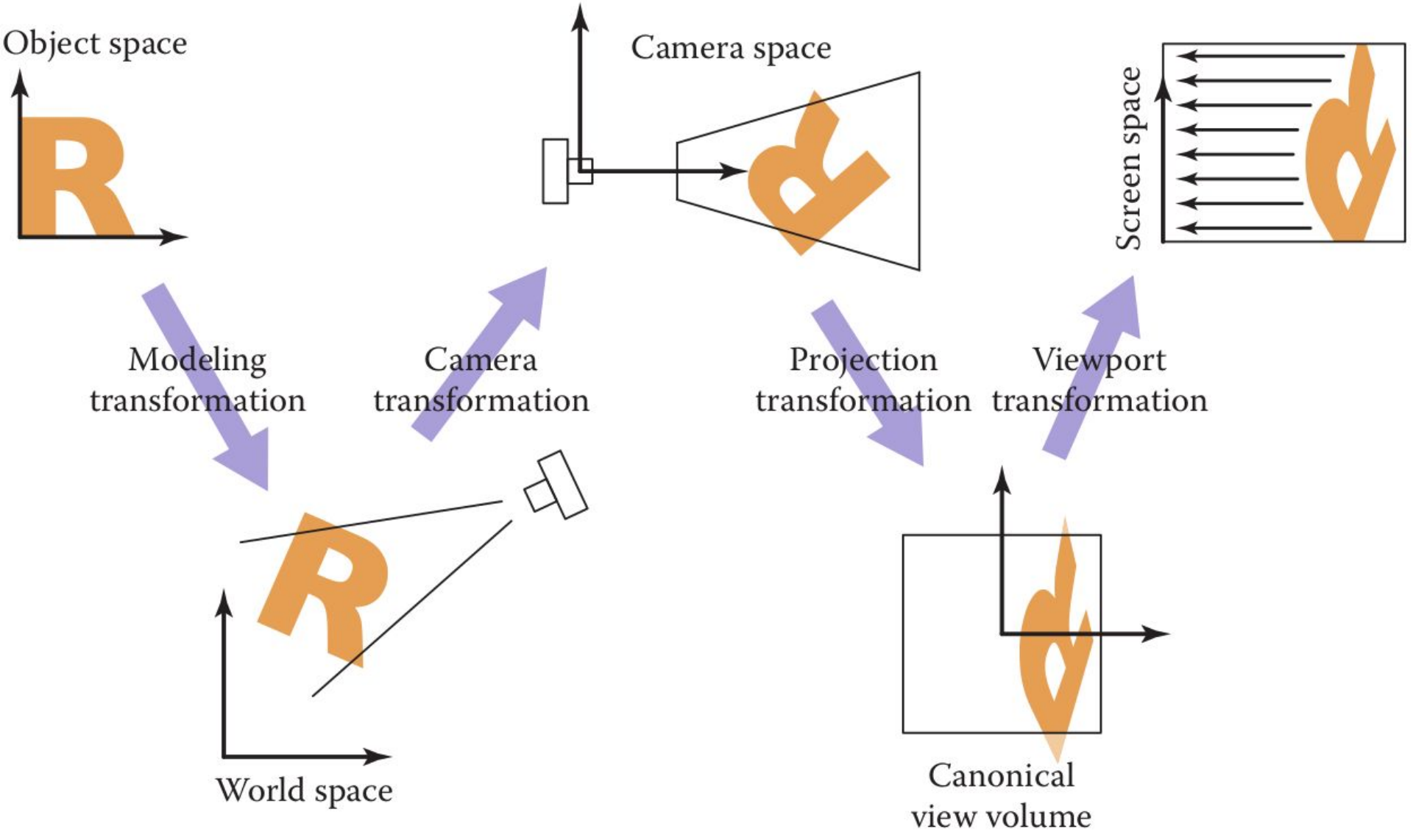


Figure 7.2. The sequence of spaces and transformations that gets objects from their original coordinates into screen space.

De onde veio a geometria?

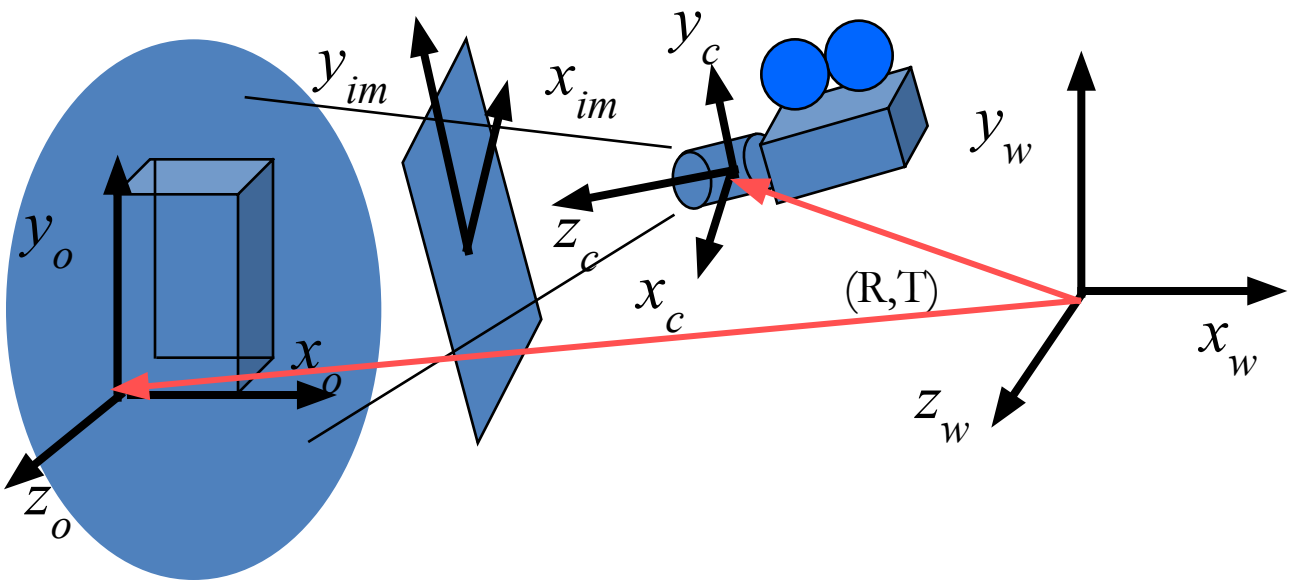
- Construída usando modeladores 3D (Maya, 3DS, etc)
- Digitalizada ou escaneada (escaner 3D)
- Simulação/modelagem física
- Combinação
  - Editar um modelo digitalizado
  - Simplificar um modelo escaneado
  - Evoluir um modelo simples (esfera, cubo)
- Geralmente precisa-se modelos múltiplos a complexidades diferentes

Modelagem Geométrica

- Resultado é uma representação no computador:
  - Esfera = origem (x_0,y_0,z_0), raio (r), angulo (o_x,o_y,o_z);
- Maneiras mais eficientes de representar objetos (Modelagem Geométrica)
    - Para superfícies mais complexas:
      - Combinação de primitivas simples (cubo + esfera)
      - Uso de retalhos poligonais (colcha de retalhos)
      - Retalhos de Bèzier ou retalhos triangulares

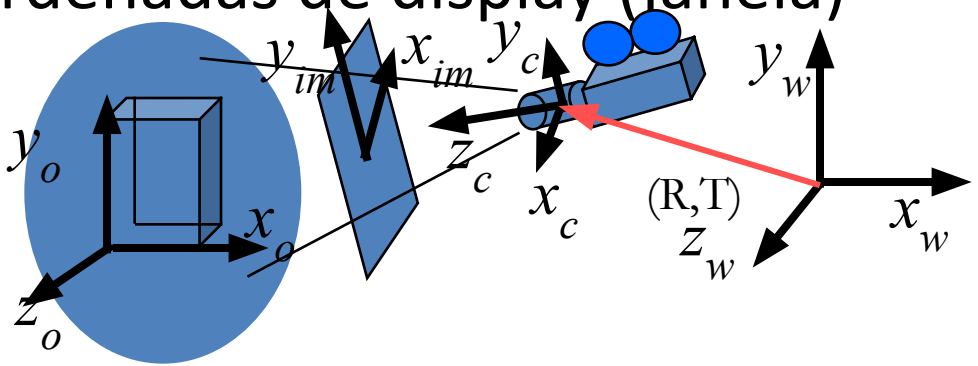
Dada a Geometria

- Como mostrar a geometria na tela?
- Dados: - modelo matemático da cena no sistema de coordenadas de mundo
  - câmera também referenciada ao sistema de mundo (R,T)



Colocando geometria na tela

- Dada a geometria (ponto) num sistema de coordenada de mundo, como visualizar na tela (pixel)?
  - Transformar para sistema de câmera
  - Transformar (warp) para o volume de visualização canônico
  - Clip (cortar partes não visíveis)
  - Projetar em coordenadas de display (janela)
  - Rasterizar

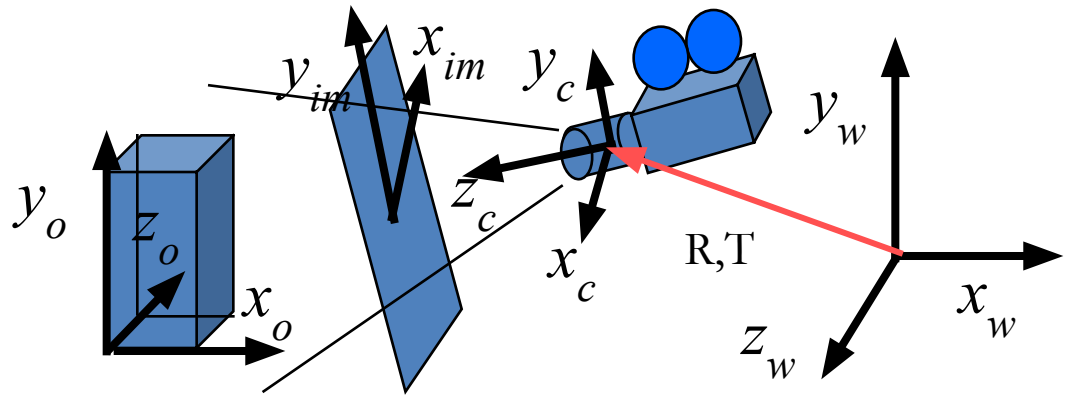


Visualização e projeção

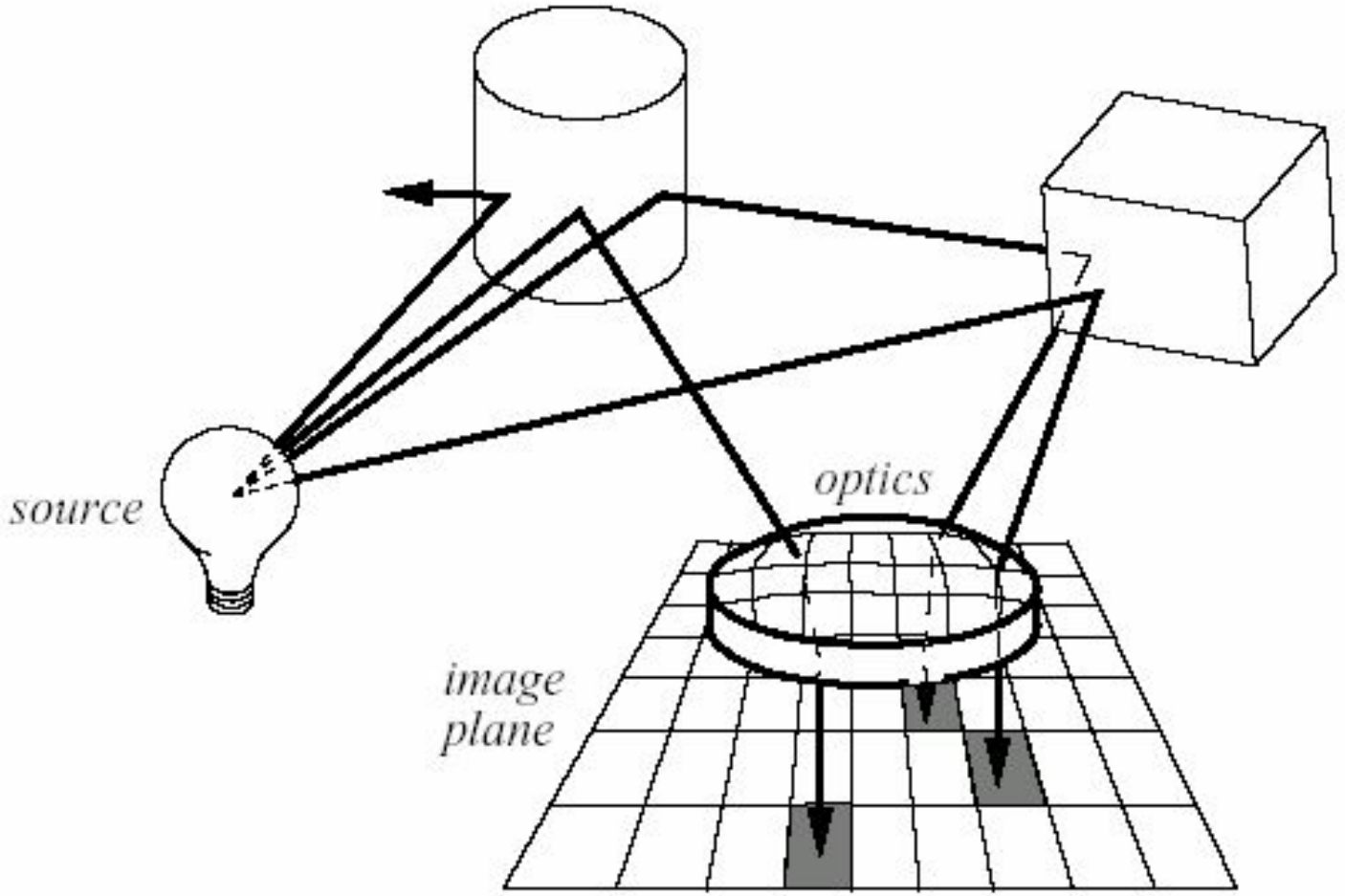
- Nossos olhos colapsam o mundo 3D para uma imagem 2D na retina
- Este processo ocorre por *Projeção*
- Em CG, projeção tem duas partes:
  - Transformação de visualização: posição da câmera e sua direção (orientação)
  - Transformação de projeção (perspectiva ou ortogonal): reduz 3D para 2D

