

Viewing 3D

SCC0250 - Computação Gráfica

Prof. Fernando V. Paulovich
<http://www.icmc.usp.br/~paulovic>
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

27 de junho de 2010



Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

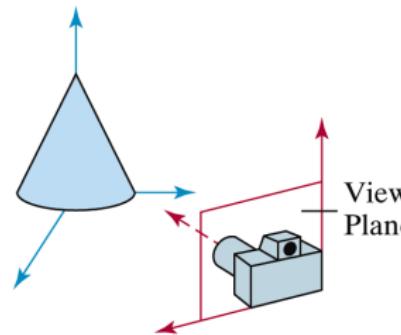
Visão Geral

Viewing Pipeline 3D

- As funções de *viewing* processam a descrição de objetos por meio de um conjunto de procedimentos a fim de projetar uma visão específica desses na superfície do dispositivo de saída
- Alguns desses procedimentos são similares aos do *Viewing Pipeline 2D*
 - Rotinas de recorte
- Mas outros são específicos do 3D
 - Rotinas de projeção
 - Identificação de partes visíveis da cena
 - Efeitos de luz

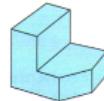
Visualizando uma Cena 3D

- Para se obter uma visão de uma cena 3D descrita por coordenadas-do-mundo, primeiro é necessário definir um sistema de referência para os parâmetros de visão (ou câmera)
 - Define a posição e orientação do **plano de visão** (ou **plano de projeção**) – plano do filme da câmera



Projeções

- É possível escolher diferentes métodos para projetar uma cena no plano de visão
 - **Projeção Paralela:** projeta os pontos de um objeto ao longo de linhas paralelas – usado para desenhos arquitetônicos e de engenharia
 - **Projeção Perspectiva:** projeta os pontos de um objeto ao longo de caminhos convergentes – cenas mais realísticas (objetos longe da posição de visão são mostrados menores)



(a) Projeção Paralela



(b) Projeção Perspectiva

Depth Cueing (Profundidade)

- Com raras exceções, informação de profundidade é importante para a composição de uma cena 3D: identificar a parte da frente e de trás dos objetos

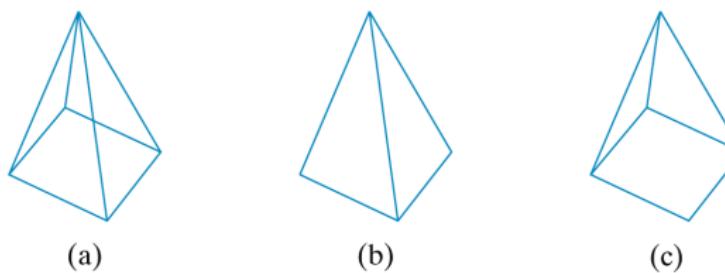
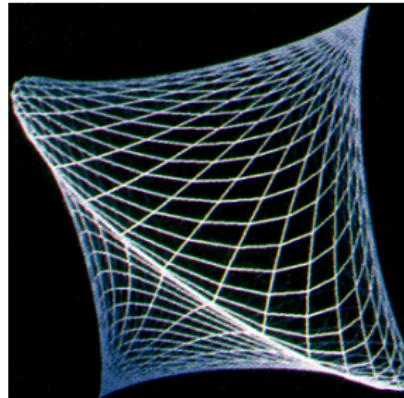


Figura: Problemas de ambiguidade pela falta de informação de profundidade em (a), que pode ser interpretada como (b) ou (c)

Depth Cueing (Profundidade)

Depth Cueing

- Forma simples de indicar profundidade de objetos aramados (wire-frame) variando o brilho das linhas
 - Linhas mais próximas da posição de visão são mais brilhantes
- Também serve para modelar outros efeitos (p.ex.: neblina, pó, fumaça, etc.) – objetos distantes mais escuros



Identificando Linhas e Superfícies Visíveis

Cenas Wire-Frame

- Existem outras formas de tornar visualizações de *wire-frame* mais realísticas
 - Colorir as linhas visíveis de forma diferente das não-visíveis
 - Mostrar as linhas não visíveis como linhas pontilhadas
 - Remover as linhas não visíveis – também remove informação sobre a forma do objeto

Cenas Realísticas

- Para cenas realísticas, as partes não visíveis de um objeto são completamente eliminadas, e somente as visíveis são mostradas
 - Os pixels da tela terão informação apenas das cores das superfícies da frente

Rendering de Superfície

- Efeitos realísticos são alcançados usando efeitos de iluminação
 - Define-se a luz ambiente
 - Especifica-se a localização e cor das diferentes fontes de luz
- As características dos materiais são também especificadas
 - Transparente, opaco, rugoso, etc.



Sumário

1 Introdução

2 Viewing Pipeline 3D

3 Parâmetros de Coordenadas de Visão 3D

4 Transformação do Sistema de Coordenadas de Mundo para o de Visão

5 Transformações de Projeção

- Projeções Ortogonais
- Projeções Paralelas Oblíquas
- Projeções Perspectivas
- Transformação de Viewport e Coordenadas de Tela 3D

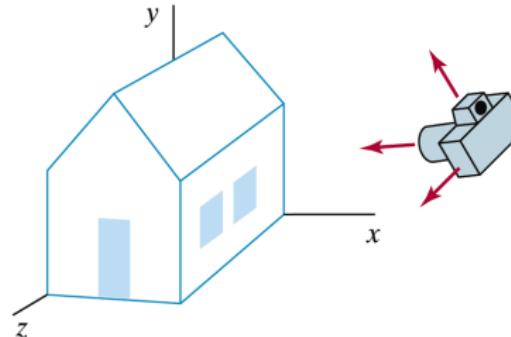
6 Funções OpenGL para Viewing 3D

- Transformação de Visão OpenGL
- Função de Projeção Ortogonal OpenGL
- Projeção Perspectiva Simétrica OpenGL
- Projeção Perspectiva Genérica OpenGL

7 Algoritmos de Recorte 3D

Viewing Pipeline 3D

- O processo para se criar uma imagem de computação gráfica de uma cena 3D é parecida com se tirar uma foto
 - É necessário escolher a posição de visão, onde será posta a câmera
 - É preciso definir a orientação da câmera
 - Como apontar a câmera a partir da posição de visão
 - Como a câmera vai estar rotacionada, definindo a posição *up*

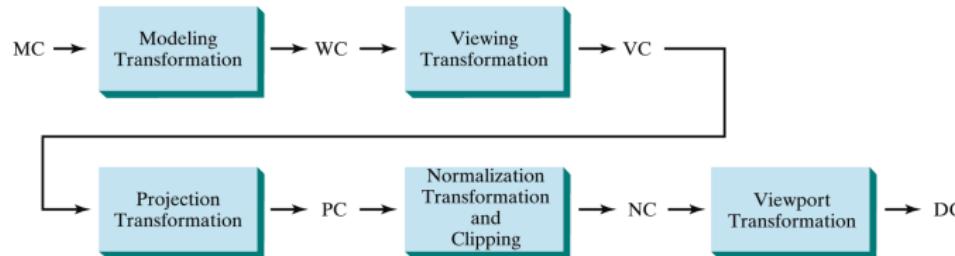


Viewing Pipeline 3D

- Algumas das operações do *Viewing Pipeline 3D* são semelhantes ao 2D
 - Uma *viewport* 2D é usada para posicionar a visão projetada no dispositivo de saída
 - Uma janela de recorte 2D é usada para selecionar uma visão que será mapeada na *viewport*
 - Uma janela de saída é definida em coordenadas da tela
- Porém, outras são diferentes
 - A janela de recorte é posicionada sobre um plano de visão, e a cena é recortada considerando um volume (**volume de recorte**) usando planos de recorte

Viewing Pipeline 3D

- A posição de visão, o plano de visão, a janela de recorte e os planos de recorte são especificados dentro do sistema de coordenadas de visão



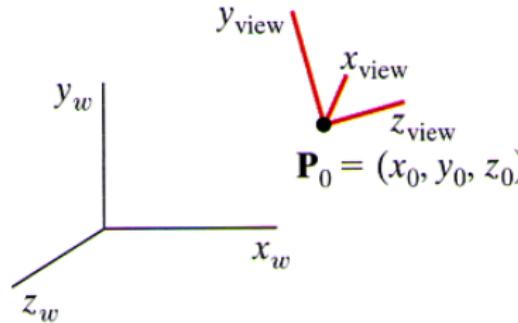
- O sistema de coordenadas de visão define os parâmetros de visão (posição e orientação do plano de projeção), o **volume de visão** e a janela de recorte

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

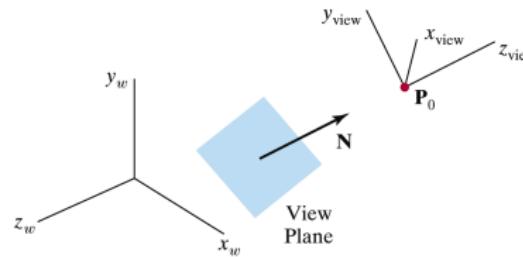
Parâmetros de Coordenadas de Visão 3D

- Para estabelecer um sistema de coordenadas de visão 3D é preciso definir
 - A origem do sistema $\mathbf{P}_0 = (x_0, y_0, z_0)$, chamada de **ponto de visão** (também referido como posição do olho ou da câmera)
 - O **vetor view-up \mathbf{V}** , que define a direção y_{view}
 - Uma segunda direção para um dos eixos coordenados restantes, normalmente o z_{view} , com a direção de visão ao longo desse eixo



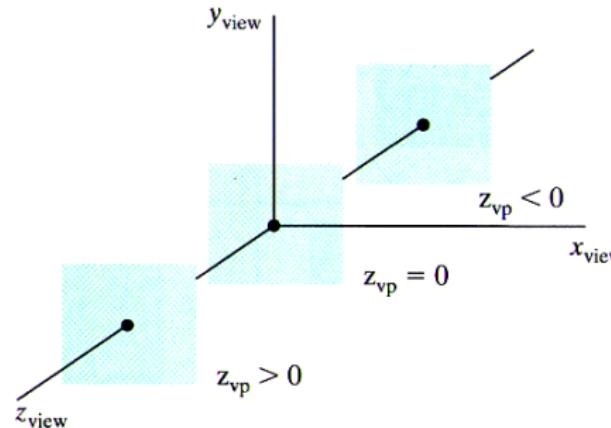
Vetor Normal ao Plano de Visão

- Como a direção de visão em geral é definida sobre o eixo z_{view} , o plano de projeção é normalmente assumido ser perpendicular a esse eixo
 - Assim, a orientação do plano de projeção e a direção positiva do eixo z_{view} podem ser definidas com um **vetor normal N ao plano de projeção**



