UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Viewing 3D

Disciplina: Computação Gráfica (BCC35F)
Curso: Ciência da Computação

Prof. Walter T. Nakamura
waltertakashi@utfpr.edu.br

Campo Mourão - PR

Baseados nos materiais elaborados pelas professoras Aretha Alencar (UTFPR) e Rosane Minghim (USP)

Sumário

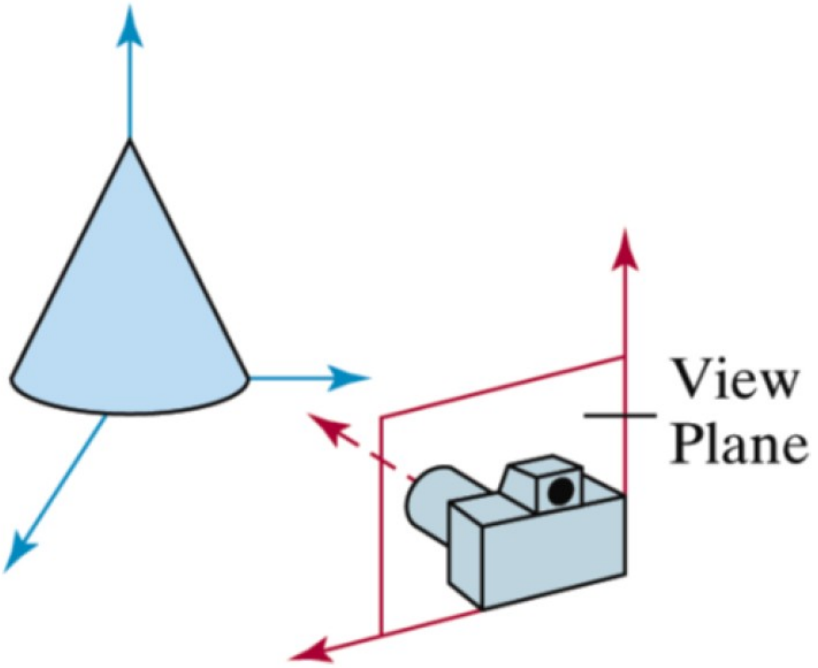
Sumário

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