Visualização e projeção

- Usar Transformações Homogêneas
- Formam a raiz da hierarquia de transformações
- Transformar coordenadas de câmera em mundo, objeto e imagem e vice-versa

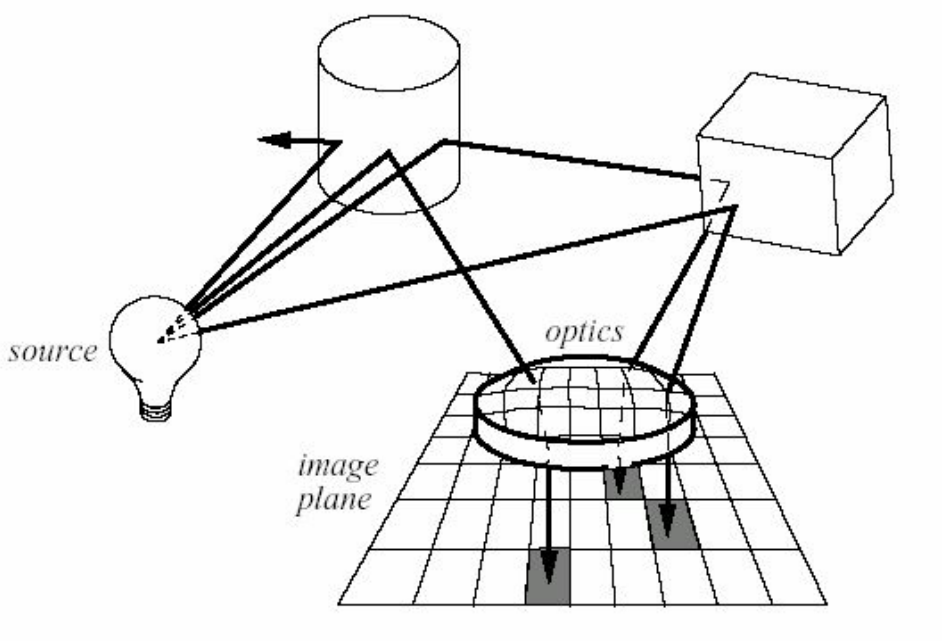


Ótica básica



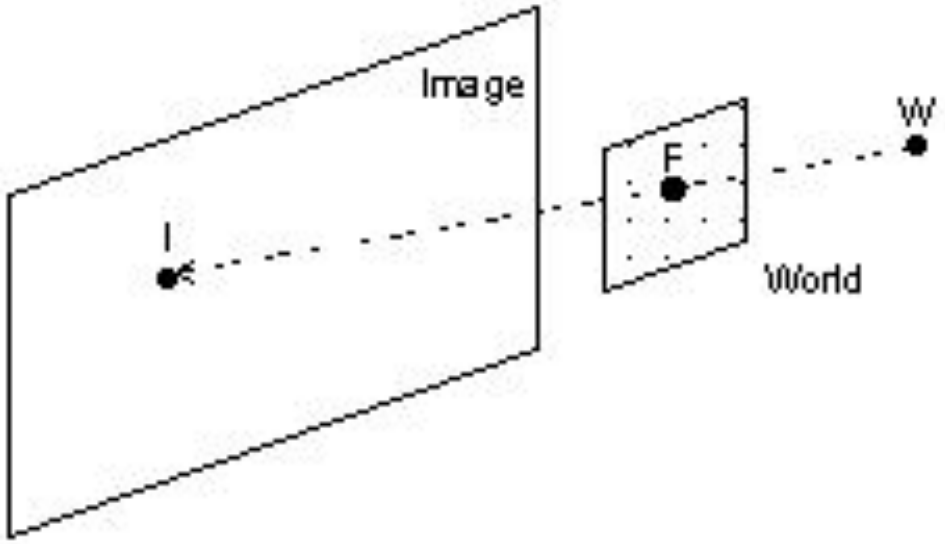
Ótica básica

- Formação da imagem começa com o raio de luz que entra na câmera através da abertura angular (pupila num ser humano)
- Raio bate na tela ou plano imagem e o sensor fotoreceptivo registra intensidade da luz
- Muitos raios vêm de luz refletida e alguns de luz direta



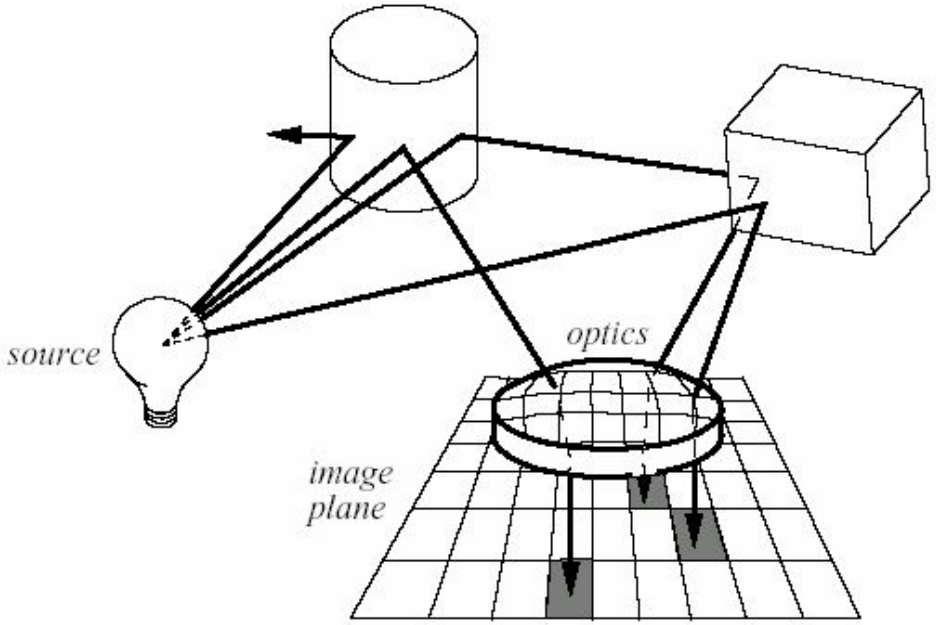
Projeção perspectiva de um ponto

- Plano de visualização ou plano imagem
  - ponto *I* “enxerga” tudo ao longo do raio que passa através de F
  - ponto W projeta ao longo do raio que passa por F em I (interseção da reta WF com plano imagem)



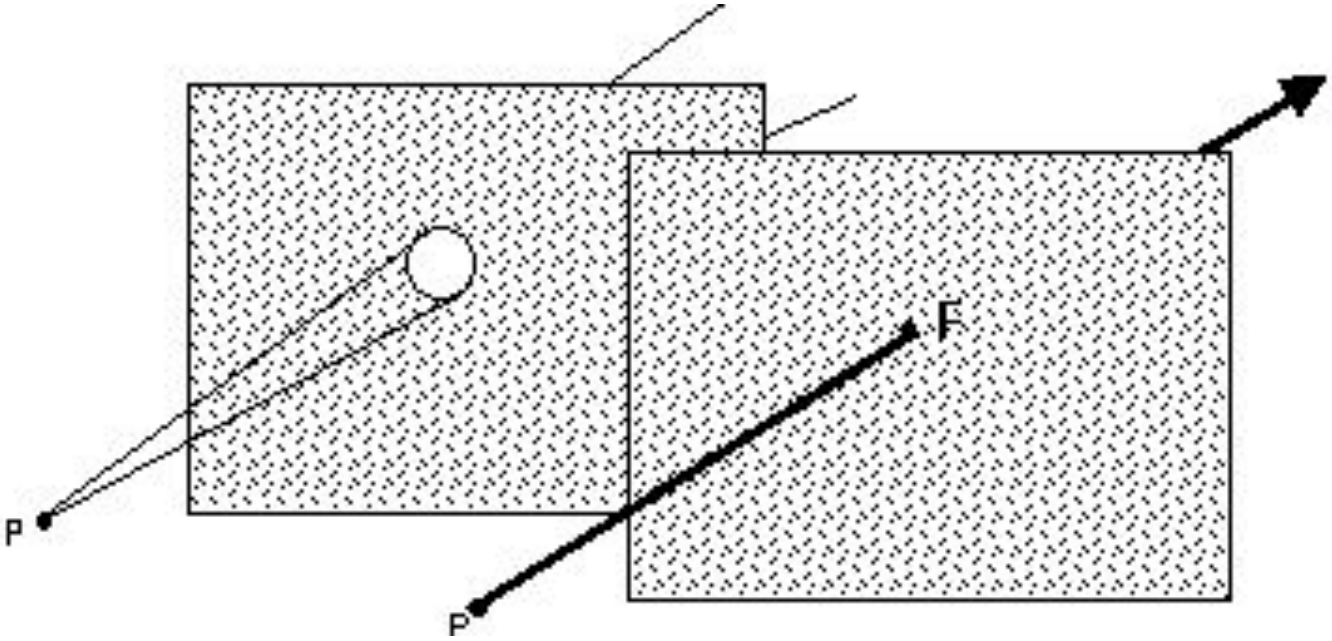
Focando uma imagem

- Qualquer ponto numa cena pode refletir raios vindos de várias direções
- Muitos raios vindos do mesmo ponto podem entrar na câmera.
- Para ter imagem nítida, todos os raios vindos de um mesmo ponto *P* da cena devem convergir para um ponto único *p* na imagem.



Ótica da câmera pin-hole

- De P, a região visível olhando por F é um cone
  - Problema: imagem borrada (difusa, espalhada)
- Reduzindo o furo, o cone reduz-se a um raio
- “Pin-hole” é o ponto focal ou centro de projeção



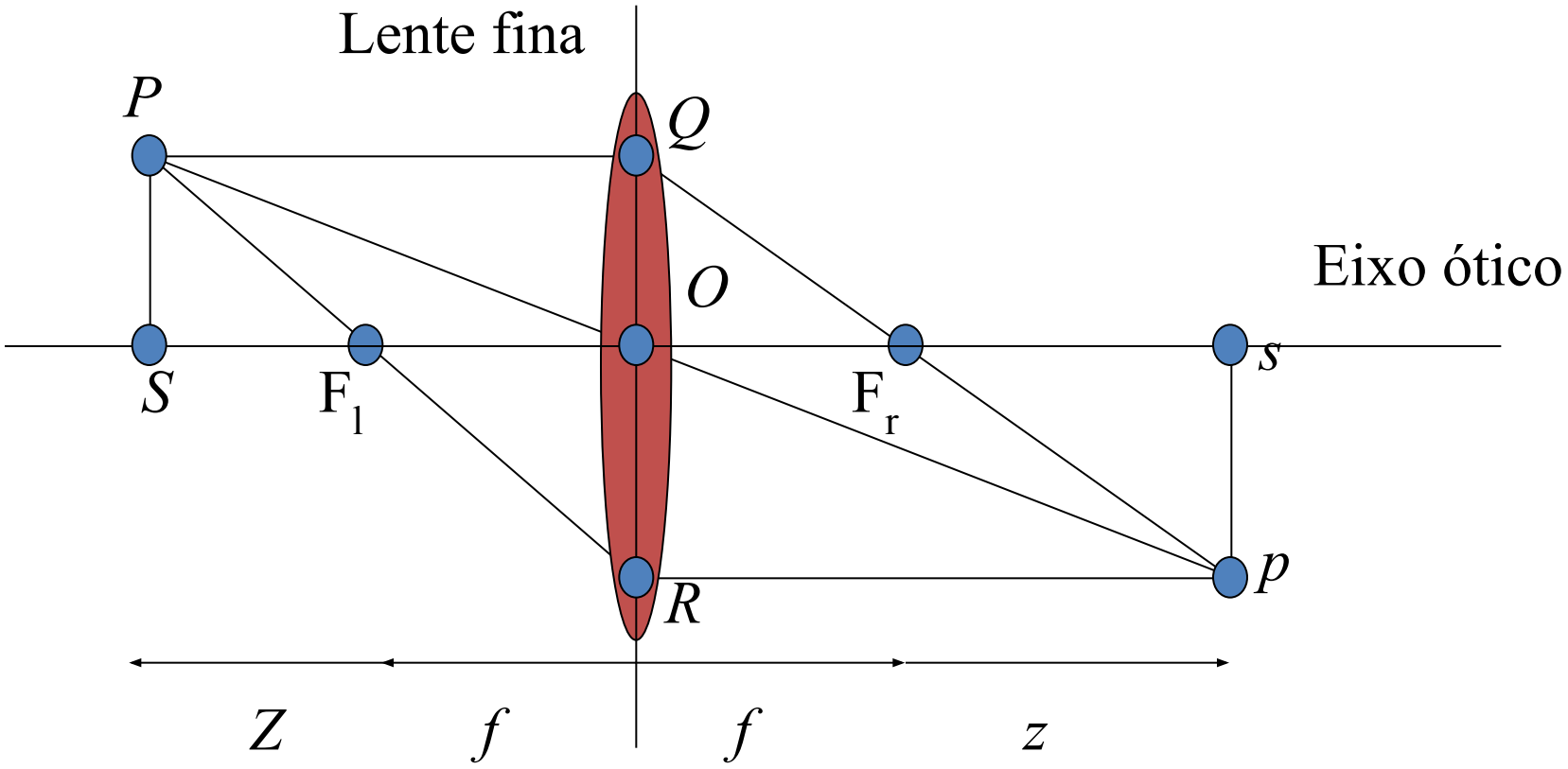
Reduzindo abertura (pin-hole)

- Apenas um raio de cada ponto entra na câmera
  - Imagens nítidas, sem distorções, mesmo à distâncias diferentes
- Problemas com o modelo pin-hole:
  - Tempo de exposição longo
  - Quantidade mínima de luz
  - Difração
- Como resolver isso?