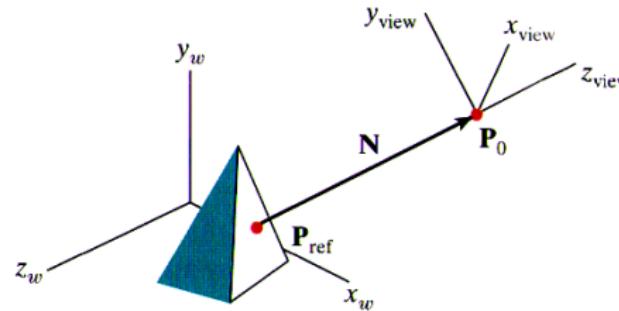
Vetor Normal ao Plano de Visão

- Uma valor escalar é usado para definir a posição do plano de projeção em alguma coordenada z_{vp} ao longo do eixo z_{view}
 - Especifica a distância da origem da visão ao longo da direção de visão, normalmente na direção negativa de z_{view}
 - Portanto, o plano de projeção é sempre paralelo ao plano $x_{view}y_{view}$



Vetor Normal ao Plano de Visão

- O vetor normal \mathbf{N} pode ser especificado de várias formas
 - A direção de \mathbf{N} pode ser definida ao longo da linha partindo de um ponto de referência \mathbf{P}_{ref} até a origem de visão \mathbf{P}_0
 - Nessa caso, esse ponto de referência é denominado **look-at point**, com a direção de visão oposta a de \mathbf{N}



O Vetor View-Up

- Uma vez estabelecida a direção normal ao plano de projeção \mathbf{N} , podemos estabelecer o **vetor view-up** \mathbf{V} que dá a direção do eixo y_{view}
- Usualmente \mathbf{V} é definido selecionando uma posição relativa a origem do sistema de coordenadas do mundo

O Vetor View-Up

- \mathbf{V} pode ser definido em qualquer direção, desde que não paralela a \mathbf{N}
 - Uma forma conveniente seria definir uma direção paralela ao eixo y_w , $\mathbf{V} = (0, 1, 0)$
 - Se esse não for exatamente perpendicular a \mathbf{N} as rotinas de visão podem ajustá-lo projetando-o em um plano perpendicular a \mathbf{N}

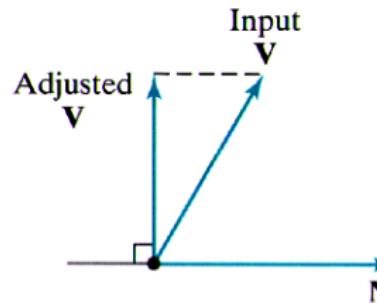


Figura: Ajuste do vetor *view-up* \mathbf{V} para torná-lo perpendicular a \mathbf{N}

Sistema de Coordenadas de Visão uvn

Sistema de Coordenadas de Visão uvn

- Como a normal **N** define a orientação z_{view} , e o vetor *view-up* **V** é usado para definir y_{view} , só é necessário definir a direção positiva de x_{view}
 - Essa direção é representada pelo vetor **U**, calculada como o produto vetorial de **N** e **V**
 - O produto vetorial de **N** e **U** também pode ser usado para ajustar o valor de **V** ao longo do eixo y_{view}

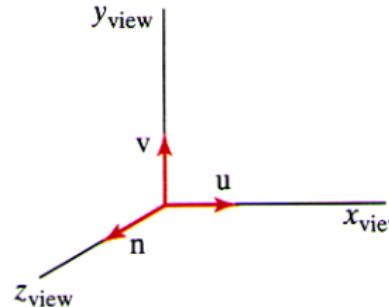
Sistema de Coordenadas de Visão uvn

- Para se obter o sistema de coordenadas **uvn** fazemos

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z)$$

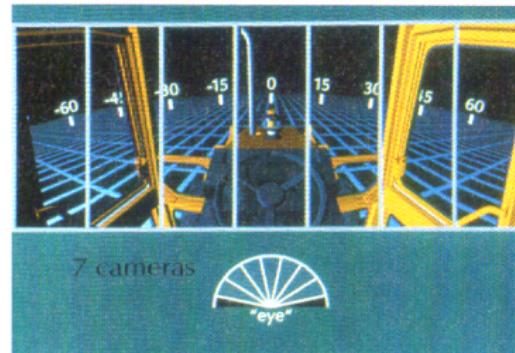
$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V} \times \mathbf{n}|} = (u_x, u_y, u_z)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z)$$



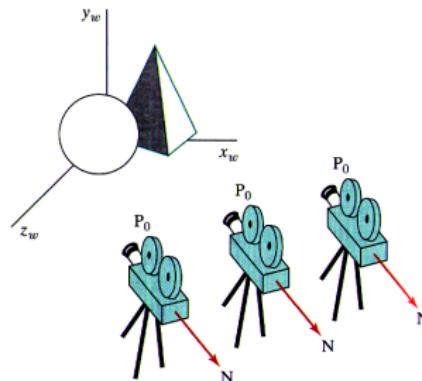
Gerando Efeitos de Visão 3D

- Variando-se os parâmetros de visão é possível obter diferentes efeitos
 - De uma posição de visão fixa é possível mudar N para mostrar objetos em posições ao redor da origem
 - Variar N para criar uma cena composta de múltiplas visões de uma posição fixa da câmera
 - Quando alterar N não esqueça de mudar os outros eixos para se manter o sistema orientado pela mão direita

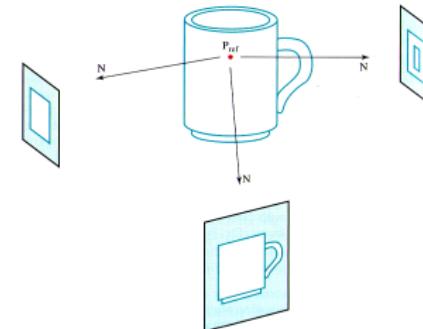


Gerando Efeitos de Visão 3D

- Efeitos de mover a câmera (pam) podem ser obtidos fixando \mathbf{N} e modificando o ponto de visão
- Para se mostrar diferentes visões de uma objeto (visão lateral, frontal, etc.) podemos mover o ponto de visão ao redor do objeto



(a) Efeito de pam



(b) Visões diferentes

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 5 Transformações de Projeção
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Transformação do Sistema de Coordenadas de Mundo para o de Visão

- No *Viewing Pipeline 3D*, o primeiro passo a ser executado após a cena ser construída é transferir a descrição dos objetos ao sistema de coordenadas de visão
 - Essa conversão é equivalente a sobrepor o sistema de referência de visão sobre o sistema de referência de mundo
- Esse mapeamento pode ser feito
 - ① Transladando a origem do sistema de coordenadas de visão para a origem do sistema de coordenadas de mundo
 - ② Rotacionando para alinhar os eixos x_{view} , y_{view} e z_{view} com os eixos de mundo x_w , y_w e z_w

Transformação do Sistema de Coordenadas de Mundo para o de Visão

- Se a origem do sistema de visão for $\mathbf{P}_0 = (x_0, y_0, z_0)$ a matriz de translação será

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação do Sistema de Coordenadas de Mundo para o de Visão

- A matriz de rotação pode ser obtida direto dos vetores $\mathbf{u} = (u_x, u_y, u_z)$, $\mathbf{v} = (v_x, v_y, v_z)$ e $\mathbf{n} = (n_x, n_y, n_z)$

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação do Sistema de Coordenadas de Mundo para o de Visão

- A matriz de transformação é portanto

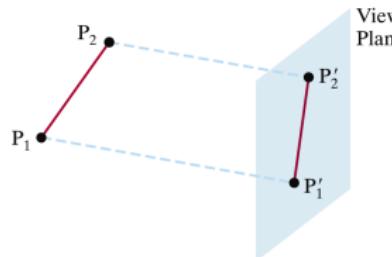
$$\begin{aligned}\mathbf{M}_{WC,VC} &= \mathbf{R} \cdot \mathbf{T} \\ &= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

Sumário

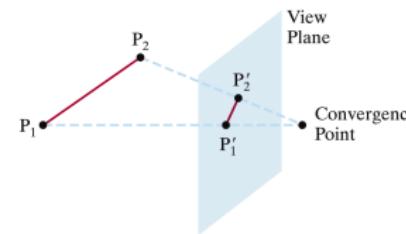
- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Transformações de Projeção

- Após a transformação para a coordenadas de visão, o próximo passo do *Viewing Pipeline 3D* é a projeção no plano de projeção
- Em geral os pacotes gráficos suportam projeção paralela e perspectiva
 - **Projeção Paralela:** as coordenadas são transferidas para o plano de projeção ao longo de linhas paralelas
 - **Projeção Perspectiva:** as coordenadas são transferidas para o plano de projeção ao longo de linhas que convergem a um ponto atrás desse



(c) Projeção Paralela



(d) Projeção Perspectiva

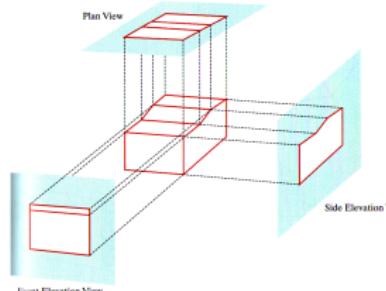
Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 **Transformações de Projeção**
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Projeções Ortogonais

Projeção Ortogonal (ou Ortográfica)

- Transformação da descrição dos objetos a um plano de projeção ao longo de linhas paralelas ao vetor normal \mathbf{N}
- Frequentemente usada para produzir a visão frontal, lateral e superior de um objeto
- Preserva tamanhos e ângulos: útil para desenhos arquitetônicos e de engenharia



Projeções Ortogonais Axonométricas e Isométricas

Projeção Ortogonal Axonométrica

- Projeção ortogonal que mostra mais de uma face de um objeto
- A projeção axonométrica mais comum é a **isométrica**
 - O plano de projeção é alinhado de forma a intersectar cada eixo coordenado no qual o objeto é definido a mesma distância da origem

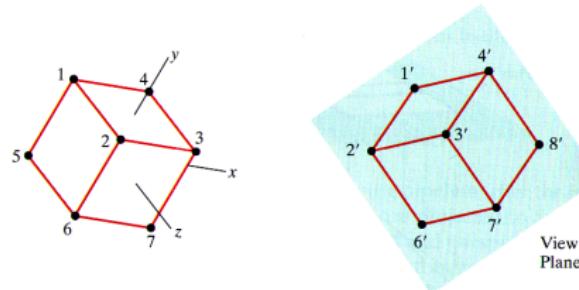


Figura: Essa projeção é obtida alinhando o vetor **N** com a diagonal do cubo

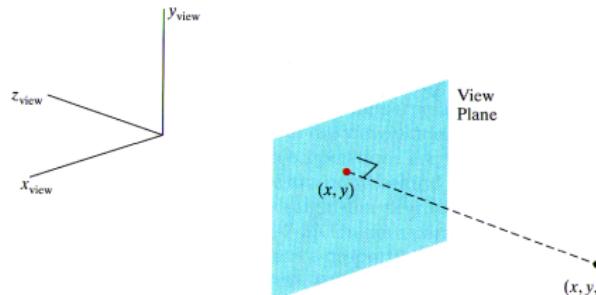
Coordenadas de Projeções Ortogonais

- Com a direção de projeção paralela a z_{view} , as equações para a transformação de projeção ortogonal de um posição (x, y, z) são

$$x_p = x$$

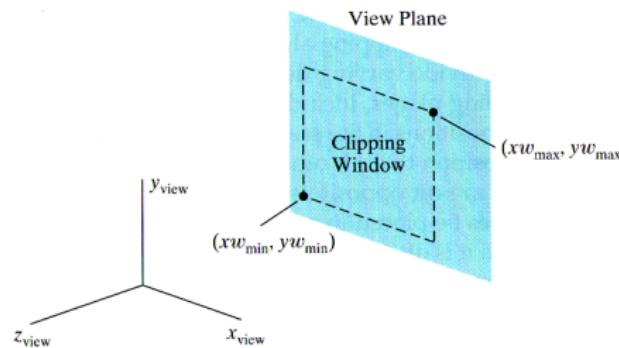
$$y_p = y$$

- O valor de z é armazenado para uso futuro nos procedimentos para determinar visibilidade