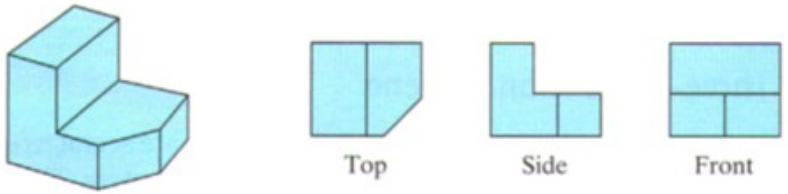
Visualizando uma Cena 3D

- 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

Projeções

- 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

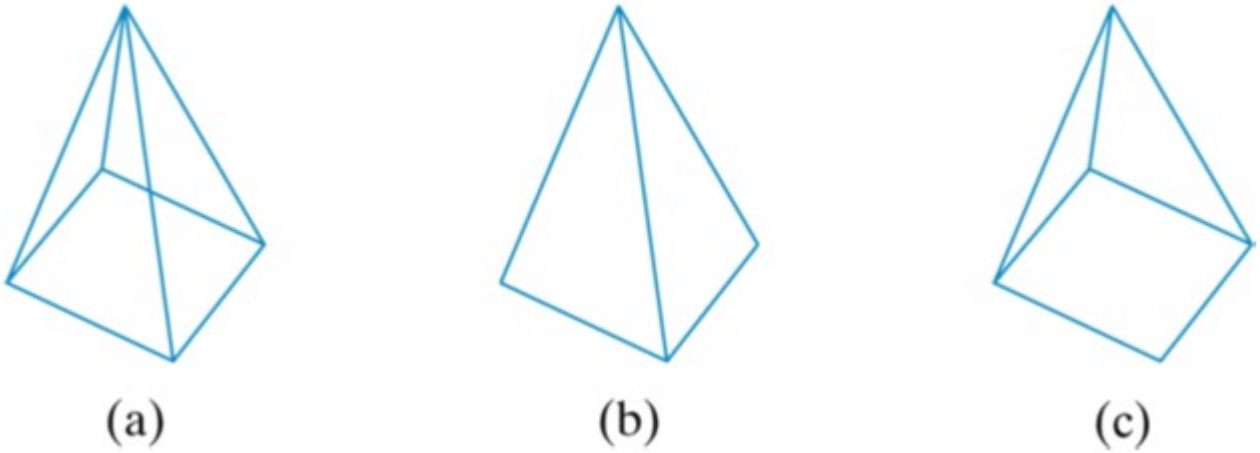
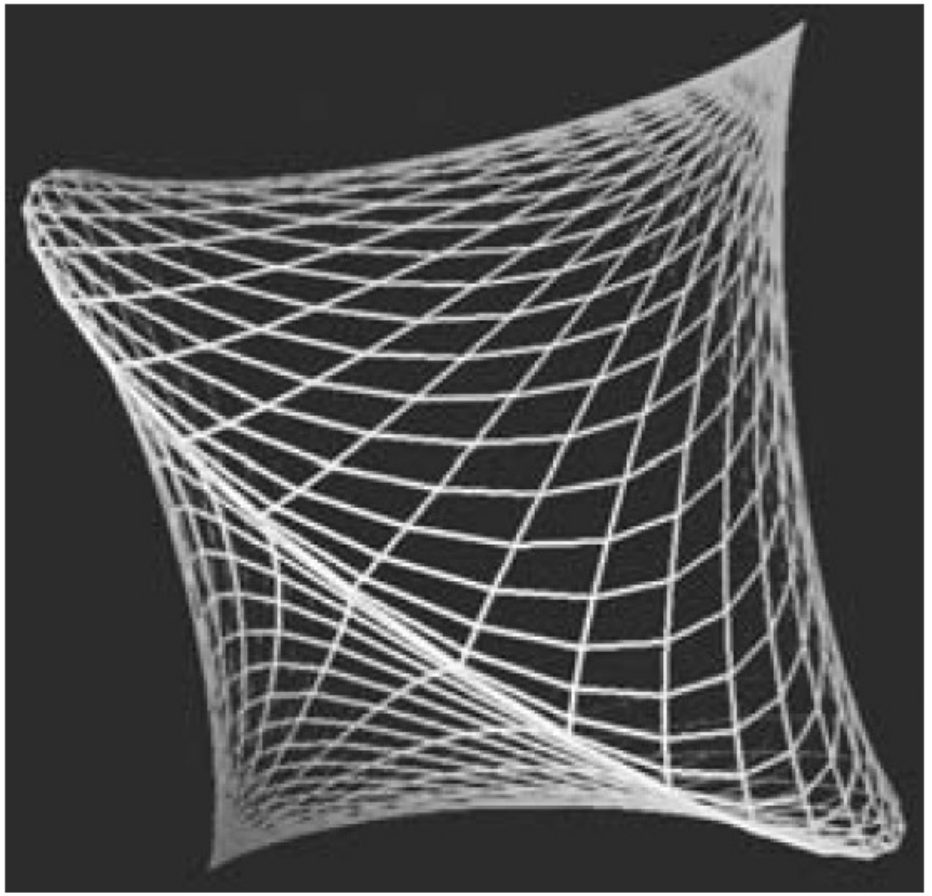


Figura: Problemas de ambiguidade pela falta de informação de profundidade em (a), que pode ser interpretada como (b) ou (c).

