

Viewing 3D

SCC0250 - Computação Gráfica

Prof. Rosane Minghim

rminghim@icmc.usp.br

P.A.E. Nícolas Roque *nrsantos@usp.br*

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

18 de abril de 2017



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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica OpenGL
- 7 Algoritmos de Recorte 3D

Visão Geral

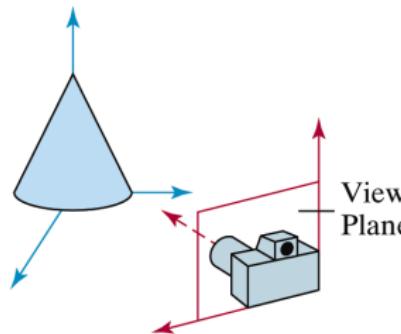
Viewing Pipeline 3D

- 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

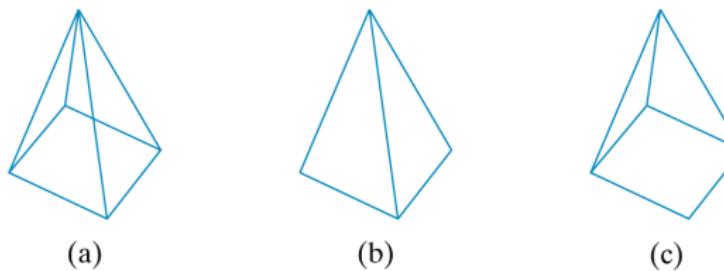
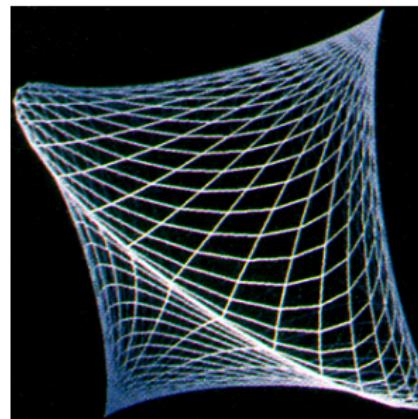


Figura: Problemas de ambiguidade pela falta de informação de profundidade em (a), que pode ser interpretada como (b) ou (c)

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



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.

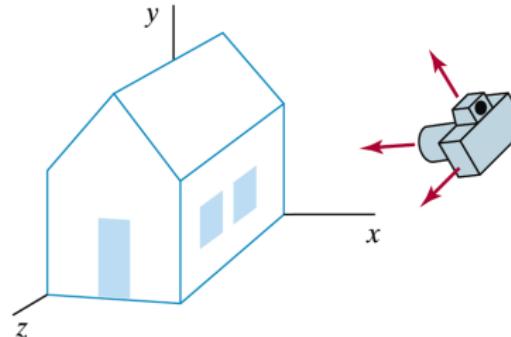


Sumário

- 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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*

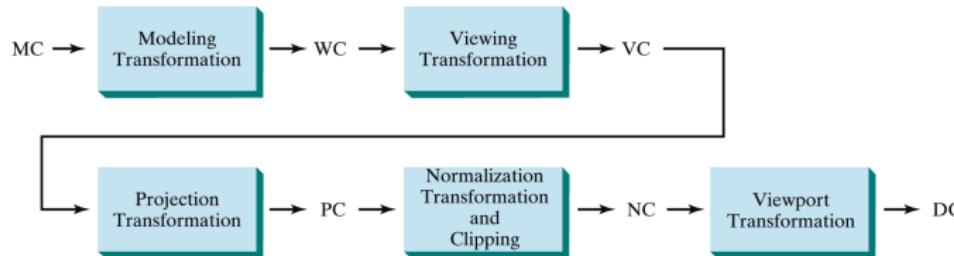


Viewing Pipeline 3D

- 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 no espaço (**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 mundo**

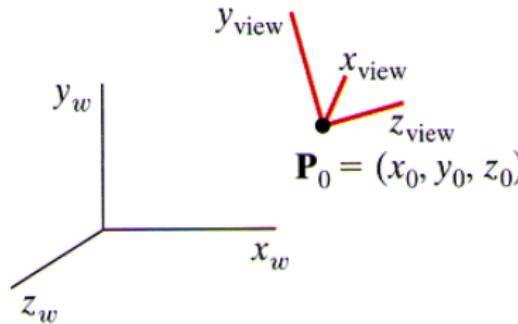


Sumário

- 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica OpenGL
- 7 Algoritmos de Recorte 3D

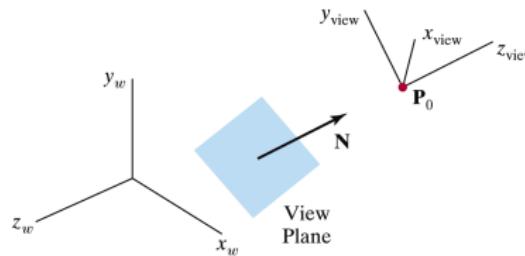
Parâmetros de Coordenadas de Visão 3D

- 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 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