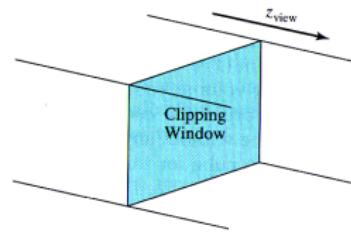
Janela de Recorte e Volume de Projeção Ortogonal

- Para se determinar o quanto de uma cena 3D será transferida para o plano de projeção, uma **Janela de Recorte** é utilizada
 - É necessário determinar os limites dessa janela sobre o plano de projeção com as arestas paralelas aos eixos x_{view} e y_{view}

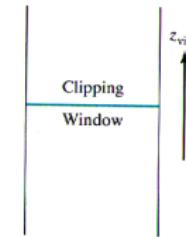


Janela de Recorte e Volume de Projeção Ortogonal

- As arestas de *Janela de Recorte* especificam os limites x e y da parte da cena que será mostrada, formando o **volume de visão de projeção ortogonal**



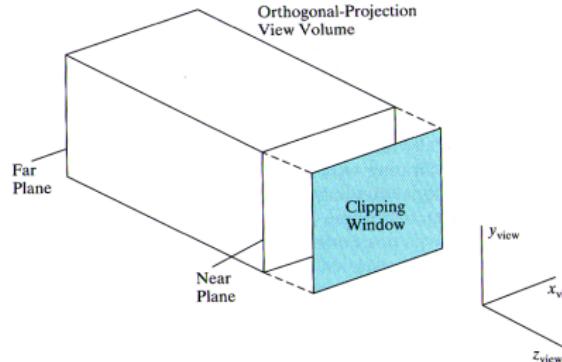
Side View
(a)



Top View
(b)

Janela de Recorte e Volume de Projeção Ortogonal

- Para se limitar a extensão desse volume na direção z_{view} dois planos de fronteira, chamados **planos de recorte near/far**, são considerados paralelos aos planos de visão
 - Permite eliminar objetos que estão na frente ou atrás de uma parte da cena
 - Com a direção de visão ao longo do eixo negativo z_{view} , temos $z_{far} < z_{near}$



Transformação de Normalização para Projeção Ortogonal

- Como qualquer posição (x, y, z) em uma projeção ortogonal é mapeada para (x, y) , as coordenadas dentro do volume de visão são as coordenadas de projeção, assim essas podem ser mapeadas para um **volume de visão normalizado** sem precisar ser reprojetadas

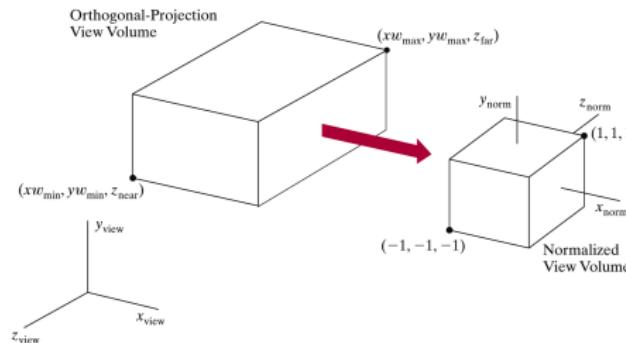


Figura: Transformação de normalização em um sistema de referência dado pela mão esquerda (normalmente o sistema da tela)

Transformação de Normalização para Projeção Ortogonal

- Essa transformação de normalização é semelhante à obtida em 2D, com a adição dos valores da coordenada z sendo normalizados do intervalo z_{near} a z_{far} para -1 a 1

$\mathbf{M}_{ortho,norm} =$

$$\begin{bmatrix} \frac{2}{xw_{max}-xw_{min}} & 0 & 0 & -\frac{xw_{max}+xw_{min}}{xw_{max}-xw_{min}} \\ 0 & \frac{2}{yw_{max}-yw_{min}} & 0 & -\frac{yw_{max}+yw_{min}}{yw_{max}-yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near}-z_{far}} & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação de Normalização para Projeção Ortogonal

- A multiplicação dessa matriz pela matriz que transforma as coordenadas do mundo em coordenadas de visão produz a transformação completa para se obter as coordenadas normalizadas da projeção ortogonal

$$\mathbf{M}_{ortho,norm} \cdot \mathbf{M}_{WC,VC}$$

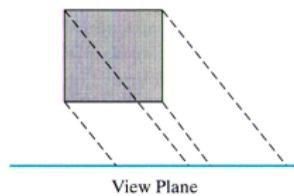
- Outras operações, como recorte, identificação de superfícies visíveis, etc. podem ser executadas de forma mais eficiente

Sumário

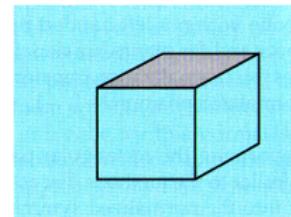
- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 **Transformações de Projeção**
 - Projeções Ortogonais
 - **Projeções Paralelas Oblíquas**
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Projeções Paralelas Oblíquas

- Quando as linhas de projeção não são perpendiculares ao plano de visão, o mapeamento obtido é conhecido como **projeção paralela oblíqua**
- Com esse tipo de projeção é possível obter visões combinando, por exemplo, visões superiores, frontais e laterais



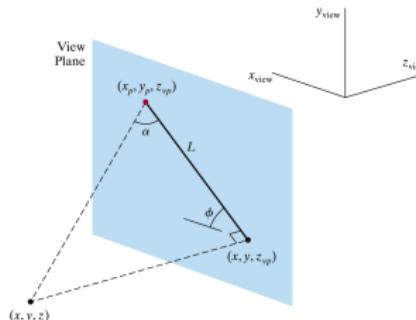
(a)



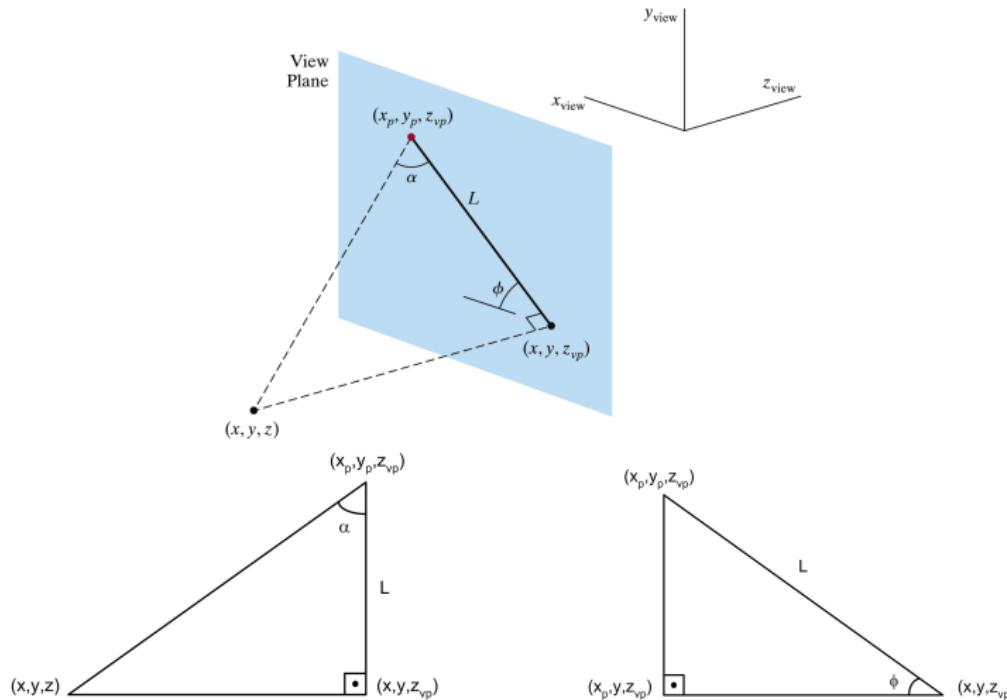
(b)

Projeções Paralelas Oblíquas para Projeto e Desenho

- A posição espacial (x, y, z) é projetada em (x_p, y_p, z_{vp}) no plano de visão que está na posição z_{vp} ao longo do eixo z
 - (x, y, z_{vp}) é a projeção ortogonal desse ponto
 - A linha de (x, y, z) até (x_p, y_p, z_{vp}) tem um ângulo de intersecção $0^0 \leq \alpha \leq 90^0$ com a linha sobre o plano de projeção que une (x_p, y_p, z_{vp}) e (x, y, z_{vp})
 - Essa linha tem tamanho L e tem um ângulo $0^0 \leq \phi \leq 360^0$ com a direção horizontal do plano de projeção



Projeções Paralelas Oblíquas para Projeto e Desenho



Projeções Paralelas Oblíquas para Projeto e Desenho

- É possível expressar as coordenadas da projeção em termos de x , y , L e ϕ

$$x_p = x + L \cos \phi$$

$$y_p = y + L \sin \phi$$

- O tamanho L depende do ângulo α e da distância perpendicular do ponto (x, y, z) ao plano de visão

$$\tan \alpha = \frac{z_{vp} - z}{L}$$

$$L = \frac{z_{vp} - z}{\tan \alpha}$$

- Fazendo $L_1 = \cot \alpha = (1 / \tan \alpha)$, temos

$$L = L_1(z_{vp} - z)$$

Projeções Paralelas Oblíquas para Projeto e Desenho

- Assim podemos escrever

$$x_p = x + L_1(z_{vp} - z) \cos \phi$$

$$y_p = y + L_1(z_{vp} - z) \operatorname{sen} \phi$$

- Com $L_1 = 0$ temos uma projeção ortogonal ($\text{ângulo de projeção } \alpha = 90^\circ$)

Projeções Paralelas Oblíquas para Projeto e Desenho

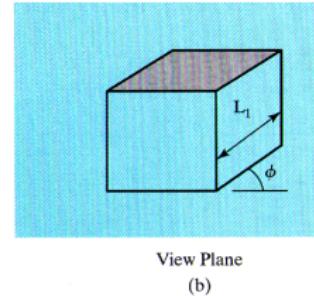
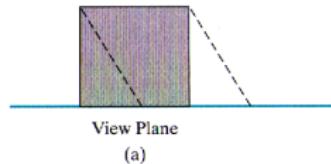
- Então temos a transformação de projeção oblíqua dado α e ϕ

$$\mathbf{M}_{obliqua} = \begin{bmatrix} 1 & 0 & -L_1 \cdot \cos \phi & z_{vp} \cdot L_1 \cdot \cos \phi \\ 0 & 1 & -L_1 \cdot \sin \phi & z_{vp} \cdot L_1 \cdot \sin \phi \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & -\cot \alpha \cdot \cos \phi & z_{vp} \cdot \cot \alpha \cdot \cos \phi \\ 0 & 1 & -\cot \alpha \cdot \sin \phi & z_{vp} \cdot \cot \alpha \cdot \sin \phi \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