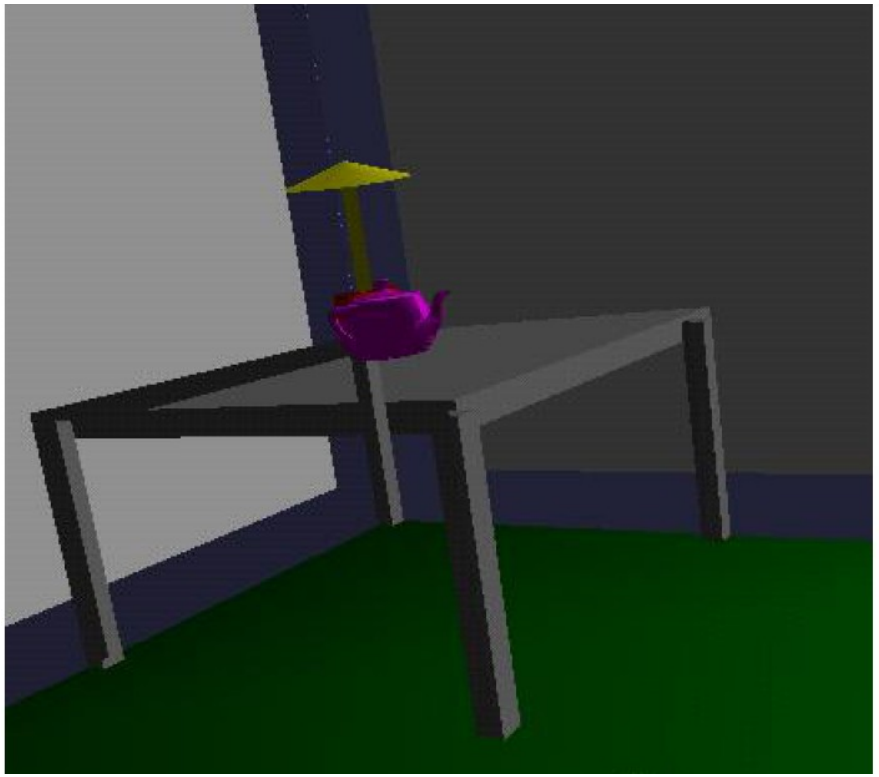
Projeções – Sugestão de Profundidade

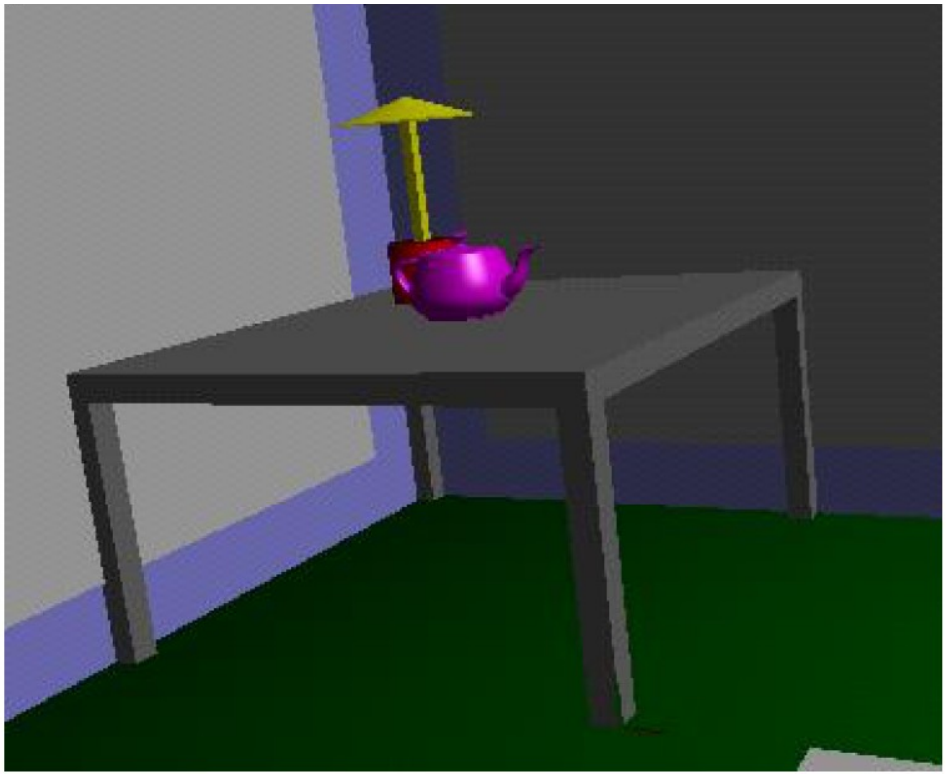
- Uma forma simples de sugerir a profundidade de um objeto representado por wireframe é por meio da **variação da intensidade do brilho** dos segmentos de linha de um objeto