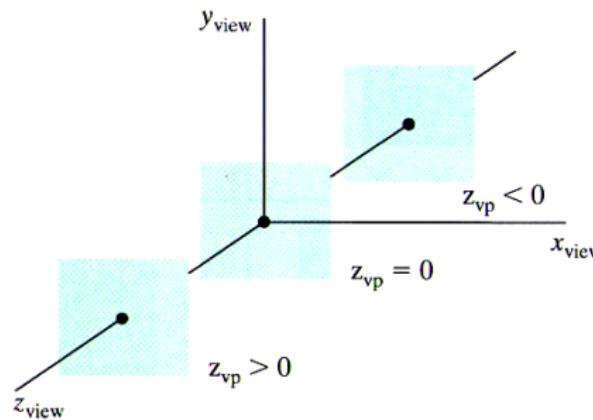
Vetor Normal ao Plano de Visão

- 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**



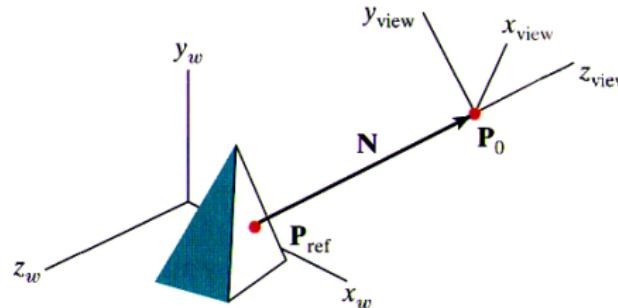
Vetor Normal ao Plano de Visão

- Um 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 (o contrário também é válido)
 - 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}

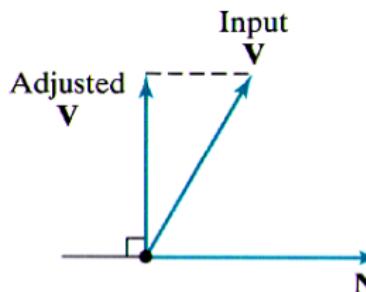


Figura: Ajuste do vetor *view-up* \mathbf{V} para torná-lo perpendicular a \mathbf{N}

Sistema de Coordenadas de Visão uvn

Sistema de Coordenadas de Visão uvn

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

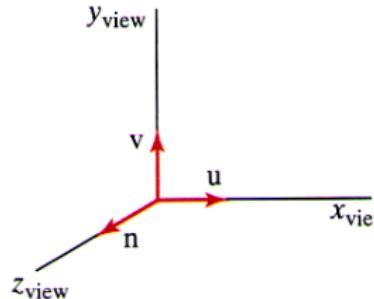
Sistema de Coordenadas de Visão uvn

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

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

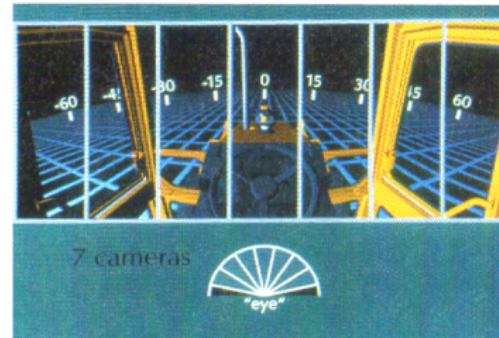
$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V} \times \mathbf{n}|} = (u_x, u_y, u_z)$$

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



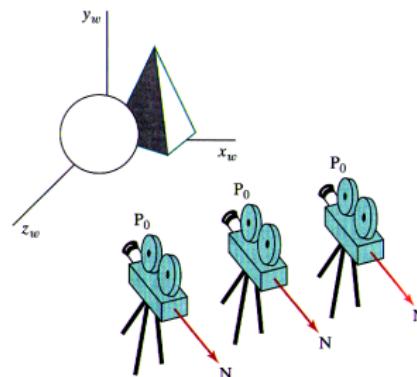
Gerando Efeitos de Visão 3D

- 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

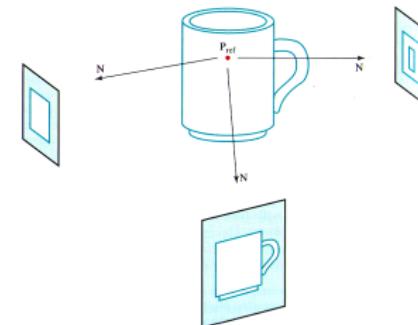


Gerando Efeitos de Visão 3D

- Efeitos de mover a câmera (pam) podem ser obtidos fixando \mathbf{N} e modificando o ponto de visão
- Para 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
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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

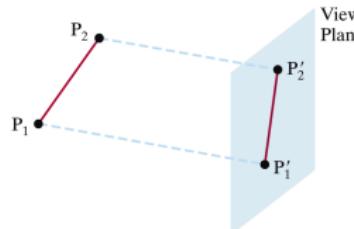
$$\begin{aligned}\mathbf{M}_{WC,VC} &= \mathbf{R} \cdot \mathbf{T} \\ &= \begin{bmatrix} u_x & u_y & u_z & -\mathbf{u} \cdot \mathbf{P}_0 \\ v_x & v_y & v_z & -\mathbf{v} \cdot \mathbf{P}_0 \\ n_x & n_y & n_z & -\mathbf{n} \cdot \mathbf{P}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