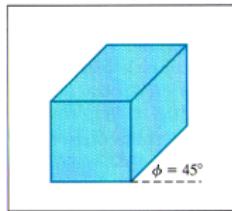
Projeções Paralelas Oblíquas para Projeto e Desenho

- Essas equações na verdade representam o cisalhamento no eixo z
 - Inclinar os planos com z constante e projetá-los no plano de projeção
 - As posições (x, y) de cada plano com z constante são modificadas por uma quantidade proporcional a distância desse para o plano de visão, projetando de forma precisa ângulos, distâncias e linhas paralelas

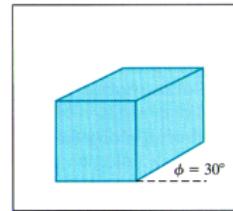


Projeções Paralelas Oblíquas Cavalier e Cabinet

- Escolhas típicas para ϕ são 30^0 e 45^0 que mostram uma combinação da frente, lateral e topo dos objetos
- Escolhas comuns para α são aquelas que resultam em $\tan \alpha = 1$ ($\alpha = 45^0$) e $\tan \alpha = 2$ ($\alpha = 63.4^0$)
 - Com $\tan \alpha = 1$ se produz projeções **cavalier** – linhas perpendiculares ao plano de projeção são projetadas sem alteração no tamanho
 - Com $\tan \alpha = 2$ se produz projeções **cabinet** – linhas perpendiculares ao plano de projeção são projetadas com metade de seu tamanho

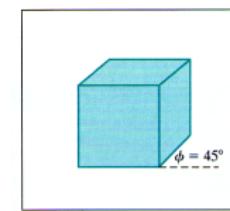


(a)

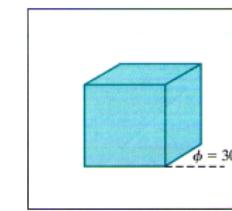


(b)

(a) Projeções Cavalier



(a)

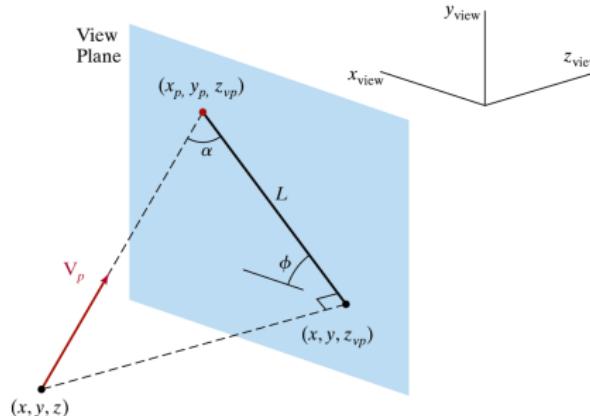


(b)

(b) Projeções Cabinet

Vetor de Projeções Paralelas Oblíquas

- Bibliotecas gráficas que suportam projeções paralelas oblíquas definem a direção de projeção ao plano de visão usando um **vetor de projeção paralela** $\mathbf{V}_p = (V_{px}, V_{py}, V_{pz})$
- Os pontos da cena são transferidos ao plano de visão ao longo de linhas paralelas a \mathbf{V}_p
 - $V_{py}/V_{px} = \tan \phi$

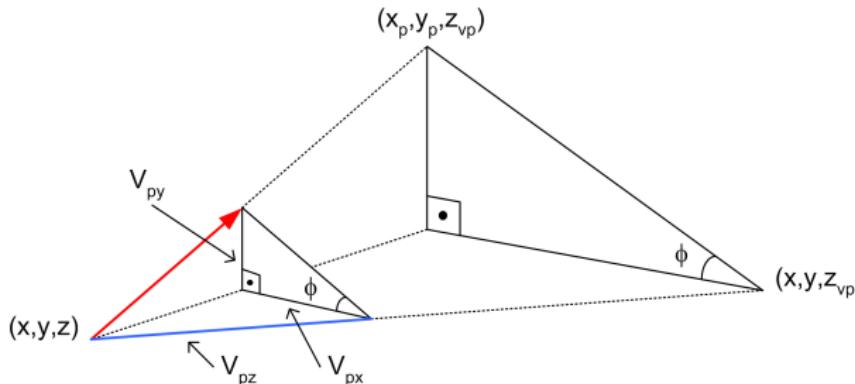


Vetor de Projeções Paralelas Oblíquas

- Usando semelhança de triângulos temos

$$\frac{x_p - x}{z_{vp} - z} = \frac{V_{px}}{V_{pz}}$$

$$\frac{y_p - y}{z_{vp} - z} = \frac{V_{py}}{V_{pz}}$$



Vetor de Projeções Paralelas Oblíquas

- Então considerando \mathbf{V}_p podemos escrever as equações de projeção paralela oblíqua

$$x_p = x + (z_{vp} - z) \frac{V_{px}}{V_{pz}}$$

$$y_p = y + (z_{vp} - z) \frac{V_{py}}{V_{pz}}$$

- Quando $V_{px} = V_{py} = 0$ temos uma projeção ortogonal

Matriz de Transformação de Projeção Oblíqua Paralela

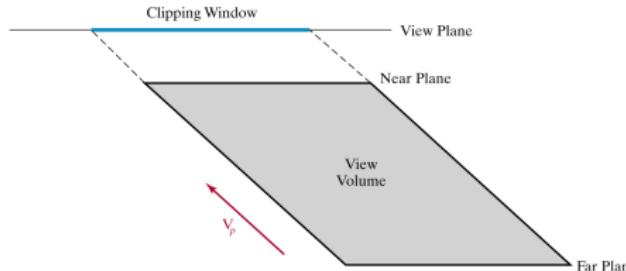
- Usando as equações anteriormente derivadas, podemos definir a matriz de transformação

$$\mathbf{M}_{obliqua} = \begin{bmatrix} 1 & 0 & -\frac{V_{px}}{V_{pz}} & z_{vp} \frac{V_{px}}{V_{pz}} \\ 0 & 1 & -\frac{V_{py}}{V_{pz}} & z_{vp} \frac{V_{py}}{V_{pz}} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Essa matriz altera os valores de x e y uma quantidade proporcional a distância do plano de visão na posição z_{vp} no eixo z_{view}

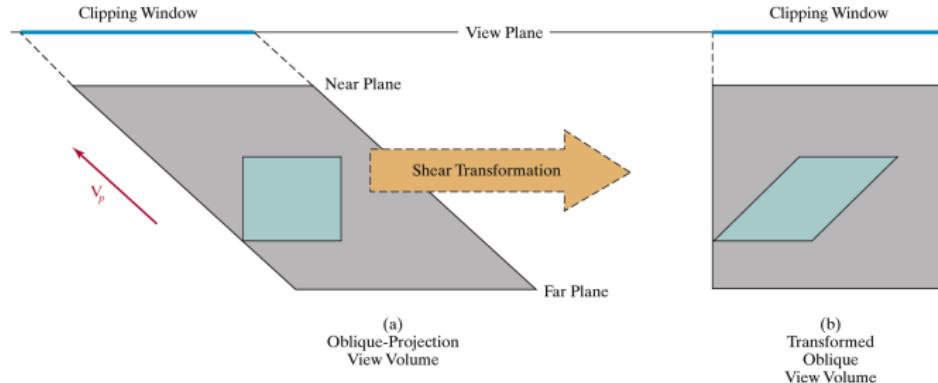
Janela de Recorte e Volume de Visão de Projeção Paralela Oblíqua

- O volume de visão de uma projeção paralela oblíqua é definido da mesma forma que o de projeção ortogonal
 - Escolhe-se uma janela de recorte no plano de visão definida por (xw_{min}, xw_{max}) e (yw_{min}, yw_{max})
 - A parte superior, inferior e laterais do volume de visão são então definidas pela direção de projeção e as arestas da janela de recorte



Matriz de Transformação de Projeção Oblíqua Paralela

- Essa matriz representa um cisalhamento no eixo z



Transformação de Normalização para Projeção Paralela Oblíqua

- Uma vez que as equações de projeção paralela oblíqua convertem as descrições dos objetos em posições ortogonais de coordenadas é possível aplicar a normalização após essa transformação – o volume obliquo de visão foi transformado em um paralelepípedo

$$\mathbf{M}_{obliqua,norm} = \mathbf{M}_{ortho,norm} \cdot \mathbf{M}_{obliqua}$$

- Essa matriz é então concatenada com a que transforma as coordenadas de mundo em coordenadas de visão para se definir a transformação total

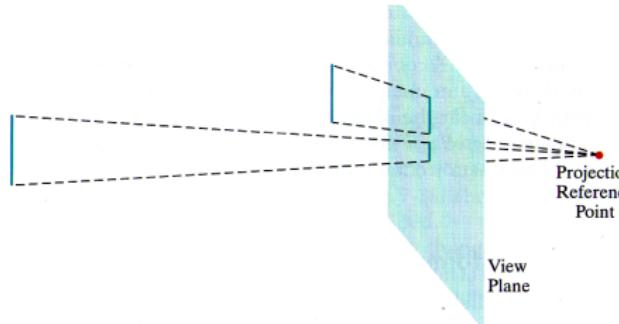
$$\mathbf{M}_{obliqua,norm} \cdot \mathbf{M}_{WC,VC}$$

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 **Transformações de Projeção**
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas**
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

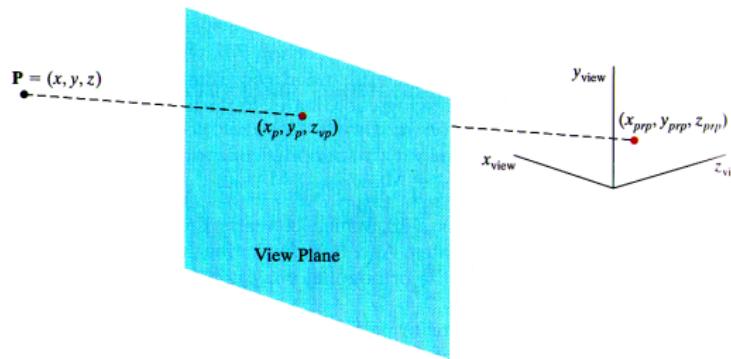
Projeções Perspectivas

- Para conseguir maior realismo que o obtido nas projeções paralelas temos que considerar que os raios de luz refletidos na cena seguem caminhos convergentes
- Isso pode ser aproximado projetando objetos ao plano de visão ao longo de caminhos convergentes a uma posição chamada **ponto de referência de projeção** (ou **centro de projeção**)



Transformação de Coordenadas de Projeção Perspectiva

- Algumas bibliotecas gráficas permitem que se escolha o ponto de referência de projeção $(x_{ppr}, y_{ppr}, z_{ppr})$