Projeções – 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**





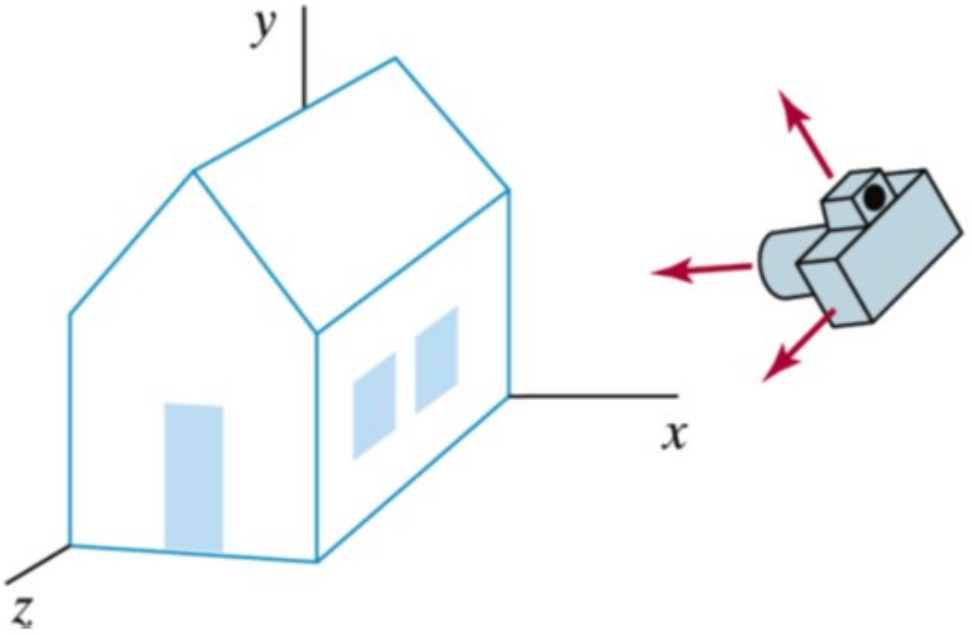
Projeções – Cenas Realísticas

- **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

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.

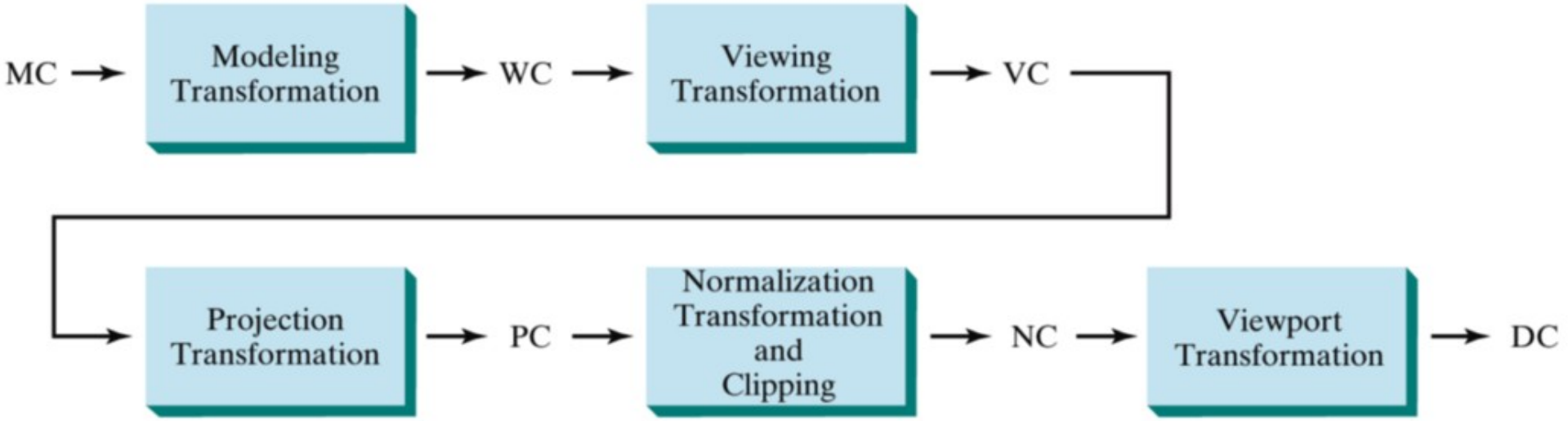


Viewing Pipeline 3D

- 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

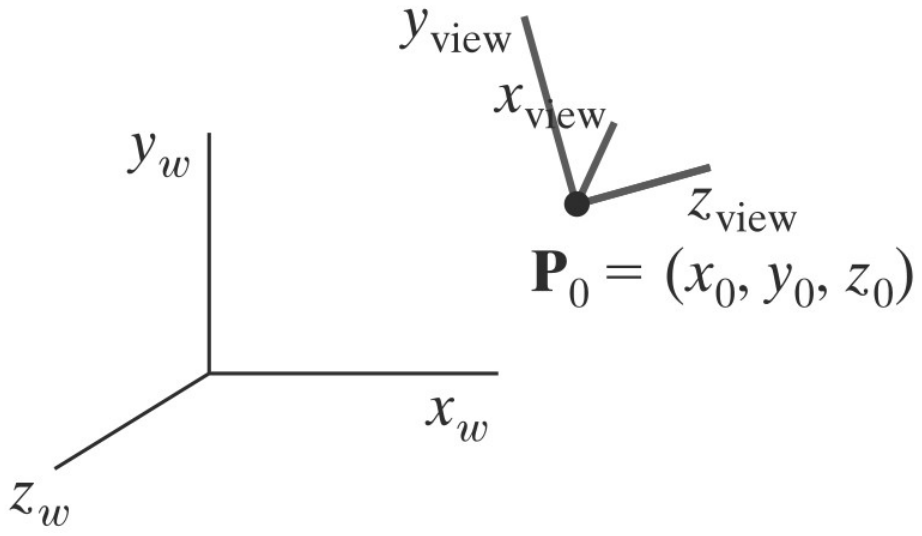
- A **posição de visão**, o **plano de visão**, a **janela de recorte** e os planos de recorte são especificados dentro do sistema de coordenadas de mundo:



Sumário

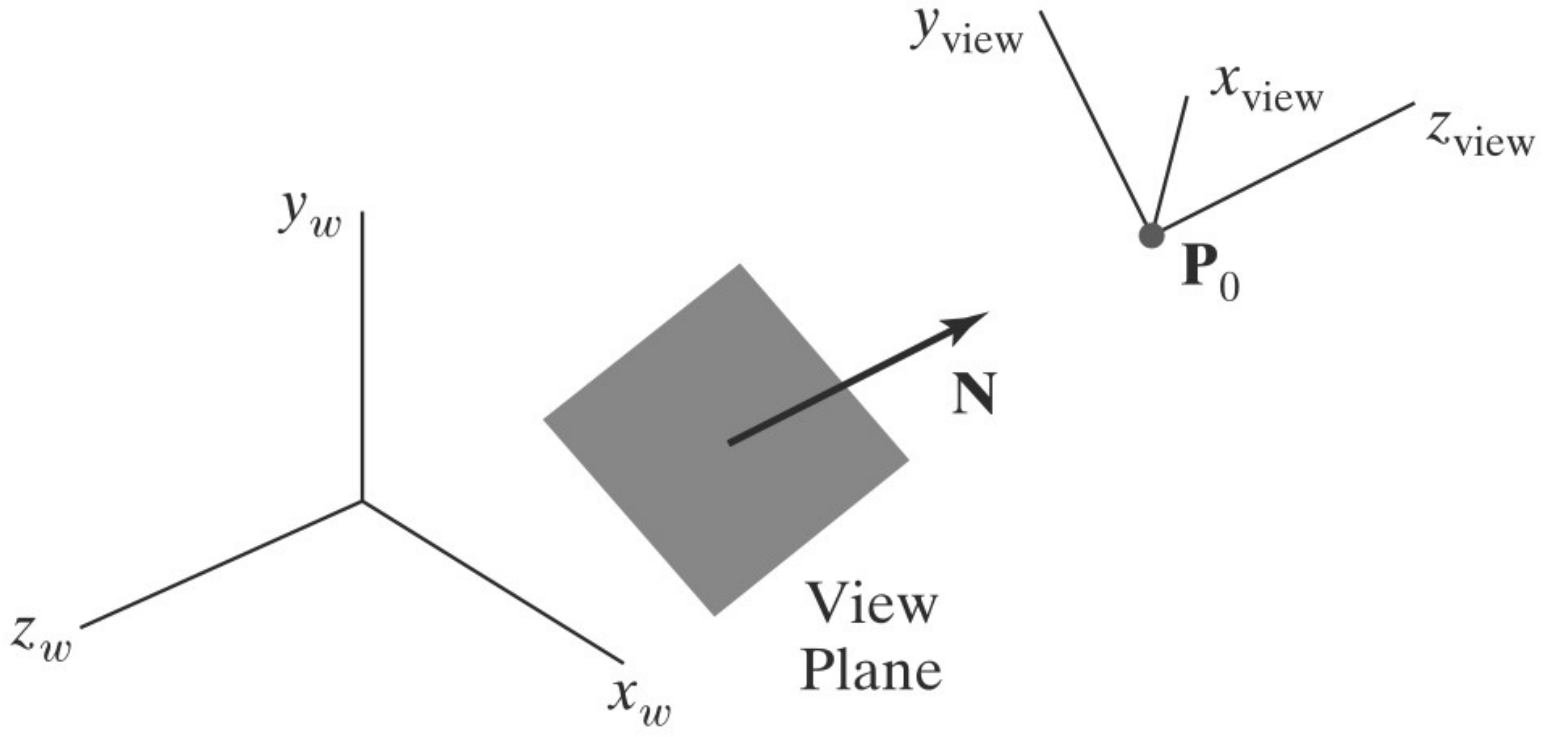
Parâmetros de Coordenadas de Visão 3D

- Para estabelecer um sistema de coordenadas de visão 3D é preciso definir:
  - A origem do sistema $P_0 = (x_0, y_0, z_0)$, chamada de **ponto de visão** (também referido como posição do olho ou da câmera)
  - O **vetor view-up** V, que define a direção de y_{view}, e fornece a orientação da câmera
  - Uma **segunda direção** para um dos eixos coordenados restantes, normalmente o z_{view}, com a direção de visão ao longo desse eixo