Sumário

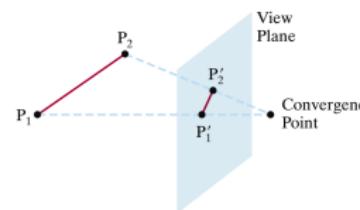
- 1 Introdução
- 2 Viewing Pipeline 3D
- 3 Parâmetros de Coordenadas de Visão 3D
- 4 Transformação do Sistema de Coordenadas de Mundo para o de Visão
- 5 Transformações de Projeção
 - Projeções Ortogonais
 - Projeções Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL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



(c) Projeção Paralela



(d) Projeção Perspectiva

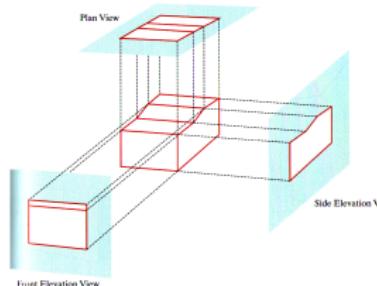
Sumário

- 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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



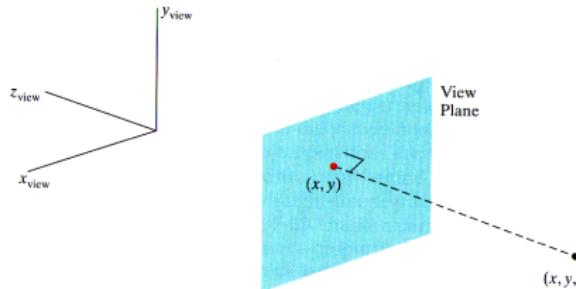
Coordenadas de Projeções Ortogonais

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

$$x_p = x$$

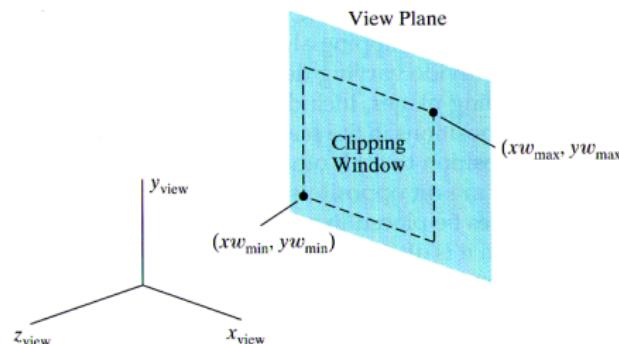
$$y_p = y$$

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



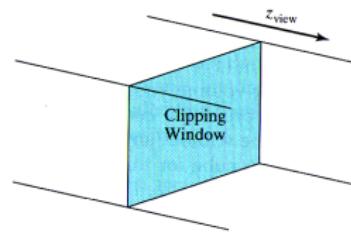
Janela de Recorte e Volume de Projeção Ortogonal

- 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}

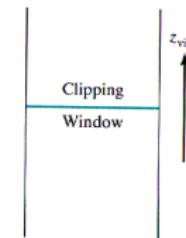


Janela de Recorte e Volume de Projeção Ortogonal

- 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**



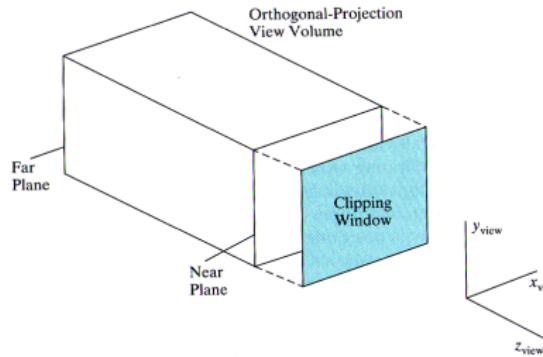
Side View
(a)



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

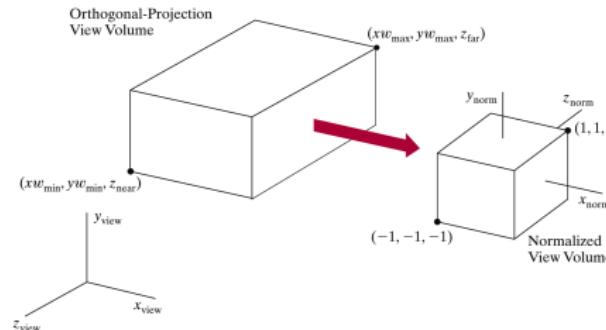


Figura: Transformação de normalização em um sistema de referência dado pela mão esquerda (normalmente o sistema da tela)

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

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

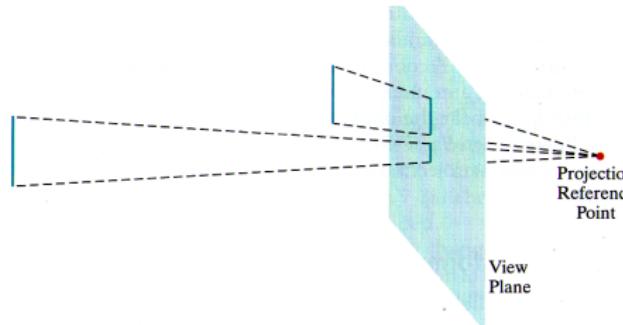
- Outras operações, como recorte, identificação de superfícies visíveis, etc. podem ser executadas de forma mais eficiente