Introduzindo sistema ótico

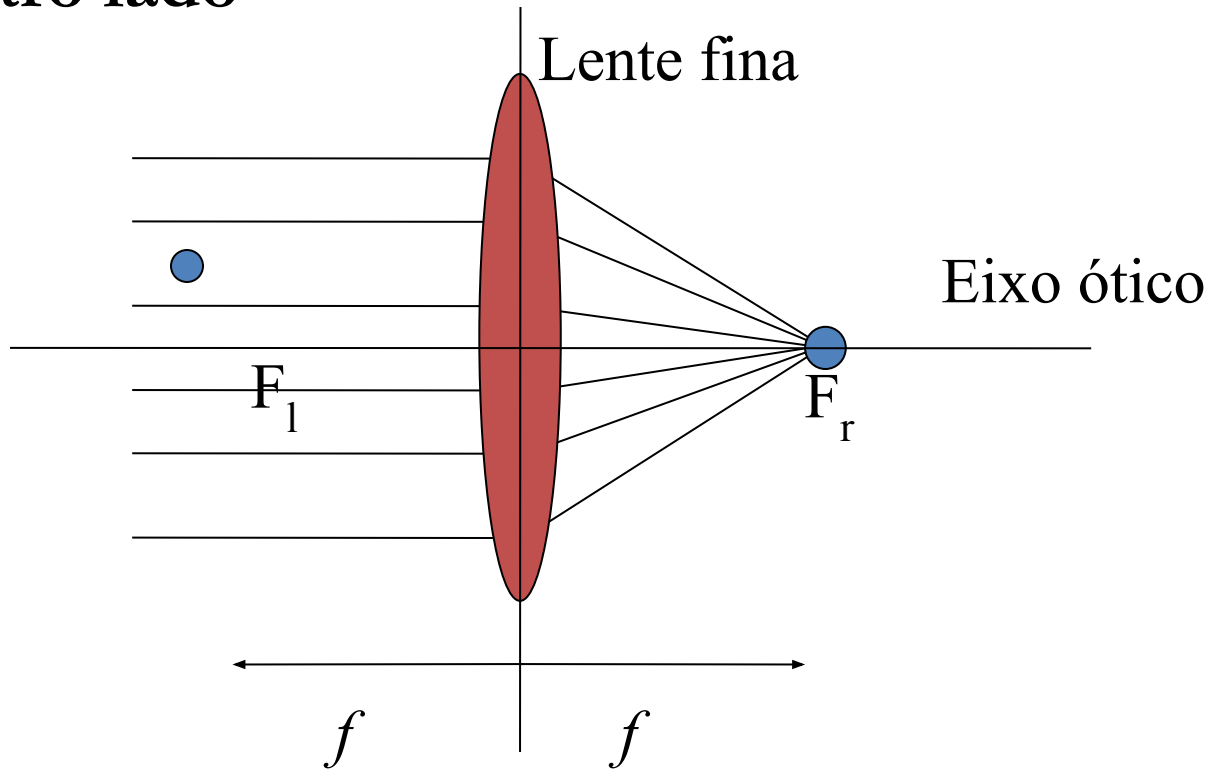
- Solução: introduzir lentes e abertura
- Raio vindo do mesmo ponto 3D converge para um único ponto na imagem
- Mesma imagem que uma pin-hole mas com tempo de exposição bem menor e abertura maior

Sistema de lente fina



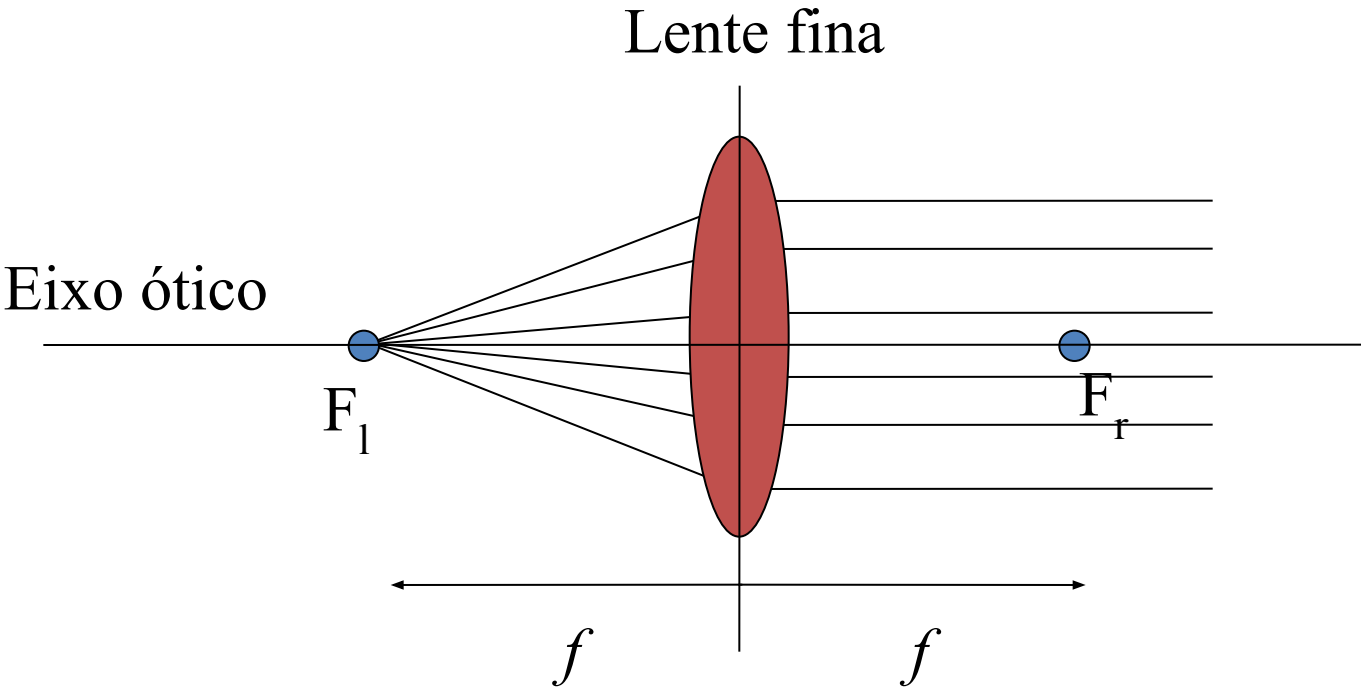
Restrição paralelismo-foco

1) Qualquer raio que entra no sistema de lentes paralelo ao eixo ótico, sai na direção do foco no outro lado



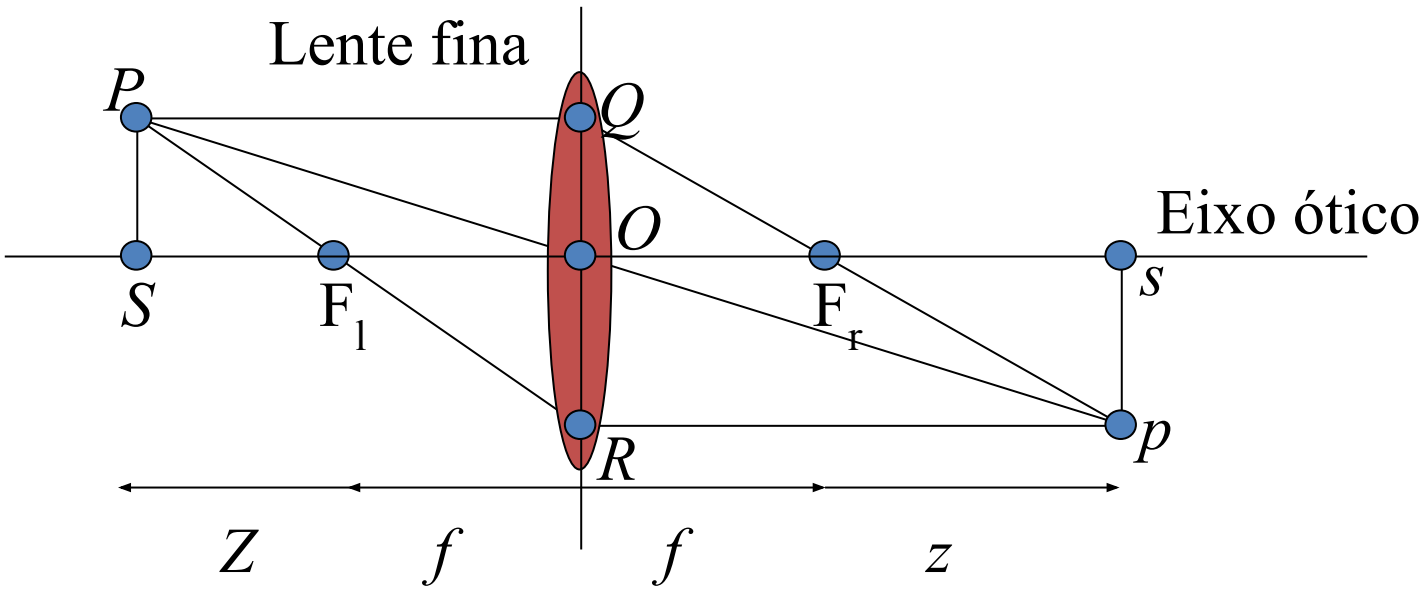
Restrição paralelismo-foco

2) Qualquer raio que entra na lente vindo da direção do foco, sai paralelo ao eixo ótico do outro lado



Modelo básico

- Propriedade 1) a *PQ* e propriedade 2) a *PR*
  - Esses raios defletem para se encontrar em algum ponto do outro lado
- Uma vez que o modelo de lente fina foca todos os raios vindos de *P* convergem para o mesmo ponto, *PQ* e *PR* se intersectam em *p*



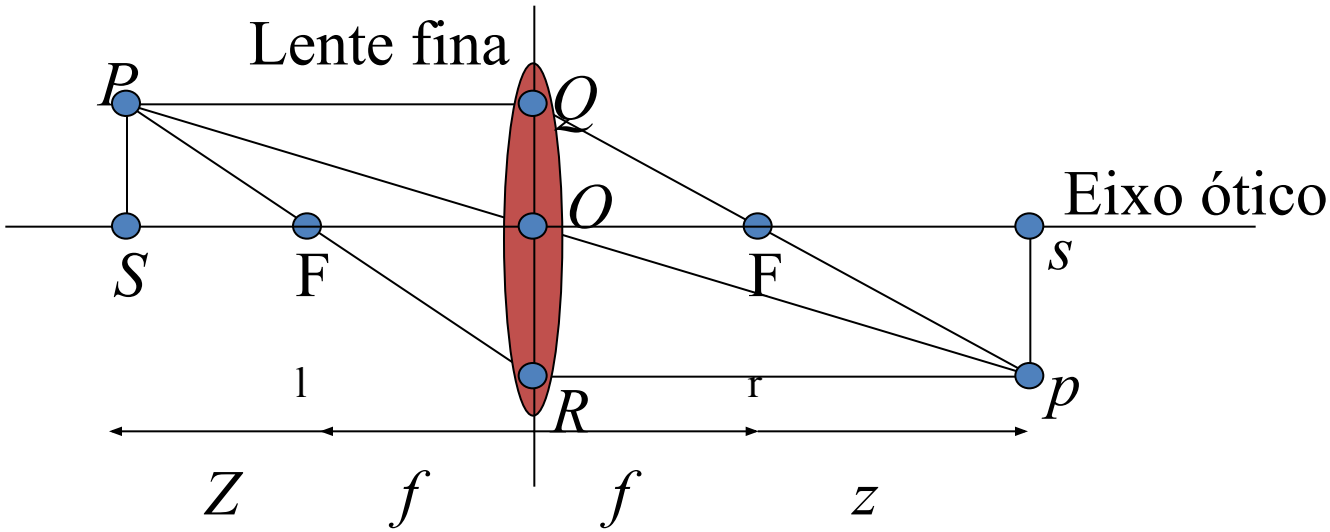
Equação fundamental

- Usando similaridade entre os pares de triângulo *(<PF$_l$S>, <ROF$_l$>) e (<psF$_r$>, <QOF$_r$>)*, obtém-se:

  $Z/f = f/z \Rightarrow Zz = f^2$

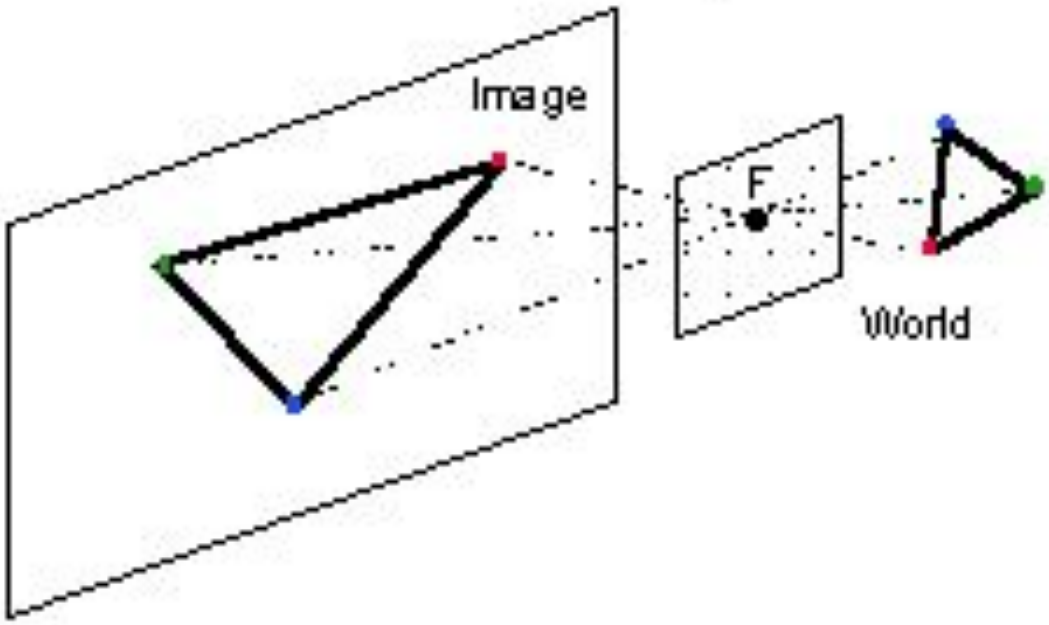
- Fazendo *Z´=Z+f e z´= z+f, encontramos*:

  $1/Z' + 1/z' = 1/f$

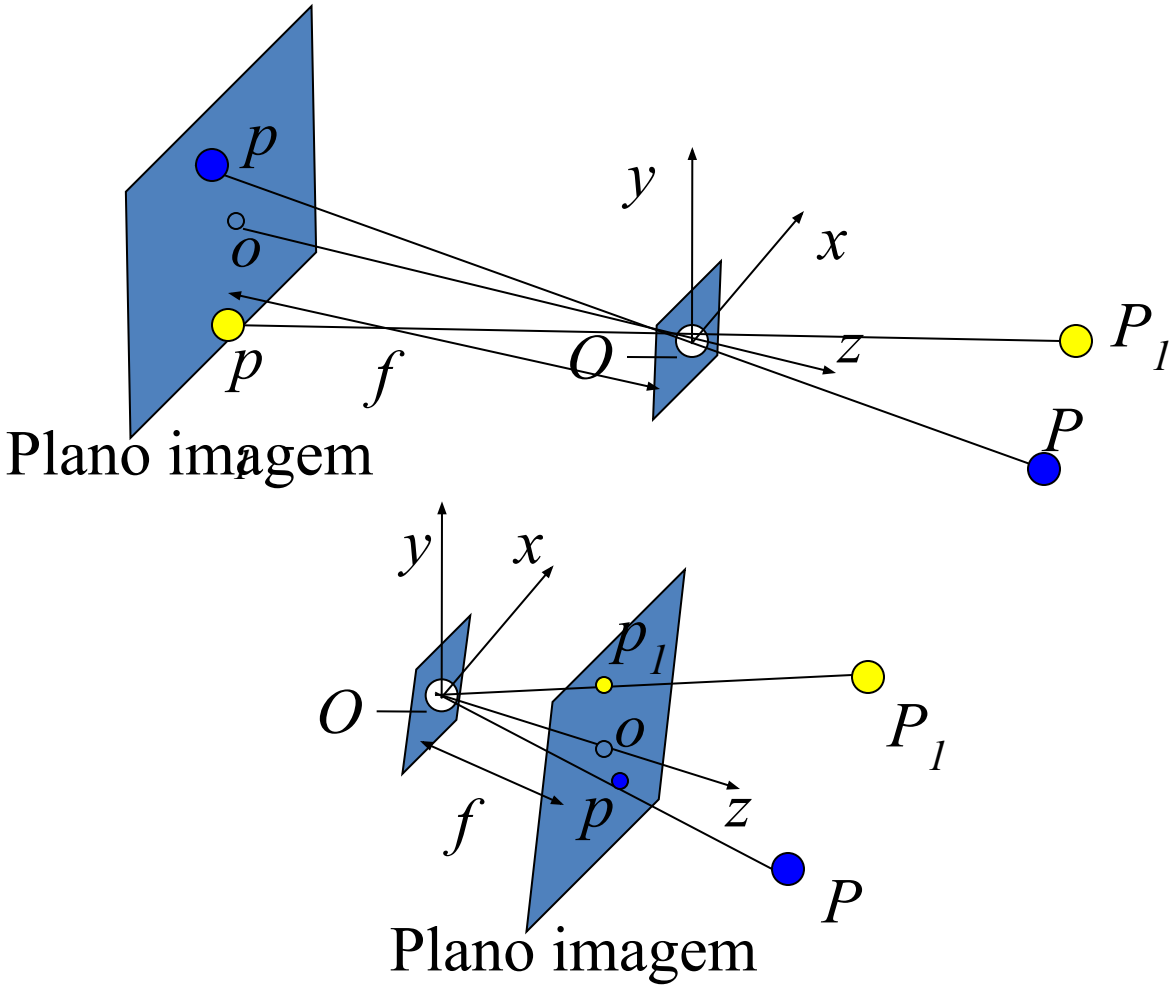


Formação da imagem (invertida)

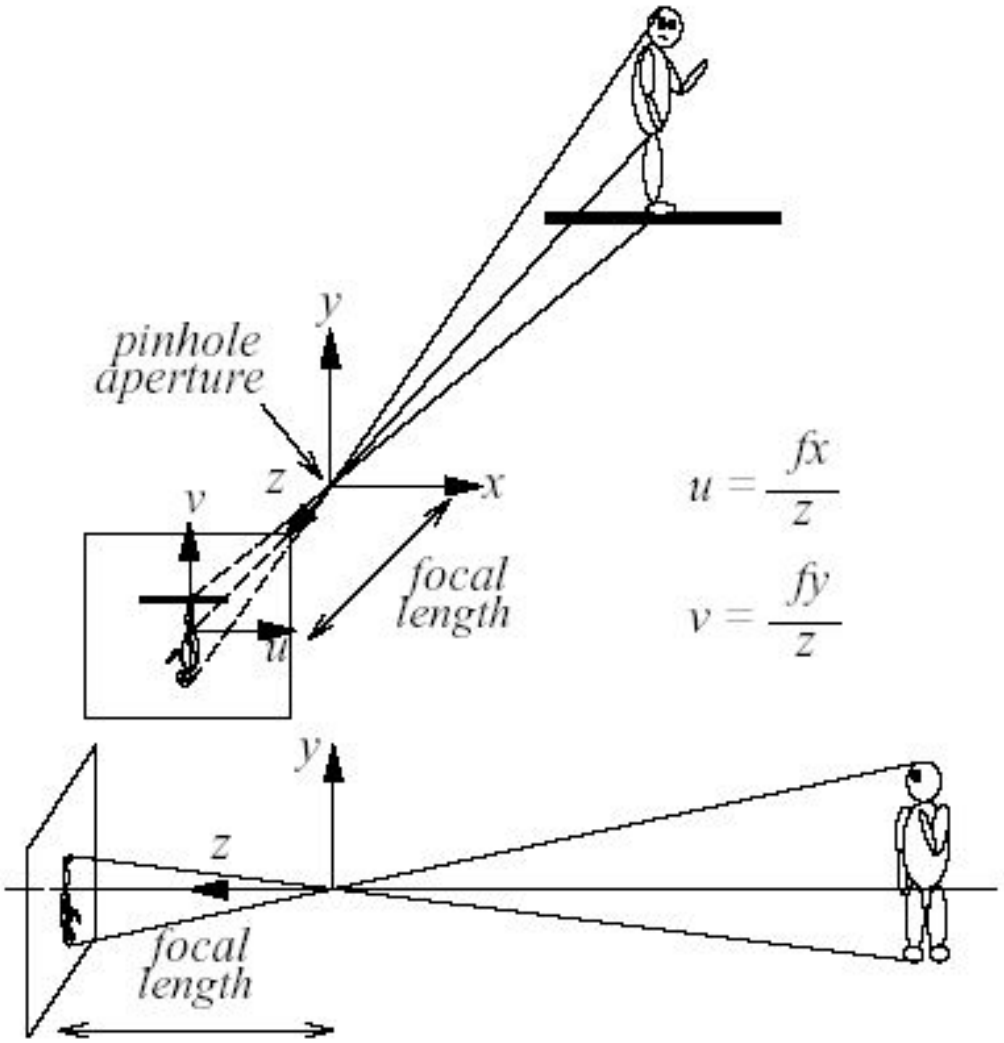
- Projetando uma forma
  - Projetar cada ponto no plano imagem
  - Linhas são projetadas projetando seus pontos finais



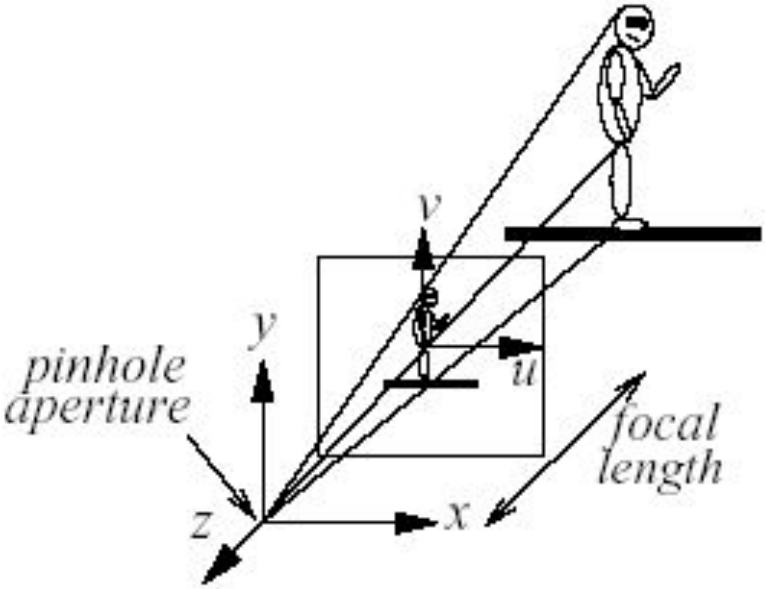
Modelo perspectivo ideal



Projeção perspectiva (pin-hole)

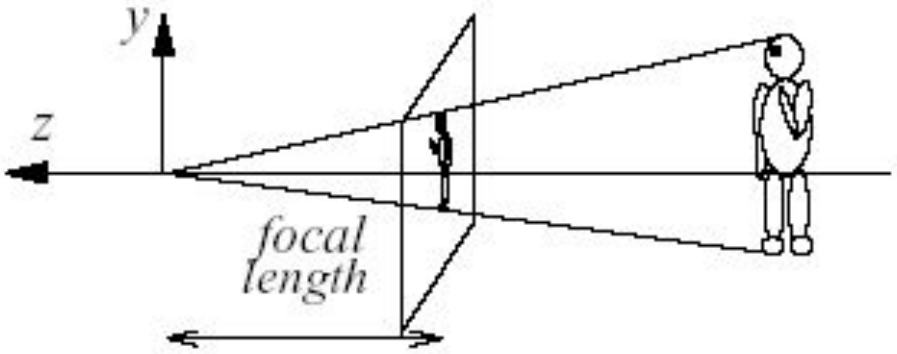


Modelo ideal



$$u = \frac{fx}{(f-z)}$$

$$v = \frac{fy}{(f-z)}$$



Implementação do modelo ideal

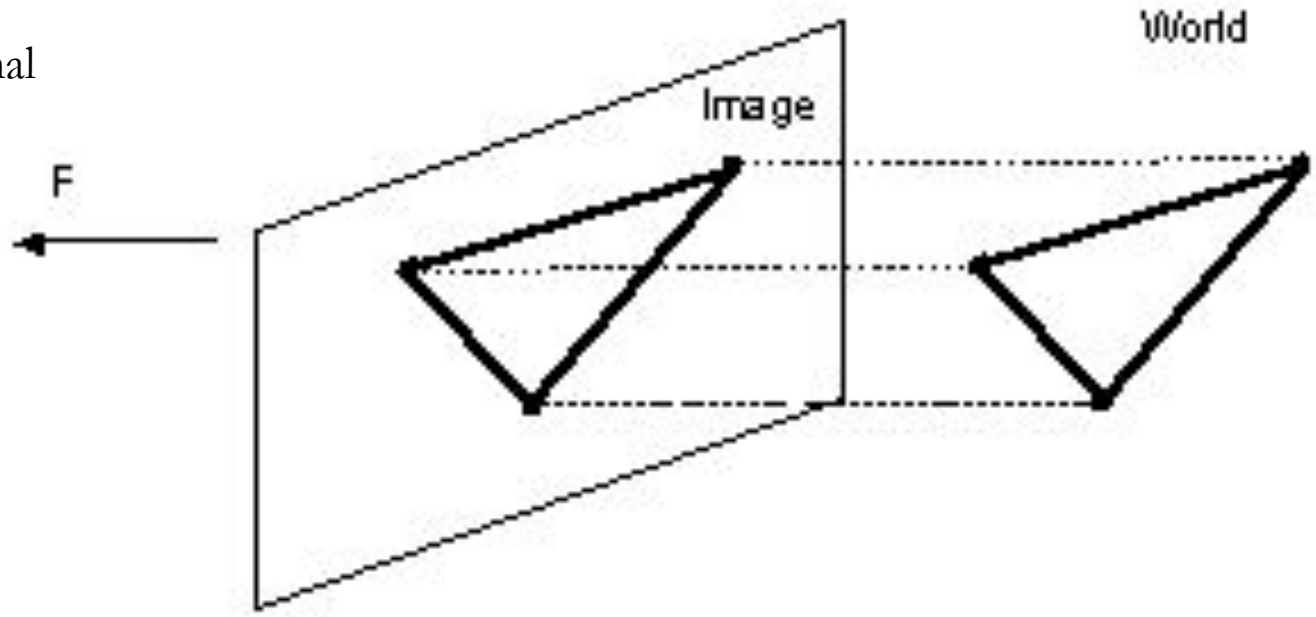
- Câmera na origem (apenas transformação de projeção)
  - Projeção ortográfica
  - Projeção perspectiva
- Câmera fora da origem (visualização + projeção)
  - Derivar uma matriz geral; ou
  - Transformar o mundo, levando a câmera para a origem