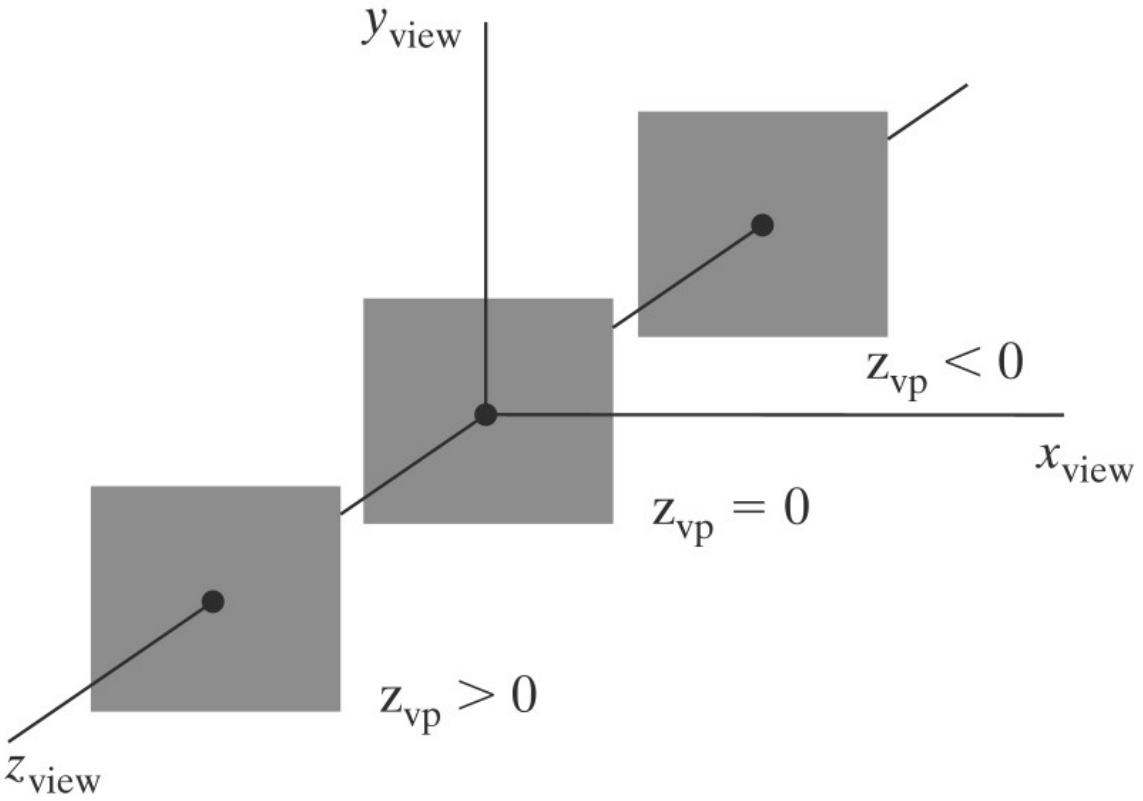
Vetor Normal ao Plano de Visão

- 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**



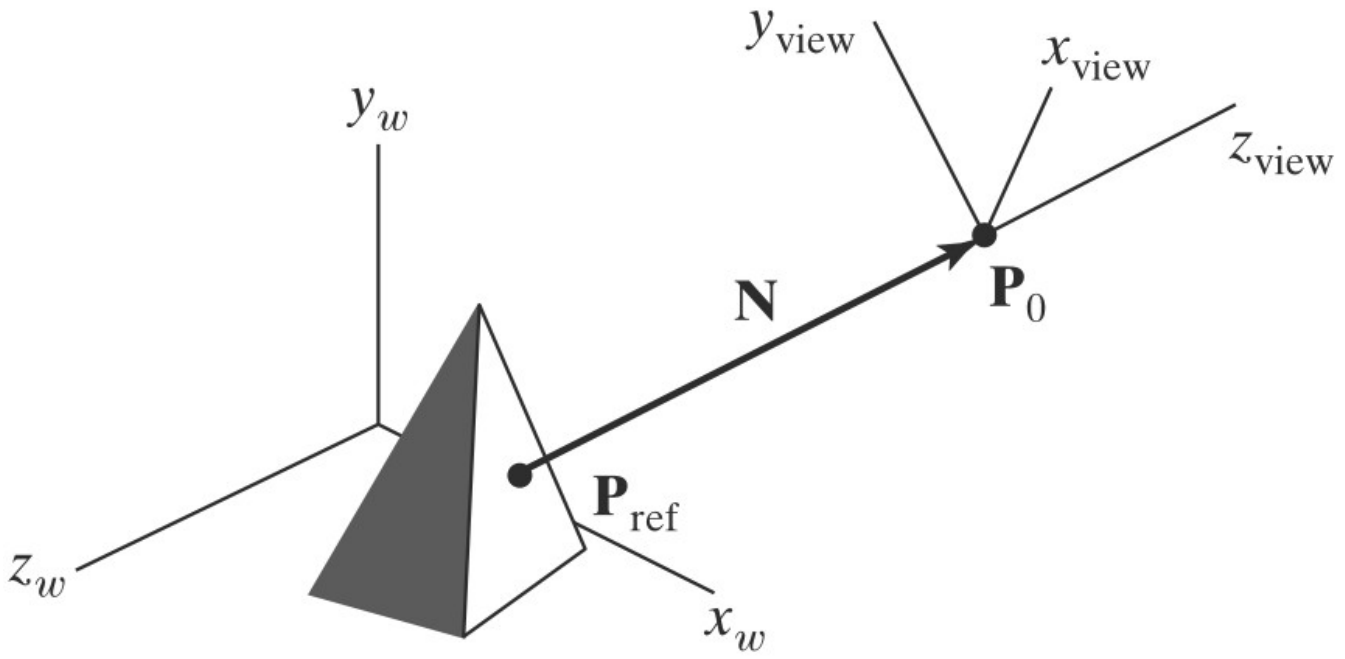
Vetor Normal ao Plano de Visão

- 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** N pode ser especificado de várias formas:
  - A direção de N pode ser definida ao longo da linha partindo de um **ponto de referência** P_{ref} até a origem de visão P_0