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 Perspectivas**
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica OpenGL
- 7 Algoritmos de Recorte 3D

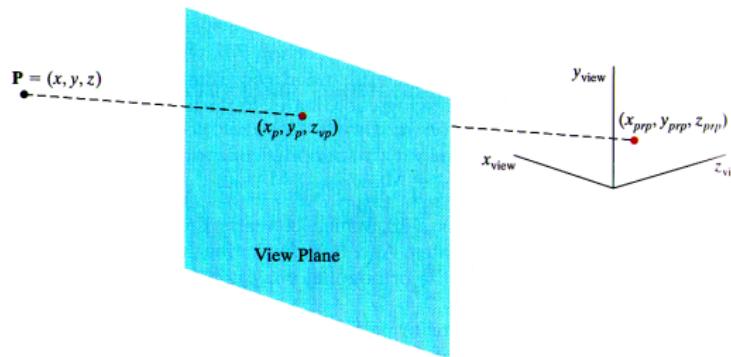
Projeções Perspectivas

- 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

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

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

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

- 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

Matriz de Transformação de Projeção Perspectiva

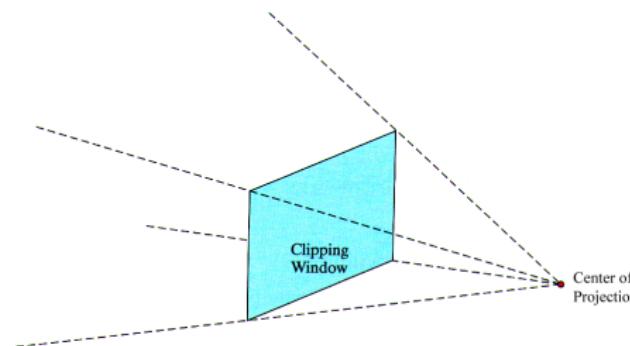
- Não se esqueça que essa matriz converte uma posição espacial em coordenadas homogêneas, portanto, o resultado obtido deve ser dividido pelo fator homogêneo h
- Em outras palavras, seja $\mathbf{P} = (x, y, z, 1)$ o ponto a ser projetado, o resultado obtido $\mathbf{P}_h = (x_h, y_h, z_h, h)$

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

- Deverá ser dividido por h para se obter as coordenadas das posições transformadas

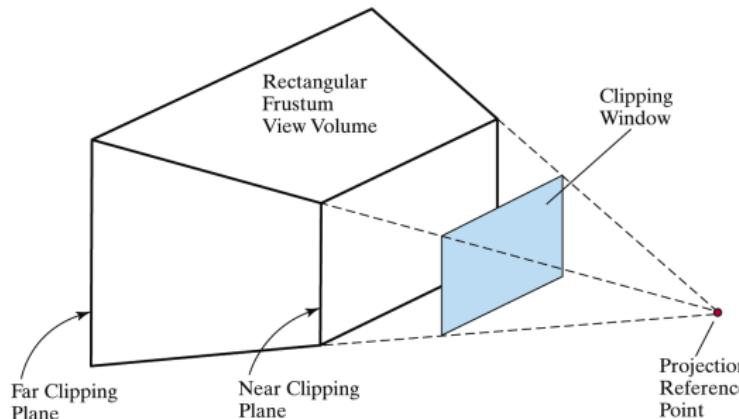
Volume de Projeção Perspectiva

- 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



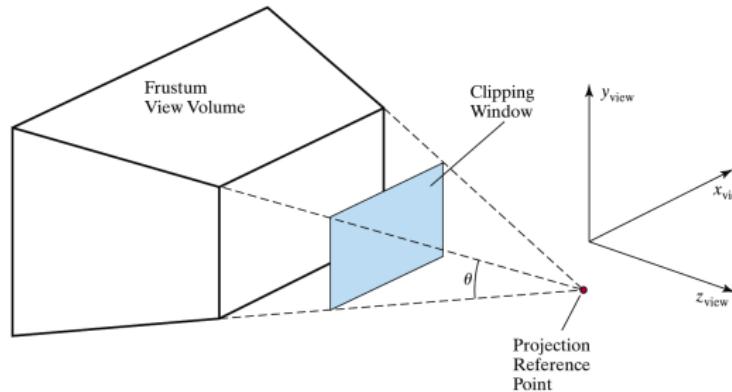
Volume de Projeção Perspectiva

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



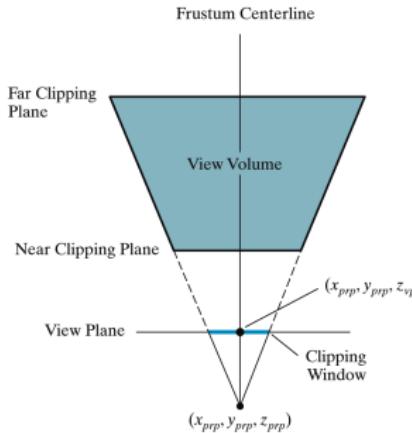
Frustum Simétrico de Projeção Perspectiva