Projeção ortográfica (câmera na origem)

- Ponto focal no infinito, raios são paralelos e ortogonais ao plano de projeção
- Ótimo modelo para lentes de telescópio
- Mapeia (*x,y,z*) -> (*x,y,0*)

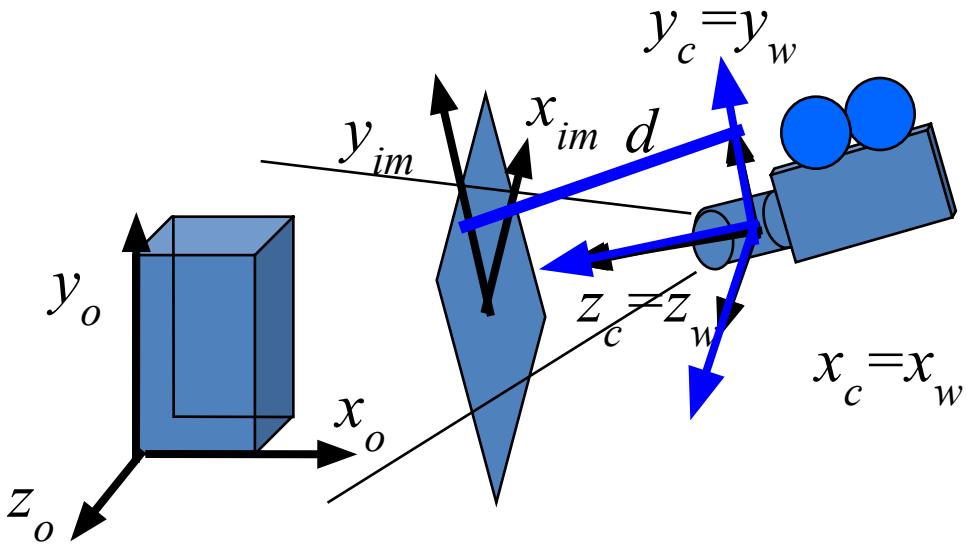
Matriz de projeção ortogonal

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

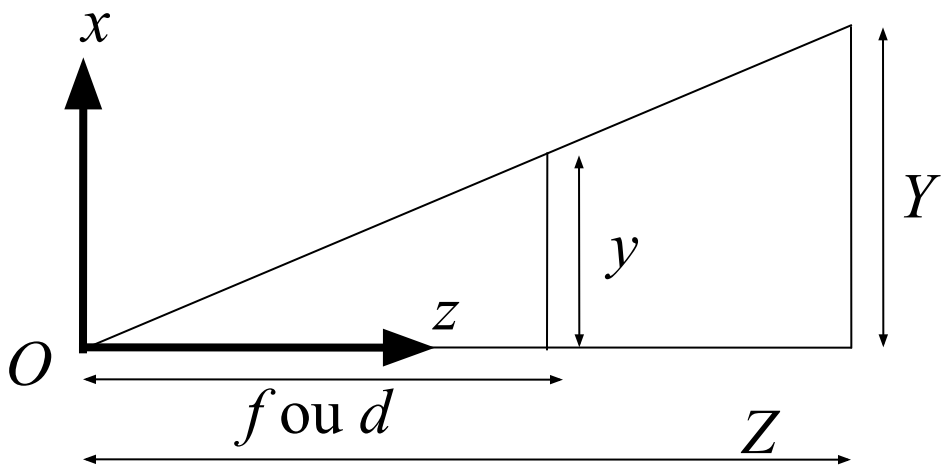


Perspectiva simples (câmera na origem)

- Caso canônico (câmera na origem)
  - Câmera olha ao longo do eixo *Z*
  - Ponto focal está na origem
  - Plano imagem paralelo ao plano *XY* a uma distância *d* (distância focal)



Equações perspectiva



$x = d(X/Z)$

$y = d(Y/Z)$

- Ponto *(X,Y,Z)* na cena projeta em *(d(X/Z),d(Y/Z),d)*
- Equações são não lineares devido à divisão

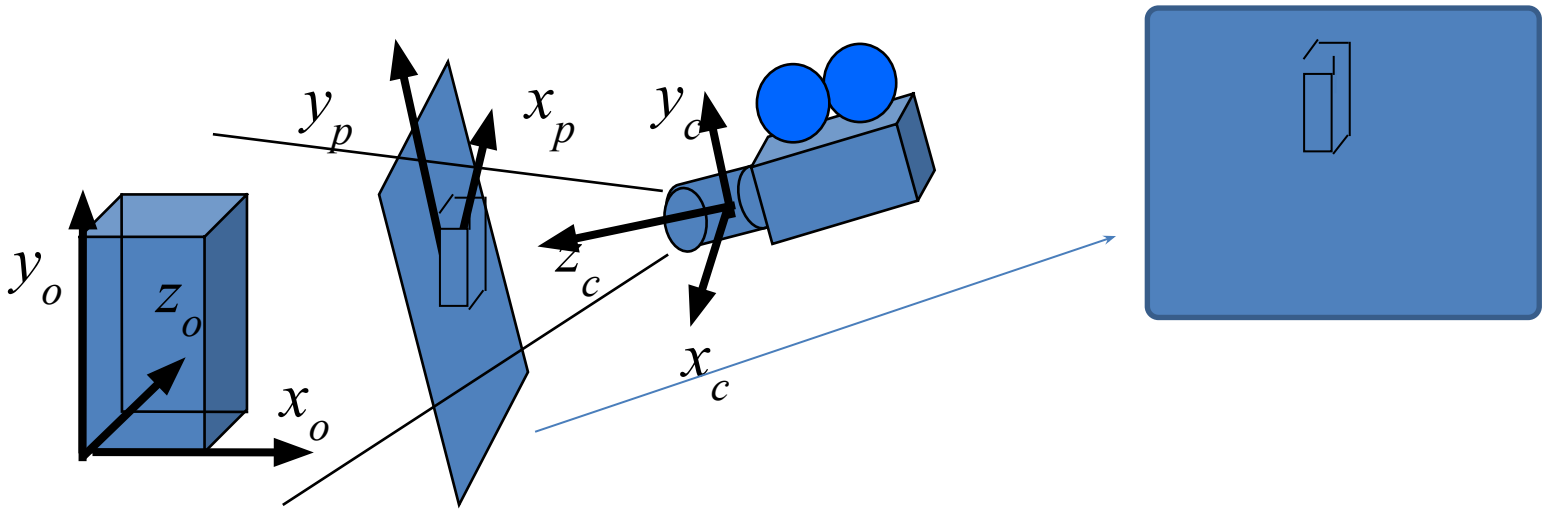
Matriz de projeção perspectiva

- Projeção usando coordenadas homogêneas
  - Transformar (***x,y,z***) em **[*(d(x/z),d(y/z),d***]

$$\begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} dx & dy & dz & z \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{d}{z}x & \frac{d}{z}y & \mathrm{d} \end{bmatrix}$$

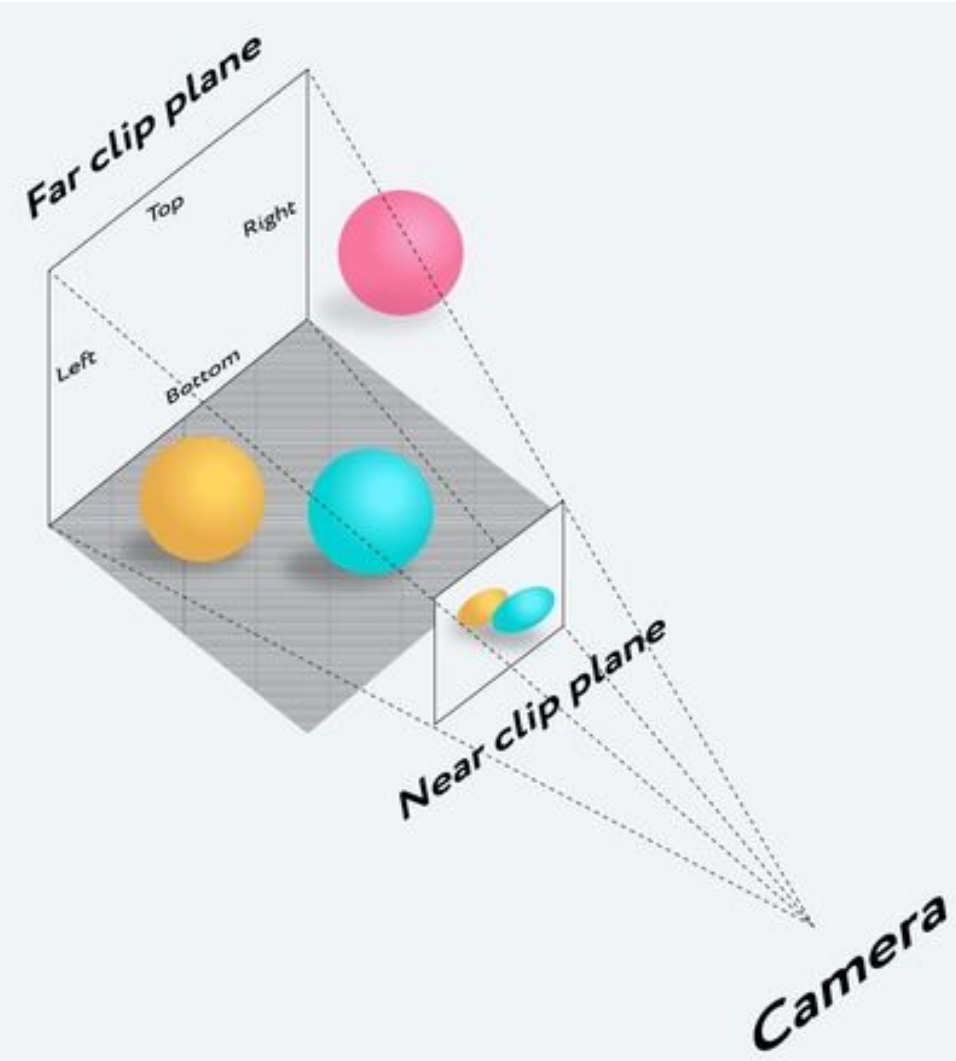
  - Divide pela 4ª coordenada (a coordenada “w”)
- Obter pontos de imagem 2D
  - Descarta a terceira coordenada e aplica transformação de viewport para obter coordenada de janela (pixel)

Transformação de projeção (resultado)

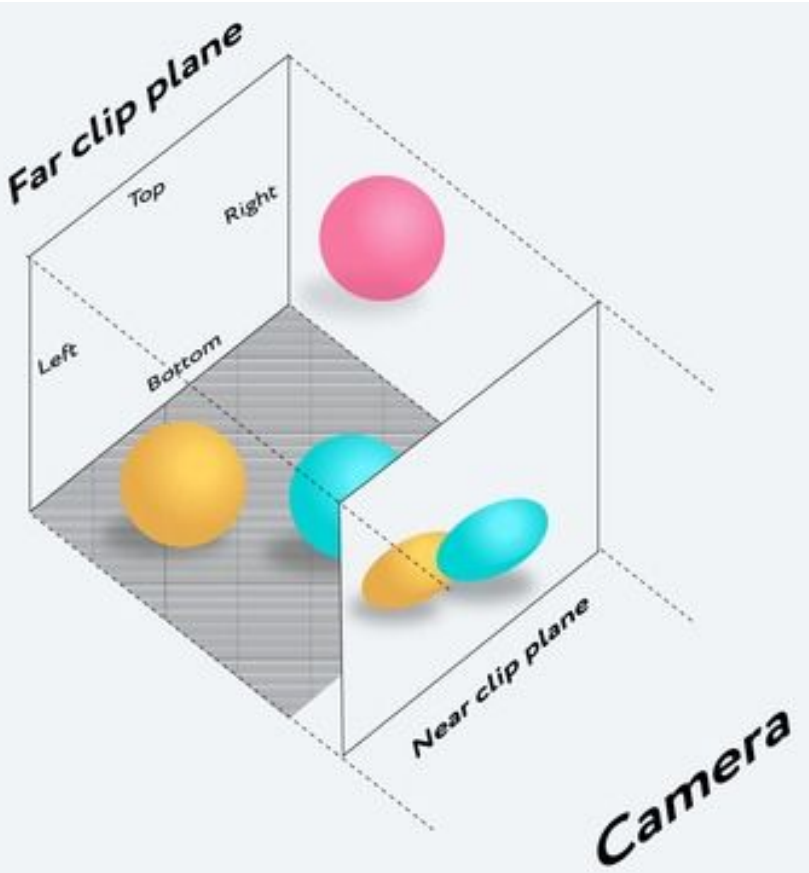


Projeção perspectiva (tem mais)

- Especificar planos ***near*** e ***far***
  - Ao invés de mapear *z* em ***d***, mapear ***z*** entre ***near*** e ***far*** (para o ***Z-Buffer***, visto adiante)
- Mapear imagem final projetada no portal de visualização
- Especificar campo de vista (ângulo fov)
- ***glViewPort, glFrustum*** e ***glPerspective*** fazem isso tudo