    - Nesse caso, esse ponto de referência é denominado **look-at point**, com a direção de visão oposta a de N.



Vetor Normal ao Plano de Visão

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

Vetor Normal ao Plano de Visão

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

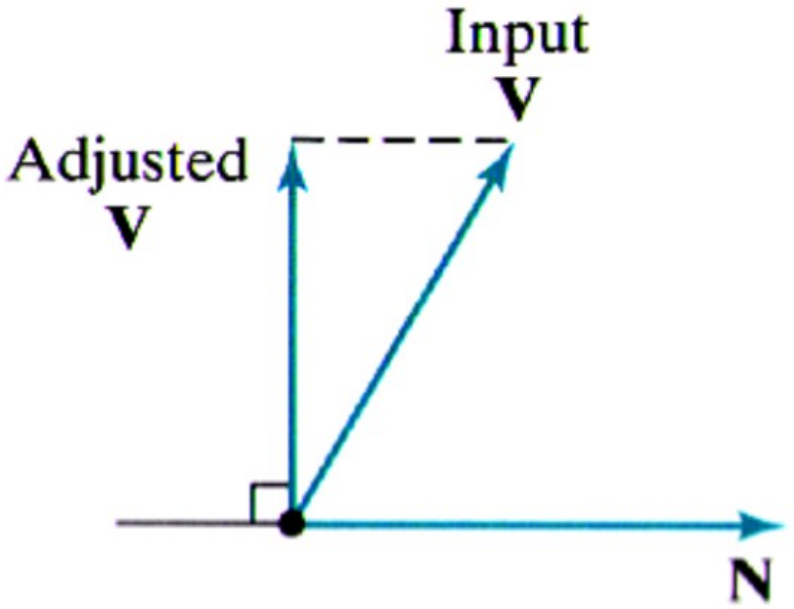


Figura: Ajuste do vetor view-up V para torná-lo perpendicular a N.