Transformação de Coordenadas de Projeção Perspectiva

- Considerando que a projeção do ponto (x, y, z) intersecte o plano de projeção na posição (x_p, y_p, z_{vp}) podemos descrever qualquer ponto ao longo desse linha de projeção como

$$x' = x + u(x_{prp} - x)$$

$$y' = y + u(y_{prp} - y)$$

$$z' = z + u(z_{prp} - z)$$

$$0 \leq u \leq 1$$

- No plano de visão $z' = z_{vp}$, então podemos encontrar u

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

Transformação de Coordenadas de Projeção Perspectiva

- Substituindo esse valor de u para as equações de x' e y' obtemos as equações de projeção perspectiva

$$x_p = x' = x \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) + x_{prp} \left(\frac{z_{vp} - z}{z_{prp} - z} \right)$$

$$y_p = y' = y \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) + y_{prp} \left(\frac{z_{vp} - z}{z_{prp} - z} \right)$$

Casos Especiais das Equações de Projeção Perspectiva

- Os parâmetros de projeção podem ser restringidos para facilitar os cálculos

- Se o centro projeção estiver sobre o eixo z_{view} , então

$$x_{prp} = y_{prp} = 0$$

$$x_p = x \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right), y_p = y \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right)$$

- Se o centro de projeção for fixado na origem, então é $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$

$$x_p = x \left(\frac{z_{vp}}{z} \right), y_p = y \left(\frac{z_{vp}}{z} \right)$$

Casos Especiais das Equações de Projeção Perspectiva

- Os parâmetros de projeção podem ser restringidos para facilitar os cálculos
- Se o plano de projeção for o plano uv e o centro de projeção estiver sobre o eixo z_{view} , então $x_{prp} = y_{prp} = z_{vp} = 0$

$$x_p = x \left(\frac{z_{prp}}{z_{prp} - z} \right), y_p = y \left(\frac{z_{prp}}{z_{prp} - z} \right)$$

Casos Especiais das Equações de Projeção Perspectiva

- Em geral o plano de projeção está entre o centro de projeção e a cena, mas outras posições são possíveis (menos sobre o plano de projeção)

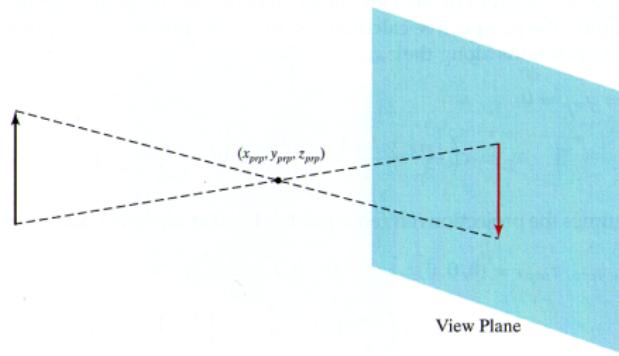


Figura: Os objetos são invertido se o ponto de referência está entre o plano de visão e a cena.

Casos Especiais das Equações de Projeção Perspectiva

- Os efeitos de perspectiva também dependem da distância entre o centro de projeção e o plano de visão

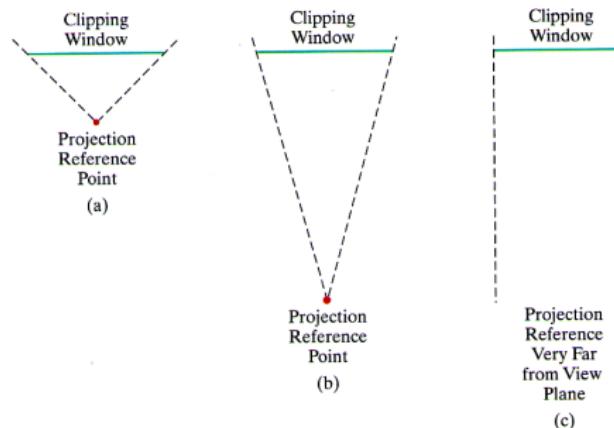


Figura: Se o centro de projeção está próximo ao plano de projeção, objetos mais próximos ao plano aparecerão muito maiores do que os distantes

Pontos de Fuga para Projeções Perspectivas

- Na projeção perspectiva
 - linhas paralelas entre si e ao plano de projeção são projetadas como linhas paralelas
 - linhas paralelas que não são paralelas ao plano de projeção são projetadas em linhas convergentes
- O ponto que parece que as linhas convergem é chamado **ponto de fuga**



Pontos de Fuga para Projeções Perspectivas

- Conjuntos de linhas que são paralelas a algum dos eixos principais de um objeto levam a composição dos **pontos de fuga principais**
 - É possível controlar o número desses pontos de fuga (um, dois ou três) por meio da orientação do plano de projeção
 - O número de pontos de fuga principais é igual a quantidade de eixos principais que intersectam o plano de projeção

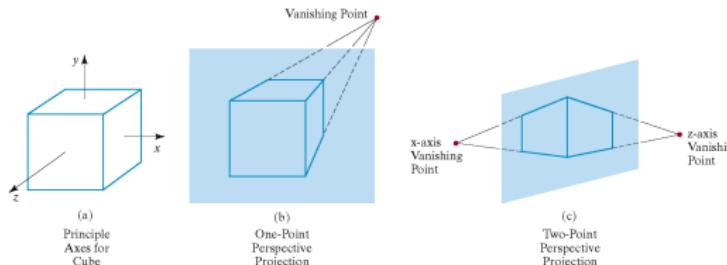
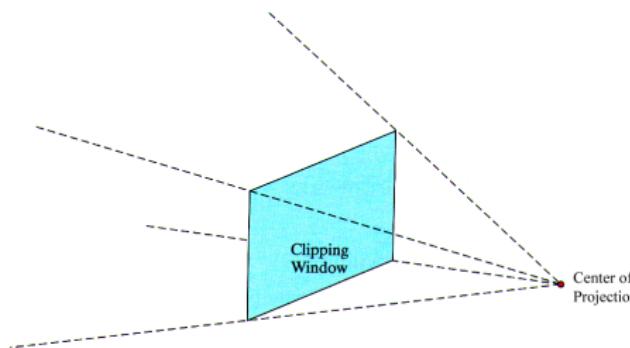


Figura: Em (b) o plano de projeção é paralelo ao plano xy (só intersecta z), e em (c) os eixos x e z são intersectados.

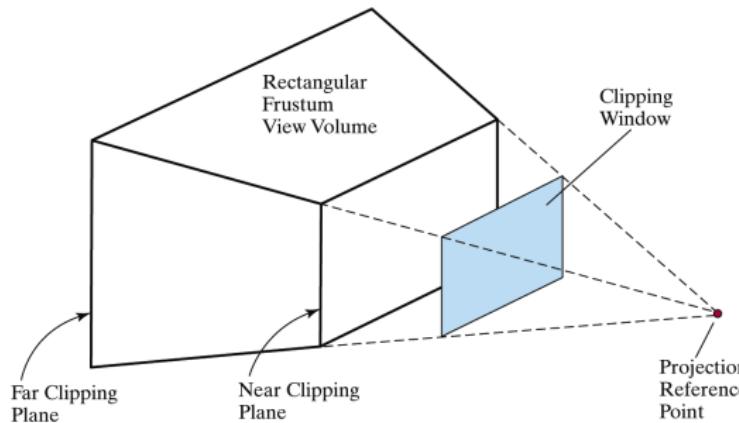
Volume de Projeção Perspectiva

- Em uma projeção perspectiva, o volume de visão definido é uma pirâmide infinita com seu ápice no centro de projeção, normalmente chamada de **pirâmide de visão**
 - Objetos fora dessa pirâmide são eliminados pelas rotinas de recorte



Volume de Projeção Perspectiva

- Adicionando os planos de recorte *near/far* perpendiculares ao eixo z_{view} essa pirâmide é truncada resultando em um tronco de pirâmide (**frustum**)



Matriz de Transformação de Projeção Perspectiva

- Não é possível a partir das equações derivadas anteriormente definir uma matriz de transformação perspectiva – os denominadores dos coeficientes de x e y são função de z

$$x_p = x \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) + x_{prp} \left(\frac{z_{vp} - z}{z_{prp} - z} \right)$$

$$y_p = y \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) + y_{prp} \left(\frac{z_{vp} - z}{z_{prp} - z} \right)$$

- Essa limitação pode ser superada usando coordenadas homogêneas

$$x_p = \frac{x_h}{h}, y_p = \frac{y_h}{h}$$

- Onde o parâmetro homogêneo é

$$h = z_{prp} - z$$

Matriz de Transformação de Projeção Perspectiva

- Os numeradores permanecem os mesmos

$$x_p = x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z)$$

$$y_p = y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z)$$

- E o fator homogêneo é

$$h = z_{prp} - z$$

Matriz de Transformação de Projeção Perspectiva

- Então pode-se definir uma matriz de transformação que converte uma posição espacial em coordenadas homogêneas
 - Inicialmente calcula-se as coordenadas homogêneas $\mathbf{P}_h = (x_h, y_h, z_h, h)$ de um ponto $\mathbf{P} = (x, y, z, 1)$ usando a matriz de projeção perspectiva

$$\mathbf{P}_h = \mathbf{M}_{pers} \cdot \mathbf{P}$$

- Então essas são divididas por h para se obter as coordenadas das posições transformadas

Matriz de Transformação de Projeção Perspectiva

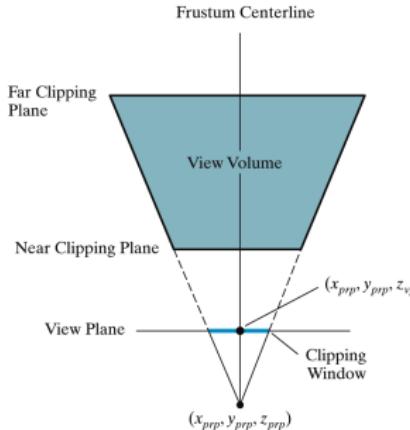
- Definir a matriz para encontrar x_h e y_h é de forma direta, mas informação de profundidade deve ser introduzida para o parâmetro homogêneo h não distorcer z
- Isso pode ser feito definindo os valores para a transformação de z de forma a normalizar as coordenadas z_p da projeção
 - Isso pode ser feito de várias formas, uma delas

$$\mathbf{M}_{pers} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

- Onde s_z e t_z são fatores de escala e translação para a normalização das coordenadas z

Frustum Simétrico de Projeção Perspectiva

- A linha do centro de projeção através do centro da janela de recorte e da volume de visão é a linha central do *frustum* de projeção perspectiva
 - Se essa for perpendicular ao plano de visão temos um **frustum simétrico**



Frustum Simétrico de Projeção Perspectiva

- Como a linha central do *frustum* intersecta o plano em $(x_{prp}, y_{prp}, z_{vp})$ é possível expressar os cantos da janela de recorte em termos das dimensões da janela