- Uma projeção perspectiva pode também ser aproximada considerando o **cone de visão**, definido pelo **ângulo do campo de visão**, de uma câmera



Frustum Simétrico de Projeção Perspectiva

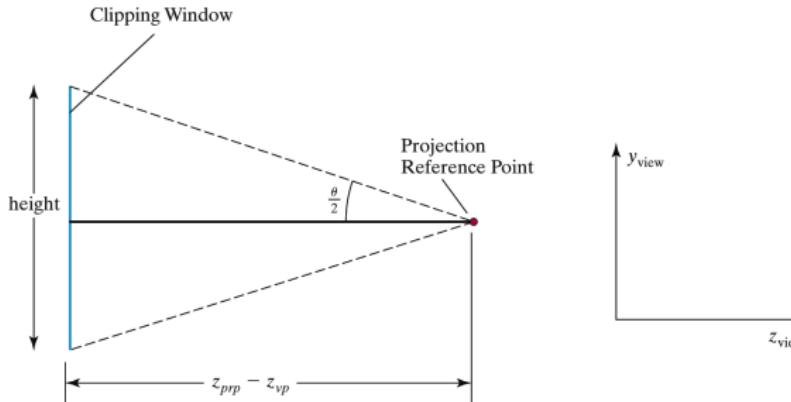
- Quando trabalhamos com **projeções perspectivas simétricas**, a linha que liga o centro de projeção ao meio do plano de visão, perpendicular a esse, é a linha central do **frustum simétrico**



Frustum Simétrico de Projeção Perspectiva

- Assim, dado um ponto de referência $(x_{prp}, y_{prp}, z_{prp})$ e a posição z_{vp} do plano de visão, o ângulo θ do campo de visão pode ser usado para determinar 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 $(z_{ppr} - z_{vp})$ da diagonal da matriz \mathbf{M}_{pers} por

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

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

- Ou por

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

Frustum Simétrico de Projeção Perspectiva

- Assim temos

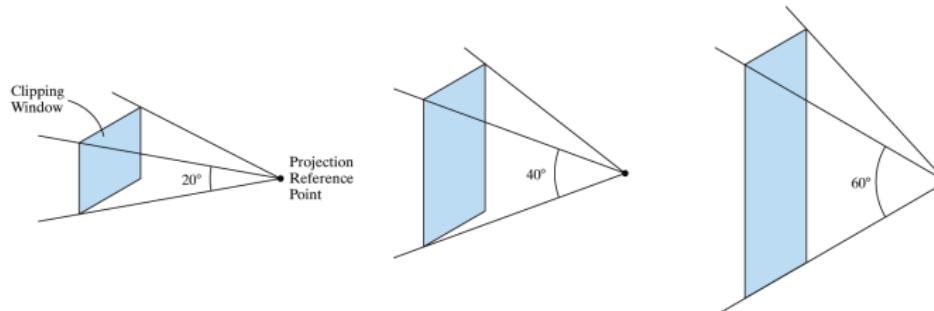
$$\mathbf{M}_{pers} = \begin{bmatrix} \frac{height}{2} \cot\left(\frac{\theta}{2}\right) & 0 & -x_{prp} & x_{prp}z_{prp} \\ 0 & \frac{height}{2} \cot\left(\frac{\theta}{2}\right) & -y_{prp} & y_{prp}z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

- 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

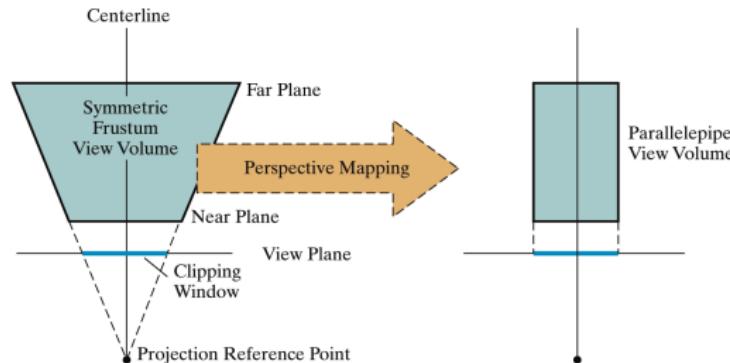
- Diminuir o ângulo do campo de visão diminui a janela de recorte
 - Mover o ponto de projeção para longe do plano de visão
 - Zoom-in de uma pequena região da cena
- Aumentar o ângulo do campo de visão aumenta a janela de recorte
 - Mover o ponto de projeção para próximo do plano de visão
 - Zoom-out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