Perspective

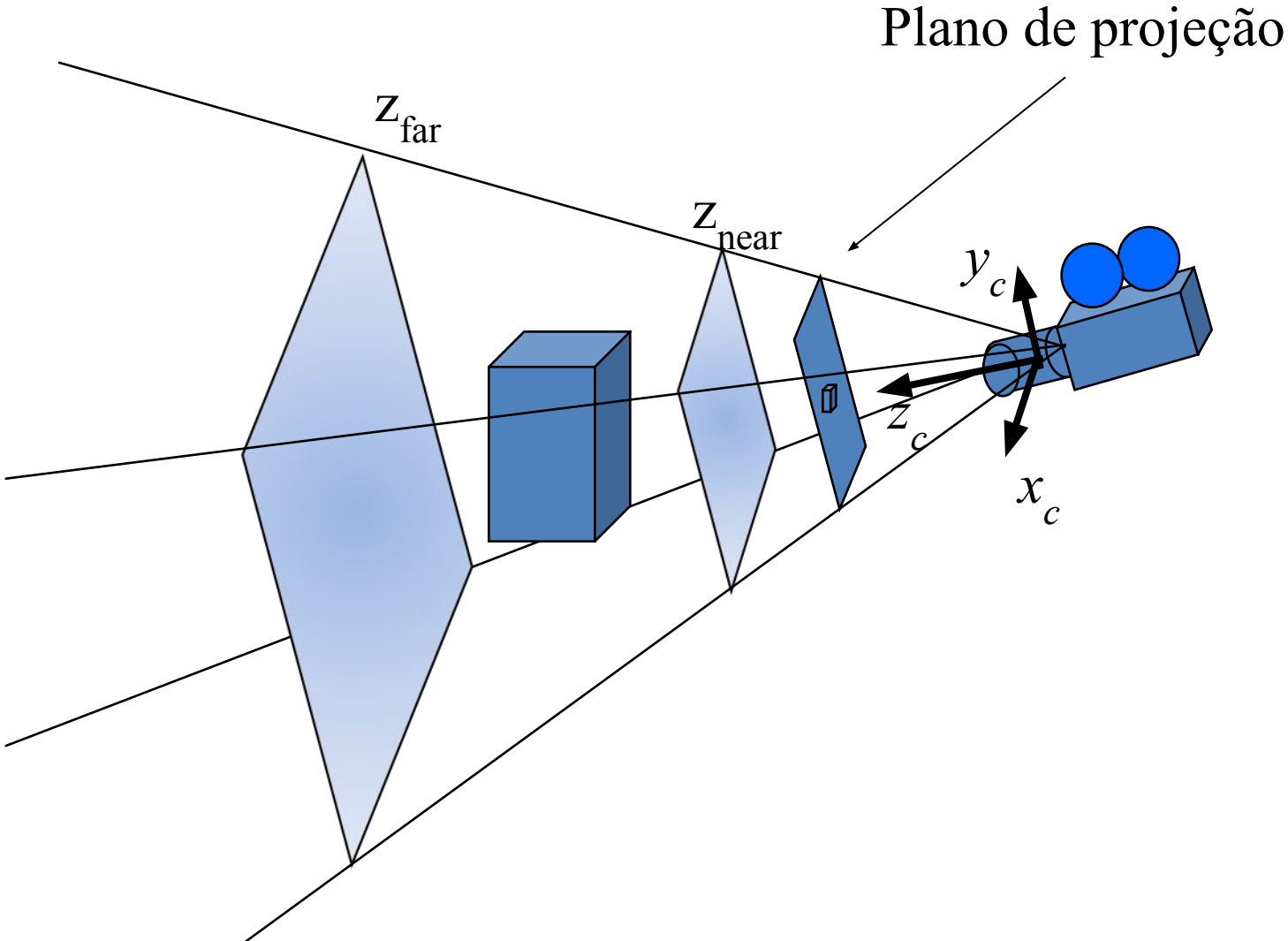


Orthographic

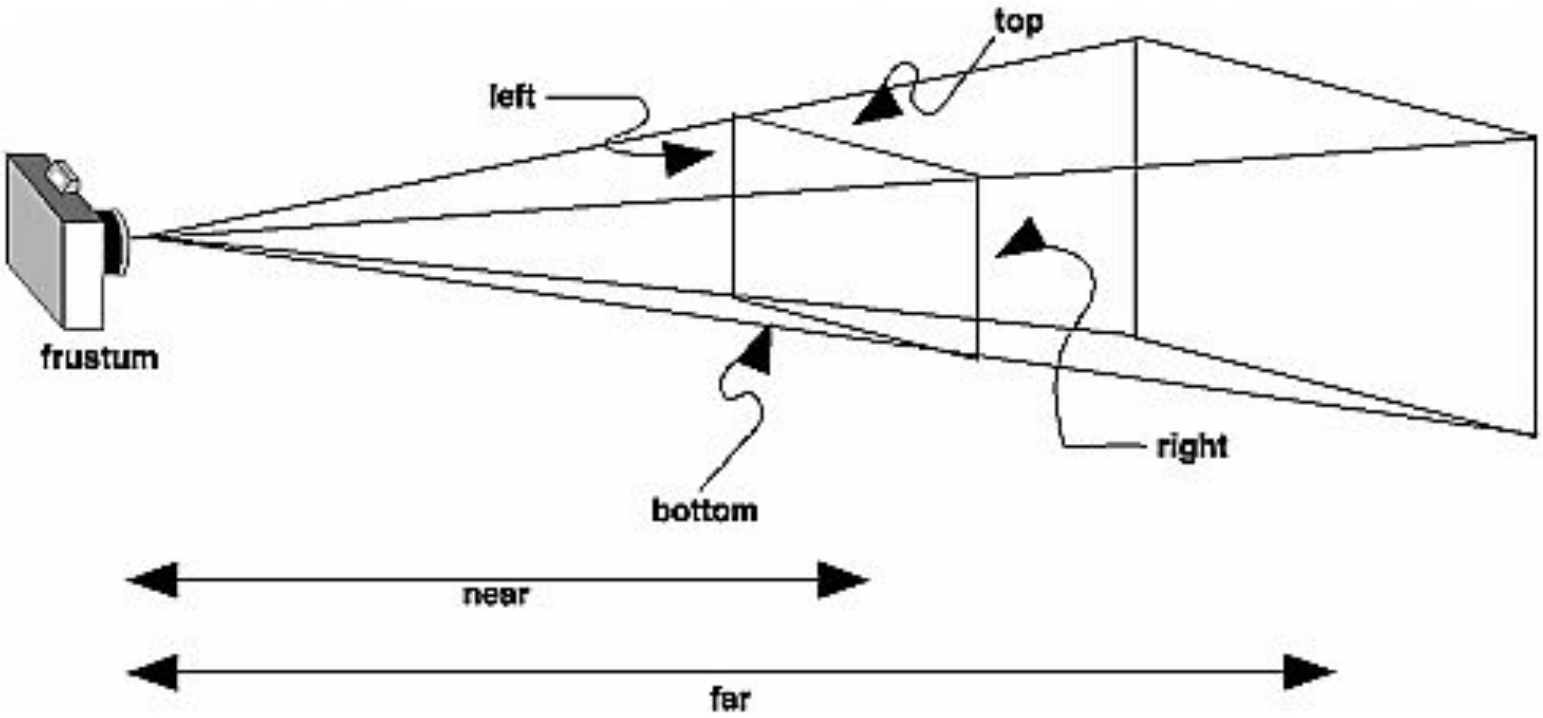
O volume de visualização

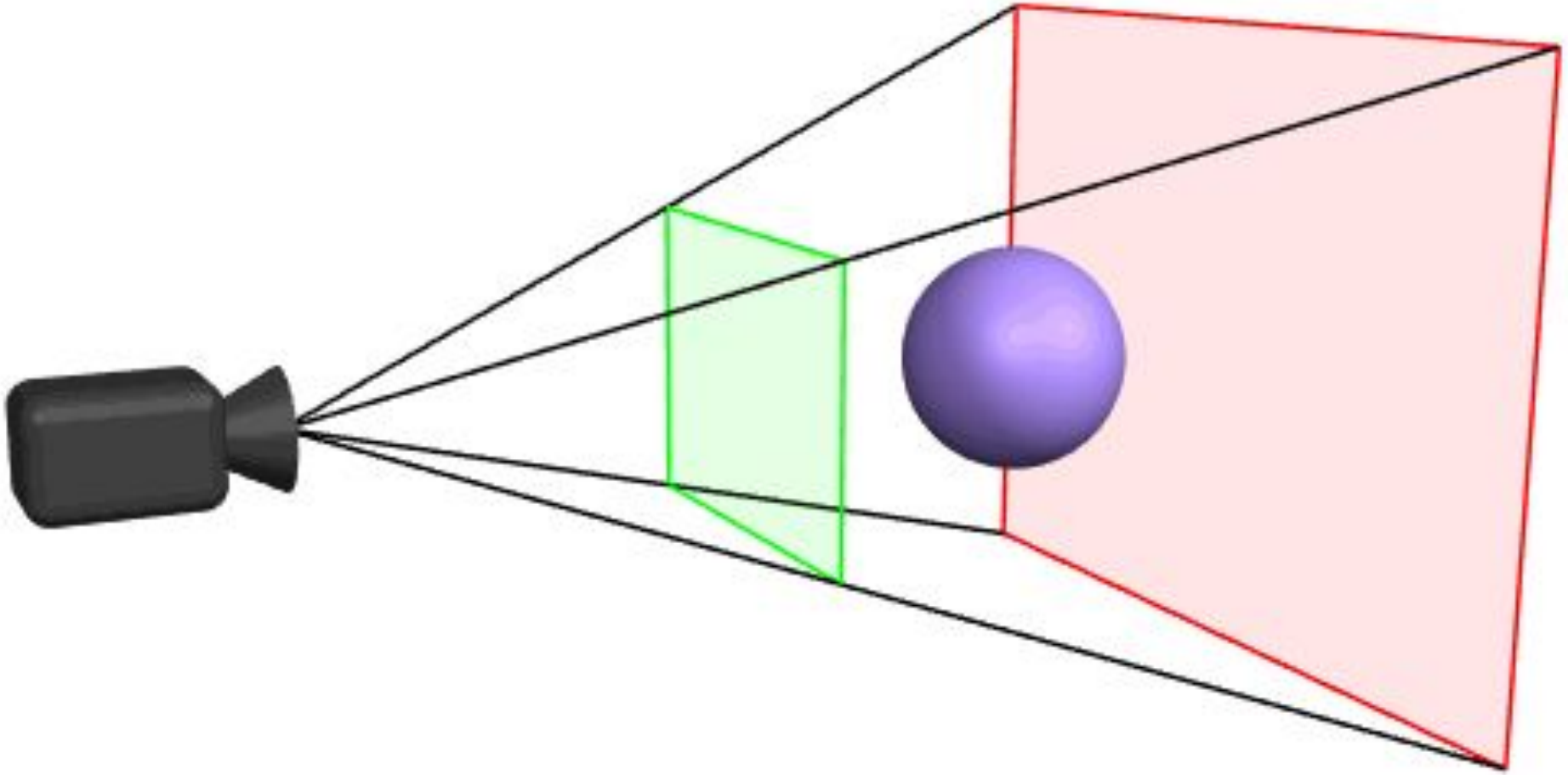
- Pirâmide no espaço definida pelo ponto focal e uma janela no plano imagem (janela mapeada no portal de visualização)
- Define a região visível do espaço
- Limites da pirâmide são os planos de corte
- Frustum = pirâmide truncada com planos de corte near e far
  - Near: previne pontos atrás da câmera de serem visualizados
  - Far: Permite escalar z num valor fixo (z-buffer)