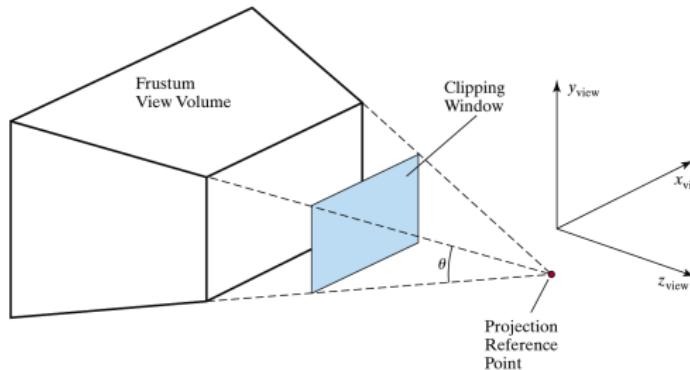
$$xw_{min} = x_{prp} - \frac{width}{2}, xw_{max} = x_{prp} + \frac{width}{2}$$

$$yw_{min} = y_{prp} - \frac{height}{2}, yw_{max} = y_{prp} + \frac{height}{2}$$

- Assim é possível especificar uma projeção perspectiva simétrica usando a largura e altura da janela de recorte ao invés das coordenadas da janela

Frustum Simétrico de Projeção Perspectiva

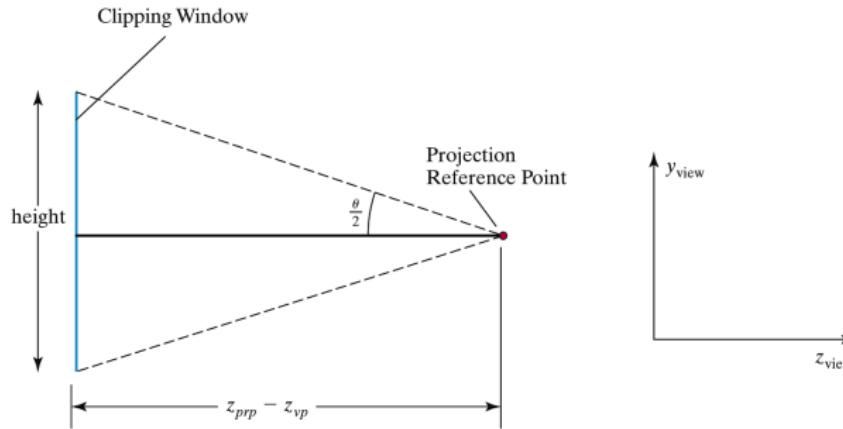
- Uma projeção perspectiva pode também ser aproximada considerando o **cone de visão**, definido pelo **ângulo do campo de visão**, de uma câmera
 - Grandes ângulos de campo de visão correspondem a lentes grandes-angulares
- De forma geral, o ângulo do campo de visão é definido entre o plano de recorte superior e o inferior do *frustum*



Frustum Simétrico de Projeção Perspectiva

- Dado um ponto de referência e a posição do plano de visão, o ângulo do campo de visão determina a altura da janela de recorte

$$\tan\left(\frac{\theta}{2}\right) = \frac{height/2}{z_{prp} - z_{vp}}$$



Frustum Simétrico de Projeção Perspectiva

- Para definir a largura é necessário considerar um parâmetro adicional que poderia ser a largura da janela ou a razão de aspecto $aspect = (width/height)$
- Assim podemos substituir os elementos $(z_{prp} - z_{vp})$ da diagonal da matriz \mathbf{M}_{pers} por

$$height = 2(z_{prp} - z_{vp}) \tan\left(\frac{\theta}{2}\right)$$

$$z_{prp} - z_{vp} = \frac{height}{2} \cot\left(\frac{\theta}{2}\right)$$

- Ou por

$$z_{prp} - z_{vp} = \frac{width \cdot \cot(\theta/2)}{2 \cdot aspect}$$

Frustum Simétrico de Projeção Perspectiva

- Assim temos

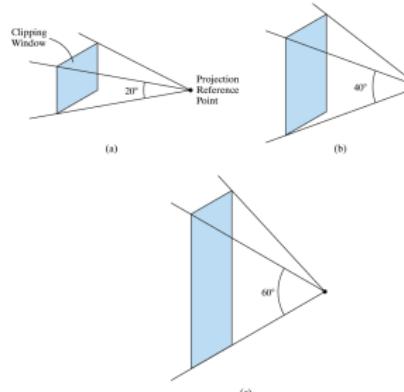
$$\mathbf{M}_{pers} = \begin{bmatrix} \frac{height}{2} \cot\left(\frac{\theta}{2}\right) & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & \frac{height}{2} \cot\left(\frac{\theta}{2}\right) & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

- Ou

$$\mathbf{M}_{pers} = \begin{bmatrix} \frac{width \cdot \cot(\theta/2)}{2 \cdot aspect} & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & \frac{width \cdot \cot(\theta/2)}{2 \cdot aspect} & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

Frustum Simétrico de Projeção Perspectiva

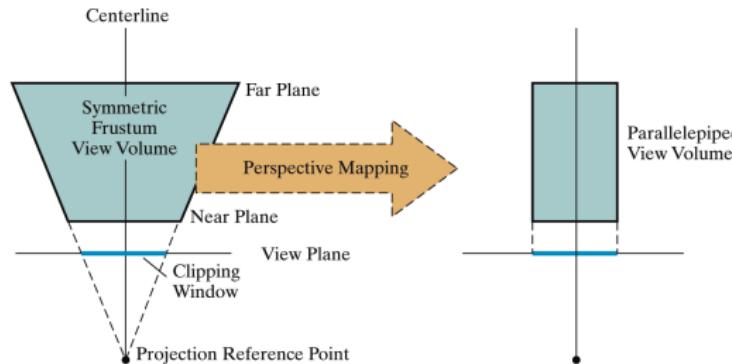
- Diminuir o ângulo do campo de visão diminui a janela de recorte
 - Mover o ponto de projeção para longe do plano de visão
 - Zoom-in de uma pequena região da cena
- Aumentar o ângulo do campo de visão aumenta a janela de recorte
 - Mover o ponto de projeção para próximo do plano de visão
 - Zoom-out da cena



Frustum Simétrico de Projeção Perspectiva

Observação Importante

- Para um *frustum* da projeção perspectiva simétrico, a transformação perspectiva mapeia localizações dentro do *frustum* a coordenadas de projeção ortogonais dentro de um paralelepípedo retangular

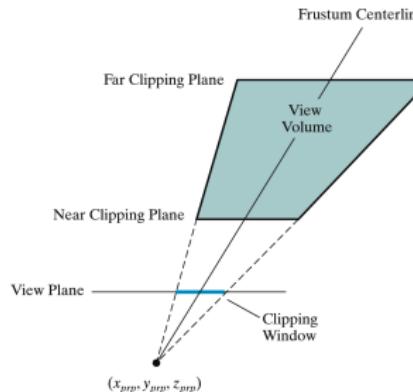


Frustum Simétrico de Projeção Perspectiva

- Com o *frustum* simétrico convertido a um volume de visão de projeção ortogonal é possível aplicar rotinas de normalização

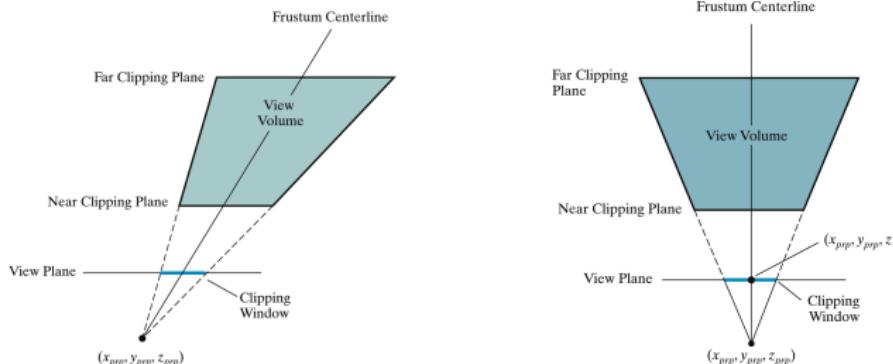
Frustum Oblíquo de Projeção Perspectiva

- Se a linha central do volume de visão não é perpendicular ao plano de visão, temos um **frustum oblíquo**
- Nesse caso podemos primeiro transformar esse em um *frustum* simétrico e então para um volume de visão normalizado



Frustum Oblíquo de Projeção Perspectiva

- Essa transformação pode ser obtida por meio de um cisalhamento na coordenada z
 - Altera as posições de qualquer plano que é perpendicular ao eixo z uma quantidade proporcional a distância do plano ao centro de projeção z_{ppr}
 - Move-se uma quantidade que traz o centro da janela de recorte para (x_{ppr}, y_{ppr})



Frustum Oblíquo de Projeção Perspectiva

- Os **cálculos** para a transformação de cisalhamento, projeção perspectiva e normalização são **facilitados** se o **centro de projeção está na origem**
 - Isso pode ser sempre garantido usando translações, movendo o centro de projeção para a origem
 - Isso pode também ser feito definindo um sistema de referência de visão com a origem no centro de projeção

Frustum Oblíquo de Projeção Perspectiva

- Considerando que o centro de projeção está em $(x_{ppr}, y_{ppr}, z_{ppr}) = (0, 0, 0)$ a matriz de cisalhamento fica

$$\mathbf{M}_{zshear} = \begin{bmatrix} 1 & 0 & sh_{zx} & 0 \\ 0 & 1 & sh_{zy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Frustum Oblíquo de Projeção Perspectiva

- Para encontrar esses fatores de cisalhamento, consideramos que o **plano de visão** está posicionado **sobre** o plano de recorte **near**
 - Movemos o centro da janela de recorte para $(0, 0)$ no plano de visão

$$\begin{bmatrix} 0 \\ 0 \\ z_{near} \\ 1 \end{bmatrix} = \mathbf{M}_{zshear} \cdot \begin{bmatrix} \frac{xw_{min} + xw_{max}}{2} \\ \frac{yw_{min} + yw_{max}}{2} \\ z_{near} \\ 1 \end{bmatrix}$$

- O que resulta em

$$sh_{zx} = -\frac{xw_{min} + xw_{max}}{2 \cdot z_{near}}$$

$$sh_{zy} = -\frac{yw_{min} + yw_{max}}{2 \cdot z_{near}}$$

Frustum Oblíquo de Projeção Perspectiva

- Com o **centro de projeção na origem** e com o plano de recorte **near sobre o plano de projeção**, a matriz de projeção perspectiva é simplificada

$$\mathbf{M}_{pers} = \begin{bmatrix} -z_{near} & 0 & 0 & 0 \\ 0 & -z_{near} & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Os valores de s_z e t_z são determinados pela normalização