Frustum Simétrico de Projeção Perspectiva

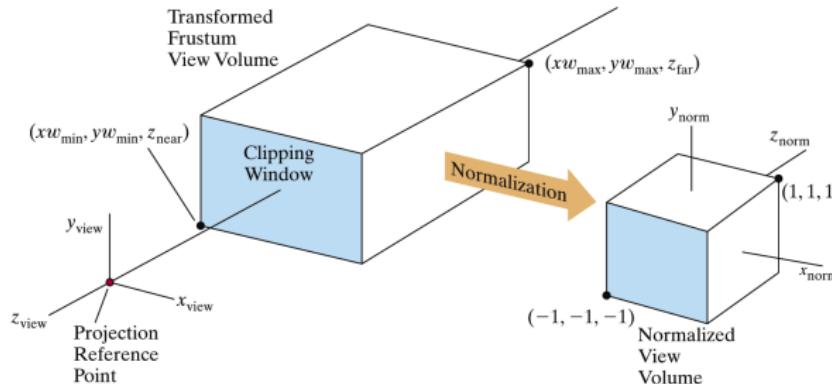
Observação Importante

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



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

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

$$\mathbf{M}_{normpers} = \begin{bmatrix} (z_{prp} - z_{vp})s_x & 0 & (-x_{prp})s_x & (x_{prp}z_{prp})s_x \\ 0 & (z_{prp} - z_{vp})s_y & (-y_{prp})s_y & (y_{prp}z_{prp})s_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- Considerando que o centro de projeção está na origem $(x_{ppr}, y_{ppr}, z_{ppr}) = (0, 0, 0)$ e que o plano de visão está sobre o plano de recorte $near z_{vp} = z_{near}$, então podemos reescrever a matriz

$$\mathbf{M}_{normpers} = \begin{bmatrix} -z_{near} \cdot s_x & 0 & 0 & 0 \\ 0 & -z_{near} \cdot s_y & 0 & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

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 nas coordenadas de projeção

$$x_p = \frac{x_h}{h} = \frac{-z_{near} \cdot s_x \cdot x}{-z}$$

$$y_p = \frac{y_h}{h} = \frac{-z_{near} \cdot s_y \cdot y}{-z}$$

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

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 as menores da janela de recorte e $(x_p, y_p, z_p) = (1, 1, 1)$ quando essas forem as maiores
- Como a linha central do *frustum* intersecta o plano em $(0, 0, z_{vp})$ é possível expressar esses valores como

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

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

Transformação de Projeção Perspectiva Normalizada

- Assim, resolvendo as equações anteriores buscando transformar $(x, y, z) = (xw_{min}, yw_{min}, z_{near})$ em $(x_p, y_p, z_p) = (-1, -1, -1)$ e $(x, y, z) = (xw_{max}, yw_{max}, z_{far})$ em $(x_p, y_p, z_p) = (1, 1, 1)$ obtemos

$$s_x = \frac{2}{width}, s_y = \frac{2}{height}$$

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

Transformação de Projeção Perspectiva Normalizada

- Substituindo s_x , s_y , s_z e t_z encontramos a matriz de projeção perspectiva normalizada

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

$$\begin{bmatrix} -z_{near} \cdot \frac{2}{width} & 0 & 0 & 0 \\ 0 & -z_{near} \cdot \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & -\frac{2 \cdot z_{near} \cdot z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Transformação de Projeção Perspectiva Normalizada

- É possível reescrever essa matriz em termos do ângulo θ

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

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

Transformação de Projeção Perspectiva Normalizada

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

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

- 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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

$M_{normviewvol,3Dscreen} =$

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

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 $(xv_{max}, yv_{max}, 1)$
- 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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 gl.glMatrixMode(GL.GL_PROJECTION);
```

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

```
1 gl.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 z_{view}
 - Por exemplo, se $dfar = 55.0$, o plano de recorte far estará na posição $z_{far} = -55.0$
 - Quaisquer valores podem ser atribuídos, desde que $dnear < dfar$
- Por padrão, a *OpenGL* usa os seguintes valores para essa função

```
1 gl.g1Ortho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

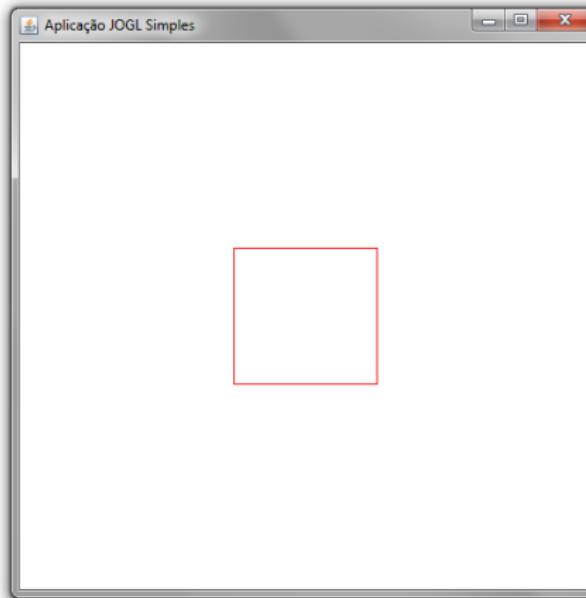
Função de Projeção Ortogonal OpenGL