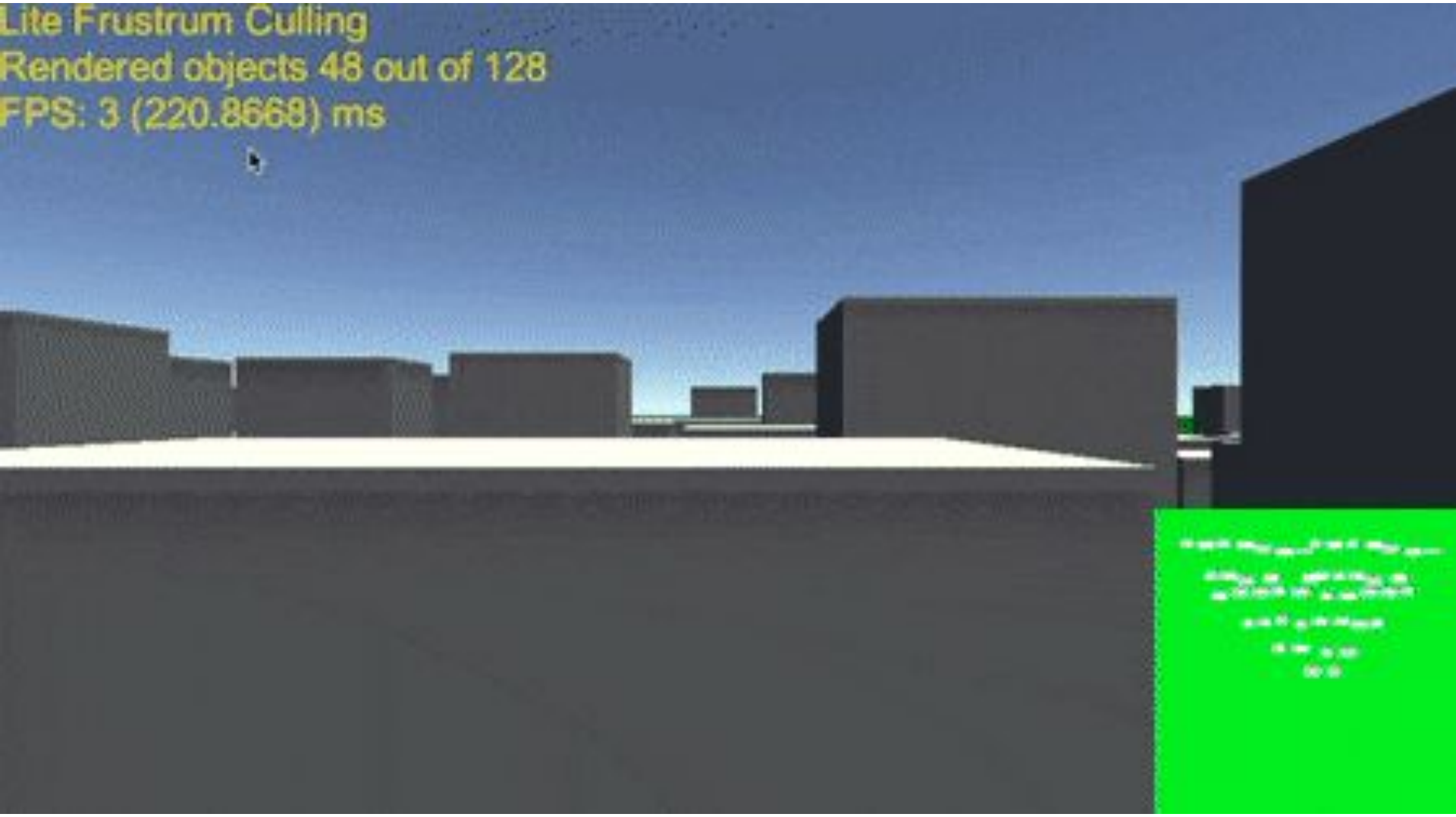
Frustum: pirâmide truncada visível



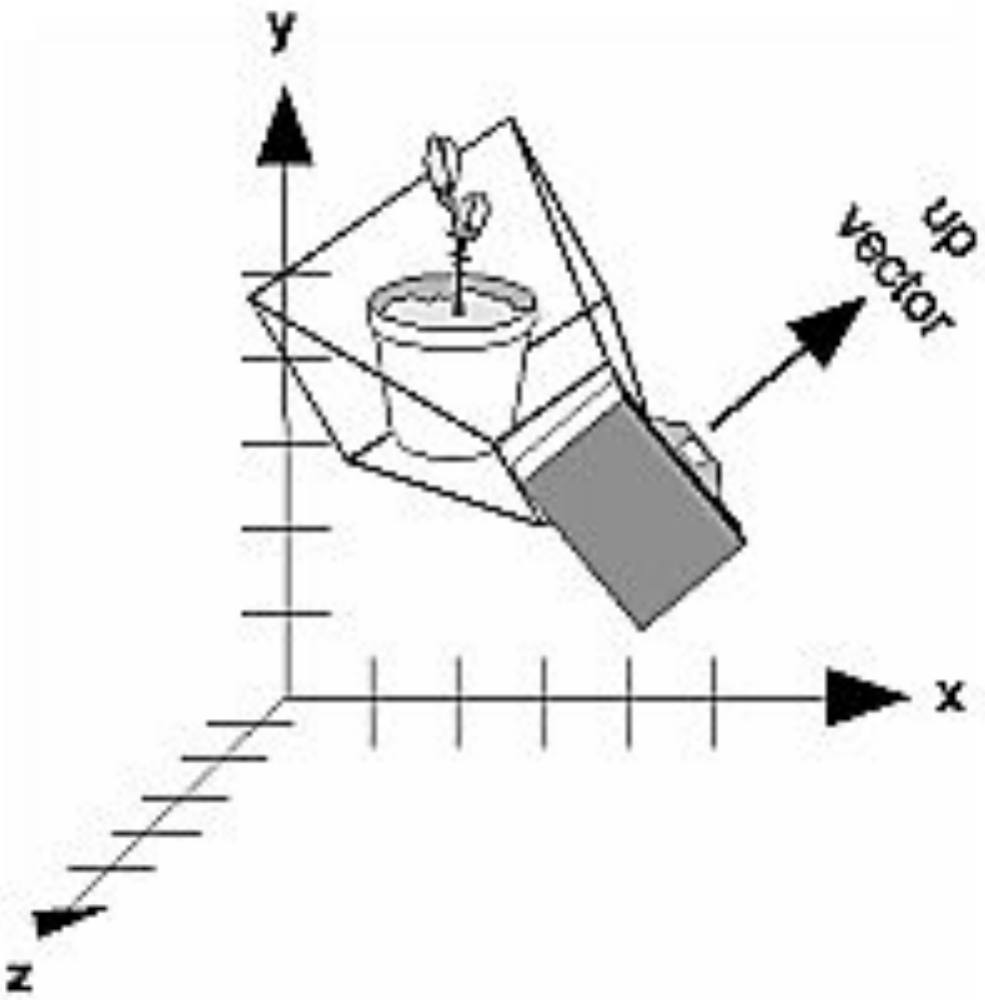
Frustum






Lite Frustrum Culling
Rendered objects 48 out of 128
FPS: 3 (220.8668) ms

Câmera fora da origem (caso geral)



Câmera fora da origem (caso geral)

- Duas alternativas:
  - Derivar uma matriz geral de projeção para transformar os objetos (trabalhoso); ou
  - Transformando o mundo, a câmera fica na posição e orientação canônica (mapeamento)
- São as transformações de visualização
- Podem ser especificadas de várias formas

Modelo mais popular para a transformação de visualização

- Distância focal, tamanho/forma da imagem e planos de corte definidos na transf. de projeção perspectiva
- Em adição, especifique:
  - *lookfrom* : onde está o ponto focal (câmera)
  - *lookat*: ponto no mundo centrado na imagem
- Especificar também orientação da câmera em torno do eixo *lookfrom-lookat*
  - *vup*: vetor no mundo indicando o “acima” da imagem (norte da câmera)

Implementação (3 passos)

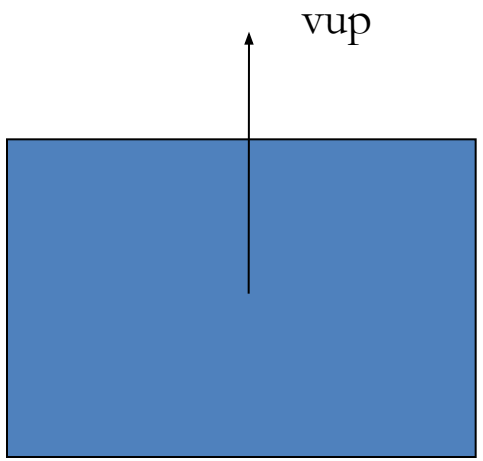
- 1) Translada de ***–lookfrom***, trazendo o ponto focal para a origem
- 2) Roda ***lookfrom-lookat*** p/ eixo *z* (mundo):
  - v=(lookat-lookfrom) normalizado e z=(0,0,1)
  - Eixo de rotação: a=(v *x* z)/|v *x* z|
  - Ângulo de rotação: cosθ=v.z e sinθ=|v *x* z|

$$\mathbf{R} = \mathbf{a}\mathbf{a}^T + (\mathbf{v}\cdot\mathbf{z})(\mathbf{I} - \mathbf{a}\mathbf{a}^T) + |\mathbf{v}\times\mathbf{z}|\mathbf{a}^* \quad \textbf{where} \quad \mathbf{a}^* = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

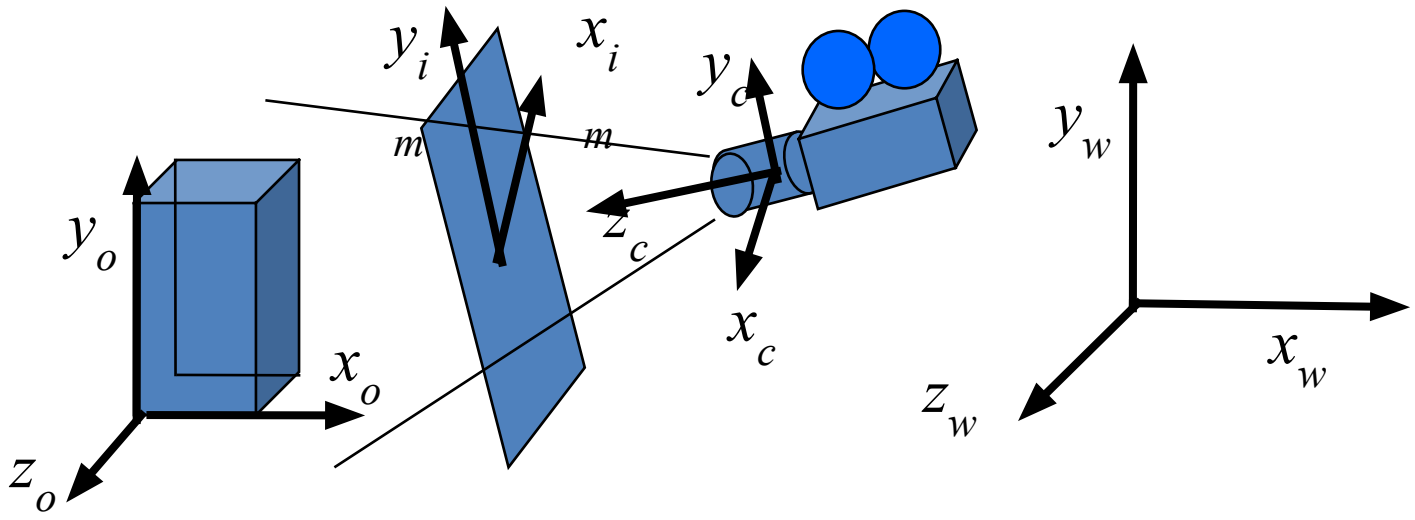
  - Ou então, mais fácil, glRotate(θ,a_x,a_y,a_z)

Implementação

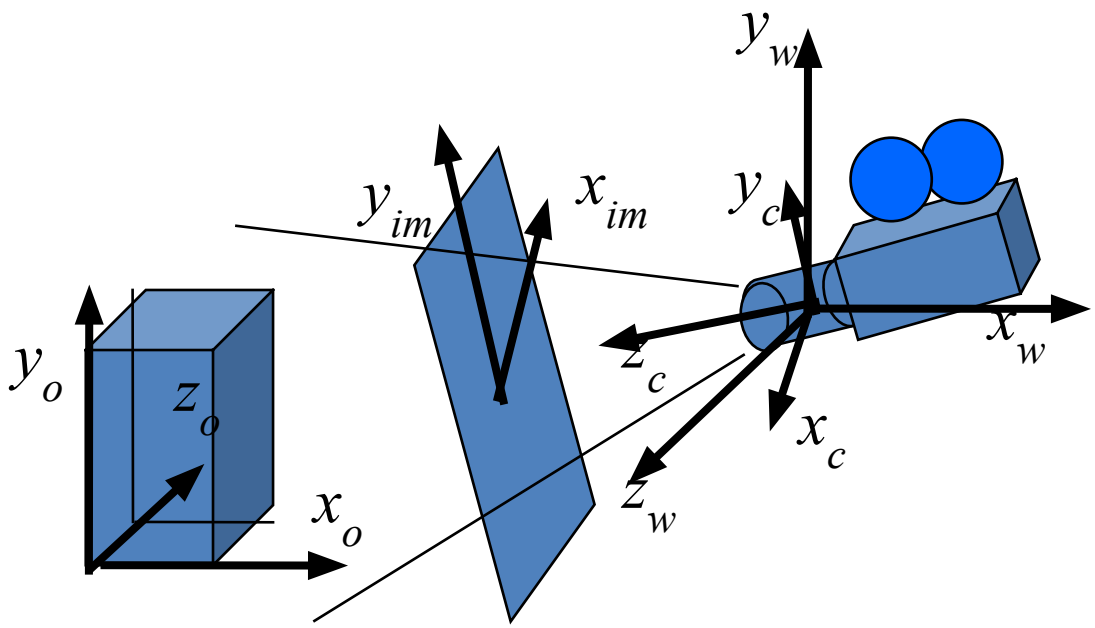
- 3) Roda em torno do eixo z para fazer vup ficar paralelo ao eixo y