Frustum Oblíquo de Projeção Perspectiva

- Concatenando essas matrizes, obtemos a matriz de projeção perspectiva obliqua que converte os pontos de uma cena em coordenadas de projeção ortogonal

$$\mathbf{M}_{obliquepers} = \mathbf{M}_{pers} \cdot \mathbf{M}_{zshear}$$

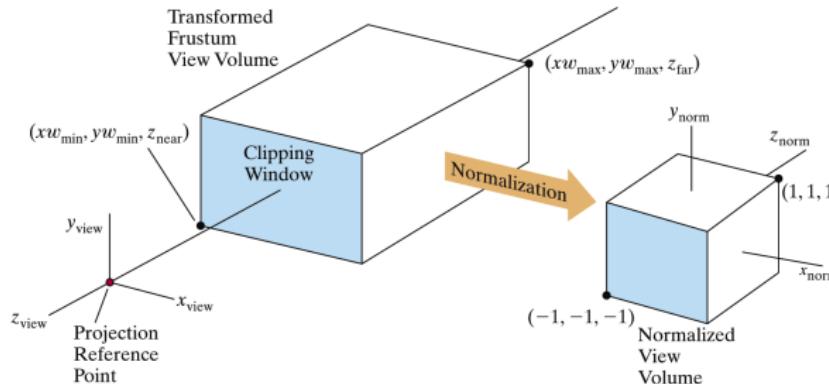
$$= \begin{bmatrix} -z_{near} & 0 & \frac{xw_{min}+xw_{max}}{2} & 0 \\ 0 & -z_{near} & \frac{yw_{min}+yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Frustum Oblíquo de Projeção Perspectiva

- Essa matriz transforma posições dos objetos para coordenadas homogêneas de projeção perspectiva
- Para se obter as coordenadas reais da projeção é necessário dividir pelo parâmetro homogêneo h
 - Essas são na verdade coordenadas de uma projeção ortogonal, de forma que uma projeção perspectiva transforma os pontos do *frustum* em um paralelepípedo de volume de visão

Transformação de Projeção Perspectiva Normalizada

- O último passo da projeção perspectiva é mapear o paralelepípedo obtido para um **volume de visão normalizado**
 - É aplicado o mesmo procedimento da projeção paralela



Transformação de Projeção Perspectiva Normalizada

- Os parâmetros para normalização da coordenada z já estão incluídos na matriz de projeção perspectiva, mas esses ainda precisam ser definidos
- Também é necessário os parâmetros para a normalização das coordenadas x e y
 - Não precisa de translação porque a linha central do paralelepípedo retangular é z_{view} , só precisa uma escala com relação a origem

$$\mathbf{M}_{xyscale} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- Concatenando essa matriz de escala com a matriz de projeção perspectiva temos

$$\mathbf{M}_{normpers} = \mathbf{M}_{xyscale} \cdot \mathbf{M}_{obliquepers}$$

$$= \begin{bmatrix} -z_{near} \cdot s_x & 0 & s_x \frac{xw_{min} + xw_{max}}{2} & 0 \\ 0 & -z_{near} \cdot s_y & s_y \frac{yw_{min} + yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- Desta transformação obtemos as coordenadas homegêneas

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M}_{normpers} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Que resulta na coordenadas de projeção

$$x_p = \frac{x_h}{h} = \frac{-z_{near} \cdot s_x \cdot x + s_x(xw_{min} + xw_{max})/2}{-z}$$

$$y_p = \frac{y_h}{h} = \frac{-z_{near} \cdot s_y \cdot y + s_y(yw_{min} + yw_{max})/2}{-z}$$

$$z_p = \frac{z_h}{h} = \frac{s_x \cdot z + t_z}{-z}$$

Transformação de Projeção Perspectiva Normalizada

- Para normalizar essa projeção queremos que $(x_p, y_p, z_p) = (-1, -1, -1)$ quando as coordenadas de entrada forem $(xw_{min}, yw_{min}, z_{near})$ e $(x_p, y_p, z_p) = (1, 1, 1)$ quando essas forem $(xw_{max}, yw_{max}, z_{far})$
- Assim, resolvendo as equações anteriores com essas restrições obtemos

$$s_x = \frac{2}{xw_{max} - xw_{min}}, s_y = \frac{2}{yw_{max} - yw_{min}}$$

$$s_z = \frac{z_{near} + z_{far}}{z_{near} - z_{far}}, t_z = \frac{2z_{near}z_{far}}{z_{near} - z_{far}}$$

Transformação de Projeção Perspectiva Normalizada

- Resultando na seguinte matriz de projeção perspectiva normalizada

$$\mathbf{M}_{normpers} =$$

$$\begin{bmatrix} \frac{-2z_{near}}{xw_{max}-xw_{min}} & 0 & \frac{xw_{max}+xw_{min}}{xw_{max}-xw_{min}} & 0 \\ 0 & \frac{-2z_{near}}{yw_{max}-yw_{min}} & \frac{yw_{max}+yw_{min}}{yw_{max}-yw_{min}} & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & -\frac{2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- Se o volume de projeção perspectiva for definido como simétrico, é possível expressar os elementos da projeção perspectiva normalizada em termos do ângulo do campo de visão e das dimensões da janela de recorte
 - Com o ponto de referência da projeção na origem e o plano de visão sobre o plano de recorte *near*, temos

$$\mathbf{M}_{normsympers} =$$

$$\begin{bmatrix} \frac{\cot(\frac{\theta}{2})}{aspect} & 0 & 0 & 0 \\ 0 & \cot(\frac{\theta}{2}) & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & -\frac{2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- A matriz completa de transformação das coordenadas do mundo para projeção perspectiva normalizada é a composição

$$\mathbf{M}_{normpers} \cdot \mathbf{R} \cdot \mathbf{T} = \mathbf{M}_{normpers} \cdot \mathbf{M}_{WC,VC}$$

- Após essa matriz ser aplicada, as rotinas de recorte podem ser executadas

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Transformação de Viewport e Coordenadas de Tela 3D

- Após o conteúdo do volume de visão ter sido definido, esse pode ser transferido para coordenadas da tela
- Esse é um processo semelhante ao 2D, porém informação de profundidade é preservada para teste de visibilidade e *rendering*
 - A variável z é normalizada entre 0 e 1, sendo que $z = 0$ temos a altura da tela

$\mathbf{M}_{normviewvol,3Dscreen} =$

$$\begin{bmatrix} \frac{xv_{max}-xv_{min}}{2} & 0 & 0 & \frac{xv_{max}+xv_{min}}{2} \\ 0 & \frac{yv_{max}-yv_{min}}{2} & 0 & \frac{yv_{max}+yv_{min}}{2} \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação de Viewport e Coordenadas de Tela 3D

- Nas coordenadas normalizadas, $z_{norm} = -1$ é mapeada para a coordenada na tela $z_{screen} = 0$
 - *Viewport* é definida como $(xv_{min}, yv_{min}, 0)$ e $(xv_{max}, yv_{max}, 0)$
- As posições x e y são enviadas para o **frame buffer** (informação de cor para os pontos na tela)
- Os valores de z são enviados para o **depth buffer** para serem usados em rotinas para determinação de cor e visibilidade

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Funções OpenGL para Viewing 3D

- A biblioteca **OpenGL** inclui funções para
 - Projeção ortogonal
 - Projeção perspectiva oblíqua (e simétrica)
 - Transformação de *viewport*

- A biblioteca **OpenGL Utility (GLU)** inclui funções para
 - Especificar os parâmetros de visão
 - Definir a transformação de projeção perspectiva simétrica

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Função de Transformação de Visão OpenGL

- Os parâmetros de visão formam uma matriz que é concatenada com a matriz *MODELVIEW* corrente, então inicia-se especificando

```
1 glMatrixMode(GL_MODELVIEW);
```

Função de Transformação de Visão OpenGL

- Para especificar os parâmetros de visão usamos a função

```
1 gluLookAt(GLdouble x0, GLdouble y0, GLdouble z0,  
2           GLdouble xref, GLdouble yref, GLdouble zref,  
3           GLdouble Vx, GLdouble Vy, GLdouble Vz);
```

- Essa função define
 - A origem do sistema de visão $\mathbf{P}_0 = (x_0, y_0, z_0)$ no sistema de coordenadas de mundo (a localização da câmera)
 - A posição de referência $\mathbf{P}_{ref} = (x_{ref}, y_{ref}, z_{ref})$ (para onde a câmera aponta)
 - O vetor *view-up* $\mathbf{V} = (Vx, Vy, Vz)$

Função de Transformação de Visão OpenGL

- A direção positiva do eixo z_{view} do sistema de coordenadas de visão está na direção $\mathbf{N} = \mathbf{P}_0 - \mathbf{P}_{ref}$
 - A direção de visão está na direção negativa do eixo z_{view}
- Os parâmetros especificados usando a função `gluLookAt(...)` são usados para compor a matriz $\mathbf{M}_{WC,VC}$
 - Matriz que alinha o sistema de visão com o sistema de coordenadas de mundo
- Por padrão os parâmetros de visão da `gluLookAt(...)` são
 - $\mathbf{P}_0 = (0, 0, 0)$
 - $\mathbf{P}_{ref} = (0, 0, -1)$
 - $\mathbf{V} = (0, 1, 0)$

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL**
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Função de Projeção Ortogonal OpenGL

- Para se definir as matrizes de projeção é necessário especificar que se está trabalhando com a matriz *PROJECTION*

```
1 glMatrixMode(GL_PROJECTION);
```

- Para se definir uma projeção ortogonal, a seguinte função é chamada

```
1 glOrtho(GLdouble xwmin, GLdouble xwmax,  
2         GLdouble ywmin, GLdouble ywmax,  
3         GLdouble dnear, GLdouble dfar);
```

- Os 4 primeiros parâmetros definem as coordenadas da janela de recorte e os 2 últimos as distâncias para os planos de recorte *near/far*

Função de Projeção Ortogonal OpenGL

- O plano de projeção é sempre coincidente com o plano de recorte *near*
- Os parâmetros *dnear* e *dfar* denotam distâncias na direção negativa de *zview*
 - Por exemplo, se *dfar* = 55.0, o plano de recorte *far* estará na posição *zfar* = -55.0
 - Quaisquer valores podem ser atribuídos, desde que *dnear* < *dfar*
- Por padrão, a *OpenGL* usa os seguintes valores para essa função

```
1 glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

Função de Projeção Ortogonal OpenGL

- Não existe função *OpenGL* para criar projeções paralelas oblíquas
 - Porém, é possível definir as matrizes de transformação diretamente dentro da *OpenGL*, ou então rotacionar a cena até uma posição apropriada de forma que a projeção ortogonal na direção z_{view} dê o resultado esperado

Função de Projeção Ortogonal OpenGL

```
1 #include <stdlib.h>
2 #include <GL/glut.h>
3 #include <math.h>
4
5 void init(void)
6 {
7     glClearColor(1.0,1.0,1.0,1.0); //define a cor de fundo
8     glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
9
10    glMatrixMode(GL_MODELVIEW); //define que a matriz é a model view
11    glLoadIdentity(); //carrega a matriz de identidade
12    gluLookAt(1.0,0.5,0.5, //posição da câmera
13               0.0,0.0,0.0, //para onde a câmera aponta
14               0.0,1.0,0.0); //vetor view-up
15
16    glMatrixMode(GL_PROJECTION); //define que a matriz é a de projeção
17    glLoadIdentity(); //carrega a matriz de identidade
18    glOrtho(-5.0,5.0,-5.0,5.0,-5.0,5.0); //define uma projeção ortogonal
19 }
```

Função de Projeção Ortogonal OpenGL