```
1  public class Renderer implements GLEventListener {  
2  
3      public static void main(String[] args) {  
4          //acelera o rendering  
5          GLCapabilities caps = new GLCapabilities();  
6          caps.setDoubleBuffered(true);  
7          caps.setHardwareAccelerated(true);  
8  
9          //cria o painel e adiciona um ouvinte GLEventListener  
10         GLCanvas canvas = new GLCanvas(caps);  
11         canvas.addGLEventListener(new Renderer());  
12  
13         //cria uma janela e adiciona o painel  
14         JFrame frame = new JFrame("Aplicação JOGL Simples");  
15         frame.getContentPane().add(canvas);  
16         frame.setSize(500, 500);  
17         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
18  
19         //inicializa o sistema e chama display() a 60 fps  
20         Animator animator = new FPSAnimator(canvas, 60);  
21         frame.setLocationRelativeTo(null);  
22         frame.setVisible(true);  
23         animator.start();  
24     }  
25  
26     public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {  
27     }  
28  
29     public void displayChanged(GLAutoDrawable drawable, boolean modeChanged, boolean deviceChanged) {  
30     }  
31     ...  
32 }  
33  
34 }
```

Função de Projeção Ortogonal OpenGL

```
1 public class Renderer implements GLEventListener {  
2     ...  
3  
4     public void init(GLAutoDrawable drawable) {  
5         GL gl = drawable.getGL();  
6  
7         gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo  
8         gl.glEnable(GL.GL_DEPTH_TEST); //habilita o teste de profundidade  
9  
10        gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix é a de projeção  
11        gl.glLoadIdentity(); //carrega a matrix de identidade  
12        gl.glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0); //define uma projeção ortogonal  
13    }  
14  
15    public void display(GLAutoDrawable drawable) {  
16        GL gl = drawable.getGL();  
17        GLUT glut = new GLUT();  
18  
19        //limpa o buffer  
20        gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);  
21  
22        //define que a matrix é a de modelo  
23        gl.glMatrixMode(GL.GL_MODELVIEW);  
24  
25        //desenha um cubo  
26        gl glColor3f(1.0f, 0.0f, 0.0f);  
27        glut.glutWireCube(1.0f);  
28  
29        //força o desenho das primitivas  
30        gl.glFlush();  
31    }  
32}
```

Função de Projeção Ortogonal OpenGL



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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - **Transformação de Visão OpenGL**
 - Projeção Perspectiva Simétrica 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 gl.glMatrixMode(GL.GL_MODELVIEW);
```

Função de Transformação de Visão OpenGL

- Para especificar os parâmetros de visão usamos a função

```
1 glu.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* $\mathbf{V} = (Vx, Vy, Vz)$

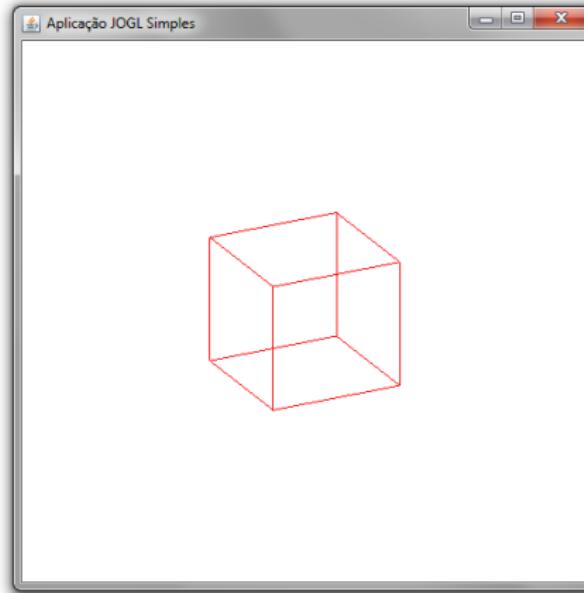
Função de Transformação de Visão OpenGL

- 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
 - $\mathbf{P}_0 = (0, 0, 0)$
 - $\mathbf{P}_{ref} = (0, 0, -1)$
 - $\mathbf{V} = (0, 1, 0)$

Função de Transformação de Visão OpenGL

```
1 public void display(GLAutoDrawable drawable) {  
2     GL gl = drawable.getGL();  
3     GLUT glut = new GLUT();  
4     GLU glu = new GLU();  
5  
6     //limpa o buffer  
7     gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);  
8  
9     //define que a matrix é a de modelo  
10    gl.glMatrixMode(GL.GL_MODELVIEW);  
11    gl.glLoadIdentity(); //carrega a matrix de identidade  
12    glu.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    //desenha um cubo  
17    gl glColor3f(1.0f, 0.0f, 0.0f);  
18    glut	glutWireCube(1.0f);  
19  
20    //força o desenho das primitivas  
21    gl.glFlush();  
22 }
```

Função de Transformação de Visão OpenGL



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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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 glu.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 $d_{far} > 0$

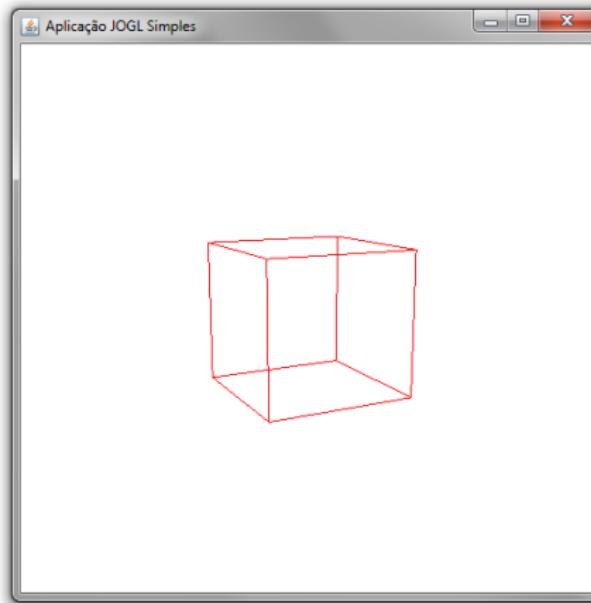
Função para Projeção Perspectiva Simétrica OpenGL