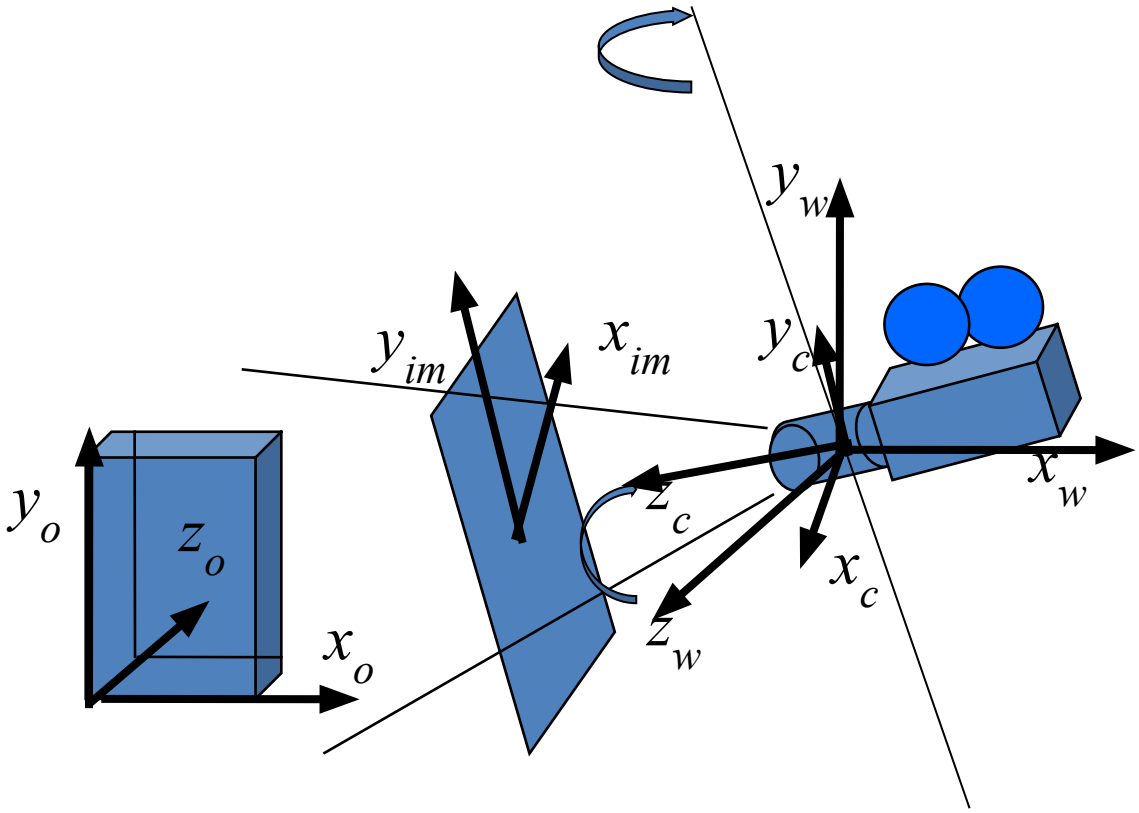
Translada de -lookfrom



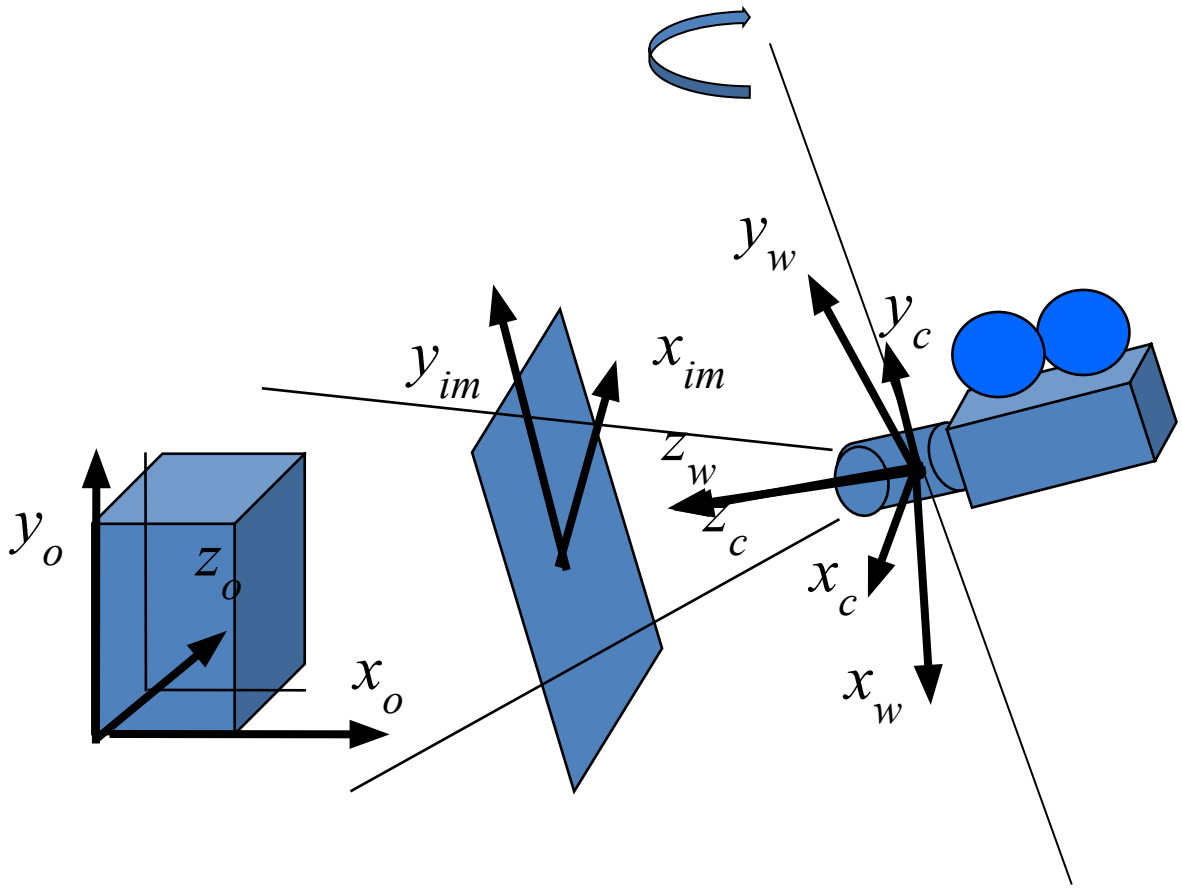
Rotate Z axis



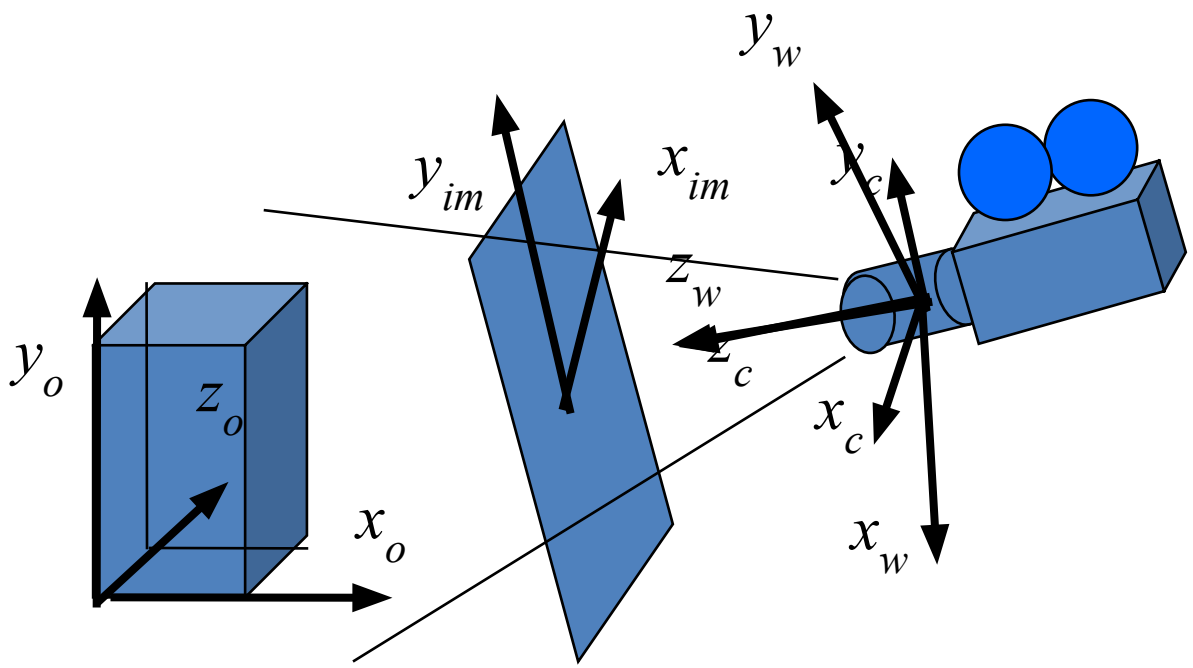
Rotate Z axis



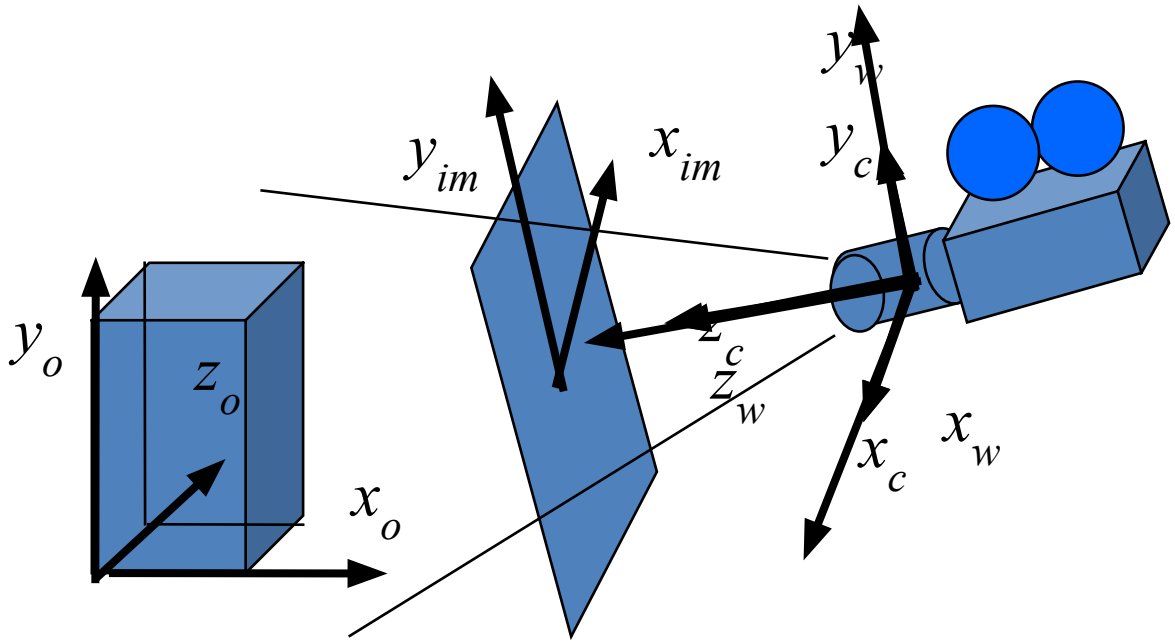
Rotate Z axis



Rotate Y axis

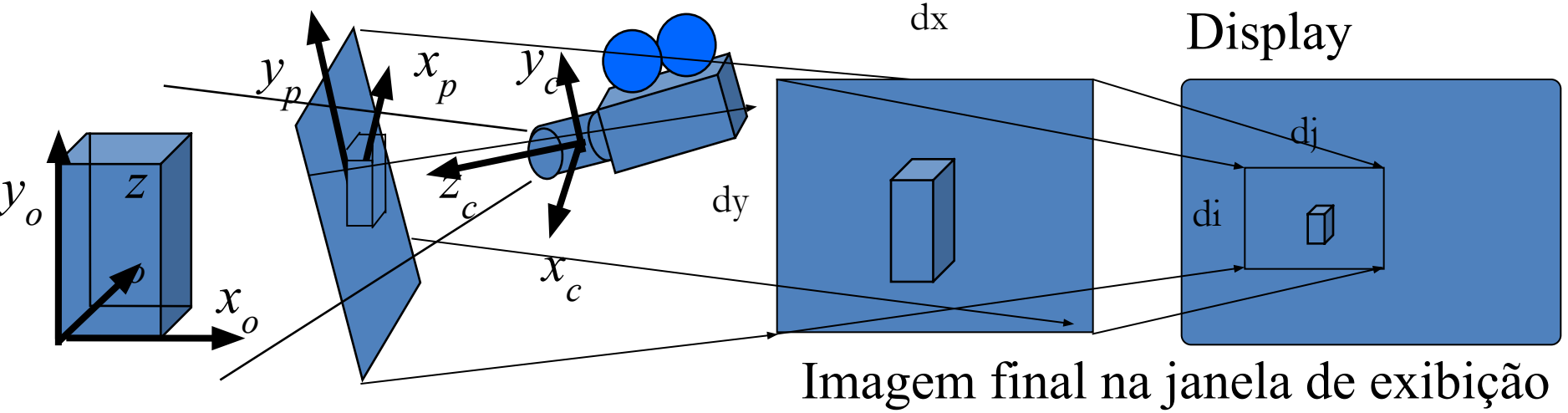


That is it



View-port (portal de visualização)

- Transformação de escala em ambos os eixos
- Mapear dx, dy (dimensões do plano de projeção) em di, dj, (dimensões da janela de visualização)



Transformação de view-port

- Devemos mapear as coordenadas (x_c, y_c) do ponto (sistema de câmera) com (x_{im}, y_{im}), suas coordenadas em pixels (sistema de imagem)
- As seguintes equações podem ser usadas:

$$x_c = -(x_{im} - o_x)s_x = (o_x - x_{im})s_x$$
$$y_c = -(y_{im} - o_y)s_y = (o_y - y_{im})s_y$$

sendo (o_x, o_y) as coordenadas em pixel do centro da imagem (ponto principal) e (s_x, s_y) o tamanho efetivo do pixel (em milímetros) horizontal e verticalmente, respectivamente

Portal de Visualização e Frustum

- Idealmente, o portal de visualização e o plano de projeção (definido pelos limites do frustum) devem guardar a mesma razão entre suas dimensões, para não gerar aberrações
- Bom, não precisa se preocupar muito, porque…

Felizmente

- OpenGL faz tudo para você. Especifique:
  - Onde você está (*lookfrom*)
  - Onde você olha (*lookat*)
  - Orientação da câmera (*vup*)
  - Campo de vista (ângulo *fov*)
  - Distância focal (*d*)
  - Dimensões da janela de visualização (pixels)
  - Razão de aspecto (Dy/Dx)
- E é isto! Done!

Implementação em OpenGL