```
1 void display(void)
2 {
3     //limpa o buffer
4     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
5
6     //define que a matrix é a de modelo
7     glMatrixMode(GL_MODELVIEW);
8
9     //desenha um cubo
10    glColor3f(1.0,0.0,0.0);
11    glutWireCube(1.0);
12
13    //força o desenho das primitivas
14    glutSwapBuffers();
15 }
16
17 int main(int argc, char **argv)
18 {
19     glutInit(&argc, argv);
20     glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
21     glutInitWindowSize(500, 500);
22     glutCreateWindow("Projecao");
23
24     init();
25     glutDisplayFunc(&display); //registra função de desenho
26
27     glutMainLoop();
28
29     return 0;
30 }
```

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - **Projeção Perspectiva Simétrica OpenGL**
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Função para Projeção Perspectiva Simétrica OpenGL

- Para se criar um volume de visão *frustum* simétrico sobre a direção de visão (direção negativa do eixo z_{view}) usamos

```
1 gluPerspective(GLdouble theta, GLdouble aspect, GLdouble dnear, GLdouble dfar);
```

- Os parâmetros *theta* e *aspect* definem o tamanho e posição da janela de recorte
- Os parâmetros *dnear* e *dfar* especificam as distâncias do ponto de visão (origem do sistema de coordenadas) para os planos de recorte *near* e *far*
- $0^0 \leq \text{theta} \leq 180^0$ define o ângulo do campo de visão
- aspect* é o valor da razão de aspecto *width/height* da janela de recorte

Função para Projeção Perspectiva Simétrica OpenGL

- O ponto de referência da projeção (ponto de visão) é a origem do sistema de coordenadas de visão
 - O plano de recorte *near* coincide com o plano de visão
-
- Os planos de recorte *near/far* devem estar ao longo da direção negativa de z_{view} e não podem estar atrás da posição de visão
 - Os valores devem ser $d_{near} > 0$ e $d_{far} > 0$

Função para Projeção Perspectiva Simétrica OpenGL

```
1 void init(void)
2 {
3     glClearColor(1.0,1.0,1.0,1.0); //define a cor de fundo
4     glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
5
6     glMatrixMode(GL_MODELVIEW); //define que a matrix é a model view
7     glLoadIdentity(); //carrega a matrix de identidade
8     gluLookAt(2.0,0.0,2.0, //posição da câmera
9                0.0,0.0,0.0, //para onde a câmera aponta
10               0.0,1.0,0.0); //vetor view-up
11
12    glMatrixMode(GL_PROJECTION); //define que a matrix é a de projeção
13    glLoadIdentity(); //carrega a matrix de identidade
14    gluPerspective(60.0, 1.0, 0.1, 10.0); //define uma projeção perspectiva
15 }
```

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Função para Projeção Perspectiva Genérica OpenGL

- É possível usar a seguinte função para definir uma projeção perspectiva simétrica ou oblíqua

```
1 glFrustum(GLdouble xwmin, GLdouble xwmax,
2           GLdouble ywmin, GLdouble ywmax,
3           GLdouble dnear, GLdouble dfar);
```

- Os 4 primeiros parâmetros definem as coordenadas da janela de recorte no plano *near*
- Os 2 últimos especificam as distâncias da origem do sistema de coordenadas dos planos *near/far* ao longo do eixo z_{view}
 - $z_{near} = -dnear$ e $z_{far} = -dfar$
 - As distâncias *dnear* e *dfar* devem ser positivas

Função para Projeção Perspectiva Genérica OpenGL

- O plano de recorte *near* é o plano de projeção
- O ponto de referência de projeção é a origem do sistema de coordenadas de visão
- A janela de recorte pode ser especificada em qualquer posição do plano *near*
 - Se essas forem $xw_{min} = -xw_{max}$ e $yw_{min} = -yw_{max}$ temos um *frustum* simétrico, com a eixo z_{view} como sua linha central

Função para Projeção Perspectiva Genérica OpenGL

```
1 void init(void)
2 {
3     glClearColor(1.0,1.0,1.0,1.0); //define a cor de fundo
4     glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
5
6     glMatrixMode(GL_MODELVIEW); //define que a matrix é a model view
7     glLoadIdentity(); //carrega a matrix de identidade
8     gluLookAt(5.0,0.0,5.0, //posição da câmera
9                0.0,0.0,0.0, //para onde a câmera aponta
10               0.0,1.0,0.0); //vetor view-up
11
12    glMatrixMode(GL_PROJECTION); //define que a matrix é a de projeção
13    glLoadIdentity(); //carrega a matrix de identidade
14    glFrustum(-2.0,2.0,-2.0,2.0,5.0,10.0); //define uma projeção perspectiva (↔
15                                         simétrica)
}
```

Sumário

- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Paralelas Oblíquas
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Transformação de Visão OpenGL
 - Função de Projeção Ortogonal OpenGL
 - Projeção Perspectiva Simétrica OpenGL
 - Projeção Perspectiva Genérica OpenGL
- 7 Algoritmos de Recorte 3D

Algoritmos de Recorte 3D

- Se o cubo de visão for normalizado, o processo de recorte pode ser aplicado de forma mais eficiente
- Os algoritmos de recorte eliminam os objetos fora do volume normalizado de visão e processam o resto
- Esses são extensões dos algoritmos de recorte 2D, mas com planos como fronteira ao invés de linhas
- O recorte é aplicado após todas as transformações terem sido aplicadas

Recorte em Coordenadas Homogêneas 3D

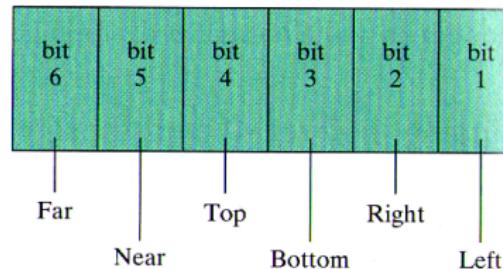
- No *pipeline* de visualização 3D, após as posições terem passado pelas transformações geométricas, de visão e projeção, temos uma representação homogênea

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Onde \mathbf{M} é a concatenação de todas as matrizes
- Um método efetivo para recorte é aplicar as rotinas de recorte sobre a representação homogênea
 - Os procedimentos de recorte podem ser iguais independente do tipo de projeção aplicada

Código de Região 3D

- O conceito de código de região 2D (algoritmo Cohen-Sutherland) pode ser estendido para representações 3D adicionando alguns bits
 - Código de 6 bits



- Os valores desses bits são calculados de forma semelhante ao 2D
 - 0 está dentro de alguma fronteira, 1 está fora

Código de Região 3D

- Considerando que estamos trabalhando com coordenadas homogêneas $\mathbf{P} = (x_h, y_h, z_h, h)$, um ponto dentro do volume de visão deve satisfazer

$$-1 \leq \frac{x_h}{h} \leq 1$$

$$-1 \leq \frac{y_h}{h} \leq 1$$

$$-1 \leq \frac{z_h}{h} \leq 1$$

- Assumindo que $h \neq 0$ podemos calcular

$$-h \leq x_h \leq h, \quad -h \leq y_h \leq h, \quad -h \leq z_h \leq h, \quad \text{se } h > 0$$

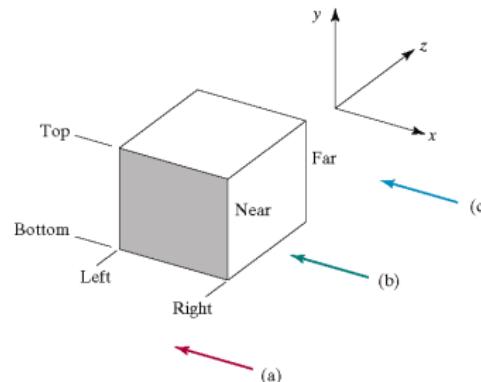
$$h \leq x_h \leq -h, \quad h \leq y_h \leq -h, \quad h \leq z_h \leq -h, \quad \text{se } h < 0$$

Código de Região 3D

- Com $h > 0$ podemos definir os valores dos bits como

$$bit_1 = 1 \text{ se } h + x_h < 0 \text{ (esquerda)}$$
$$bit_2 = 1 \text{ se } h - x_h < 0 \text{ (direita)}$$
$$bit_3 = 1 \text{ se } h + y_h < 0 \text{ (inferior)}$$
$$bit_4 = 1 \text{ se } h - y_h < 0 \text{ (superior)}$$
$$bit_5 = 1 \text{ se } h + z_h < 0 \text{ (near)}$$
$$bit_6 = 1 \text{ se } h - z_h < 0 \text{ (far)}$$

Código de Região 3D



011001	011000	011010
010001	010000	010010
010101	010100	010110

Region Codes
In Front of Near Plane
(a)

001001	001000	001010
000001	000000	000010
000101	000100	000110

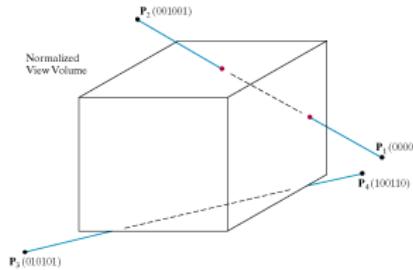
Region Codes
Between Near and Far Planes
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

Region Codes
Behind Far Plane
(c)

Recorte de Ponto e Linha 3D

- Pontos que não tenham código 000000 são eliminados
- O recorte de linha é essencialmente o mesmo do 2D
 - Se ambos os pontos finais da linha são 000000, a linha está dentro
 - Uma linha está dentro se a operação “ou” sobre seus pontos finais for igual a 000000
 - Uma linha está fora se a operação “e” sobre seus pontos finais for diferente de 000000



Recorte de Ponto e Linha 3D

- Se uma linha falha nesses testes, suas equações são avaliadas para calcular as intersecções
- Para uma linha definida por $\mathbf{P}_1 = (x_{h1}, y_{h1}, z_{h1}, h_1)$ e $\mathbf{P}_2 = (x_{h2}, y_{h2}, z_{h2}, h_2)$, podemos escrever as equações paramétricas

$$\mathbf{P} = \mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)u, \quad 0 \leq u \leq 1$$

- Ou seja

$$x_h = x_{h1} + (x_{h2} - x_{h1})u$$

$$y_h = y_{h1} + (y_{h2} - y_{h1})u$$

$$z_h = z_{h1} + (z_{h2} - z_{h1})u$$

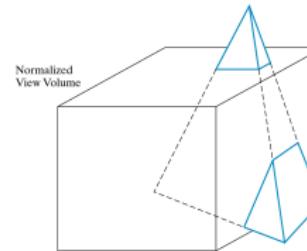
$$h = h_1 + (h_2 - h_1)u$$

Recorte de Ponto e Linha 3D

- Usando essas equações, o processo é o mesmo do recorte 2D
 - As intersecções são calculadas considerando as equações dos planos que definem o volume de recorte
 - Conforme essas intersecções são calculadas, os códigos de região são atualizados, junto com os valores de u

Recorte de Polígonos 3D

- Pacotes gráficos normalmente lidam com objetos descritos com equações lineares, portanto rotinas de recorte 3D devem ser aplicadas sobre polígonos



- Testes triviais podem ser aplicados ao *bounding box* dos objetos poligonais para testes de rejeição aceitação