```
1 public void init(GLAutoDrawable drawable) {  
2     GL gl = drawable.getGL();  
3     GLU glu = new GLU();  
4  
5     gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo  
6     gl.glEnable(GL.GL_DEPTH_TEST); //habilita o teste de profundidade  
7  
8     gl.glMatrixMode(GL.GL_PROJECTION); //define que a matrix é a de projeção  
9     gl.glLoadIdentity(); //carrega a matrix de identidade  
10    glu.gluPerspective(45.0, 1.0, 0.1, 10.0); //define uma projeção perspectiva  
11 }
```

Função para Projeção Perspectiva Simétrica OpenGL

```
1 public void display(GLAutoDrawable drawable) {
2     GL gl = drawable.getGL();
3     GLUT glut = new GLUT();
4     GLU glu = new GLU();
5
6     //limpa o buffer
7     gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
8
9     //define que a matrix é a de modelo
10    gl.glMatrixMode(GL.GL_MODELVIEW);
11    gl.glLoadIdentity(); //carrega a matrix de identidade
12    glu.gluLookAt(4.0, 1.0, 2.0, //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     //desenha um cubo
17     gl glColor3f(1.0f, 0.0f, 0.0f);
18     glut	glutWireCube(1.0f);
19
20     //força o desenho das primitivas
21     gl.glFlush();
22 }
```

Função para Projeção Perspectiva Simétrica OpenGL



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 Perspectivas
 - Transformação de Viewport e Coordenadas de Tela 3D
- 6 Funções OpenGL para Viewing 3D
 - Função de Projeção Ortogonal OpenGL
 - Transformação de Visão OpenGL
 - Projeção Perspectiva Simétrica 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

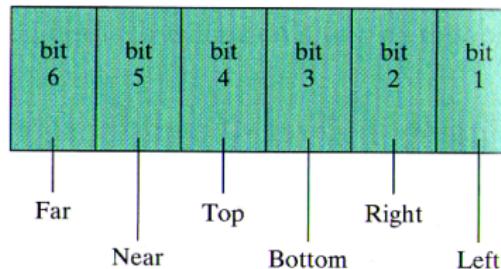
- No *pipeline* de visualização 3D, após as posições terem passado pelas transformações geométricas, de visão e projeção, temos uma representação homogênea

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

- 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

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

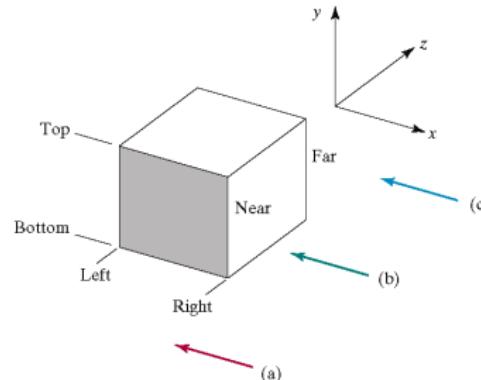
$$h \leq x_h \leq -h, \quad h \leq y_h \leq -h, \quad h \leq z_h \leq -h, \quad \text{se } h < 0$$

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

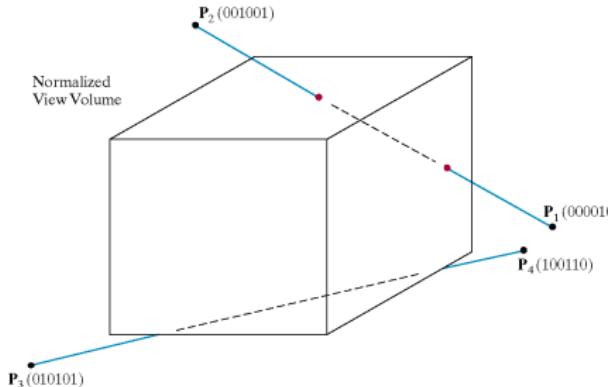
Region Codes
Between Near and Far Planes
(b)

101001	101000	101010
100001	100000	100010
100101	100100	100110

Region Codes
Behind Far Plane
(c)

Recorte de Ponto e Linha 3D

- 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

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

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

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

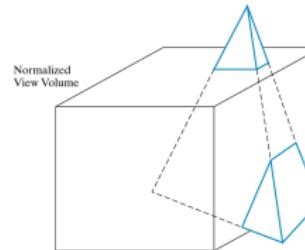
$$h = h_1 + (h_2 - h_1)u$$

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