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 V e N
  - O produto vetorial de N e U também pode ser usado para ajustar o valor de V ao longo do eixo y_{view}

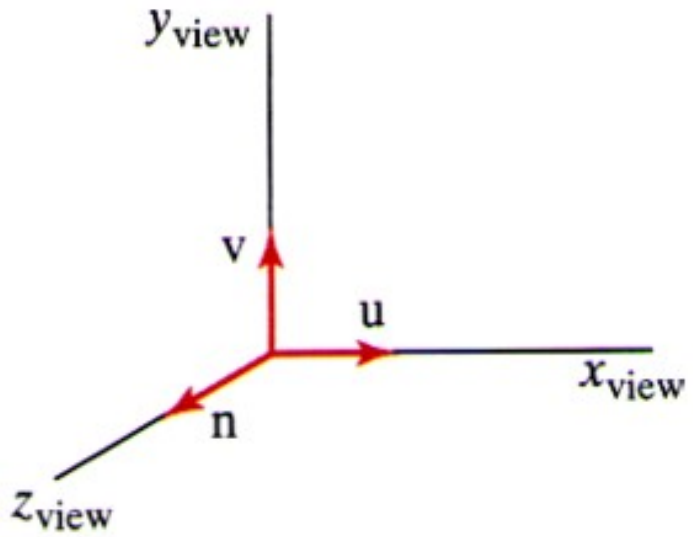
Sistema de Coordenadas de Visão uvn

- Para se obter o sistema de coordenadas uvn fazemos:

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

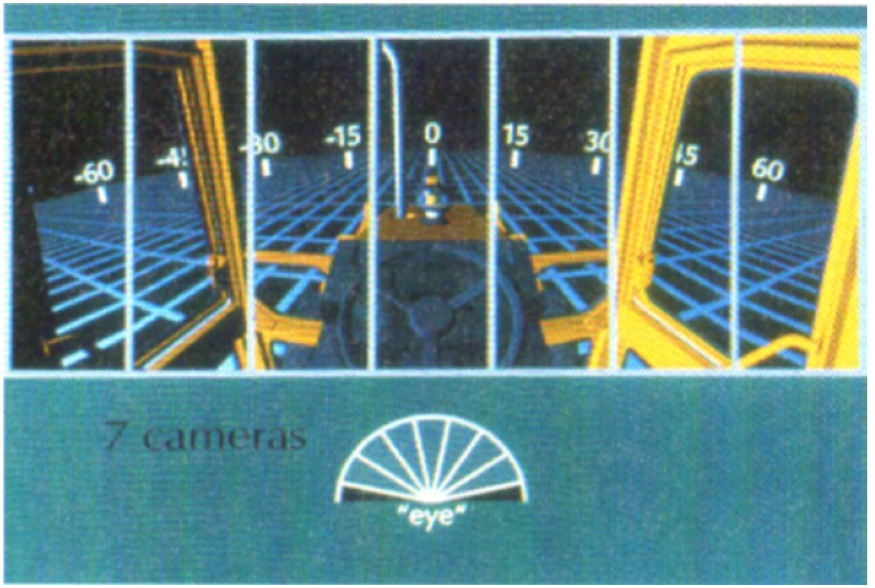
$$u = \frac{V \times n}{|V \times n|} = (u_x, u_y, u_z)$$

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



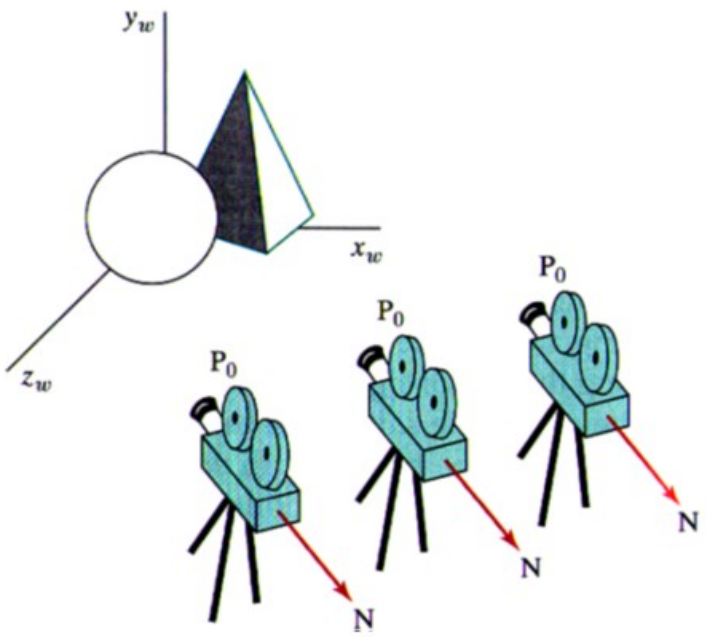
Gerando Efeitos de Visão 3D

- **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 se esqueça de mudar os outros eixos para manter o sistema orientado.

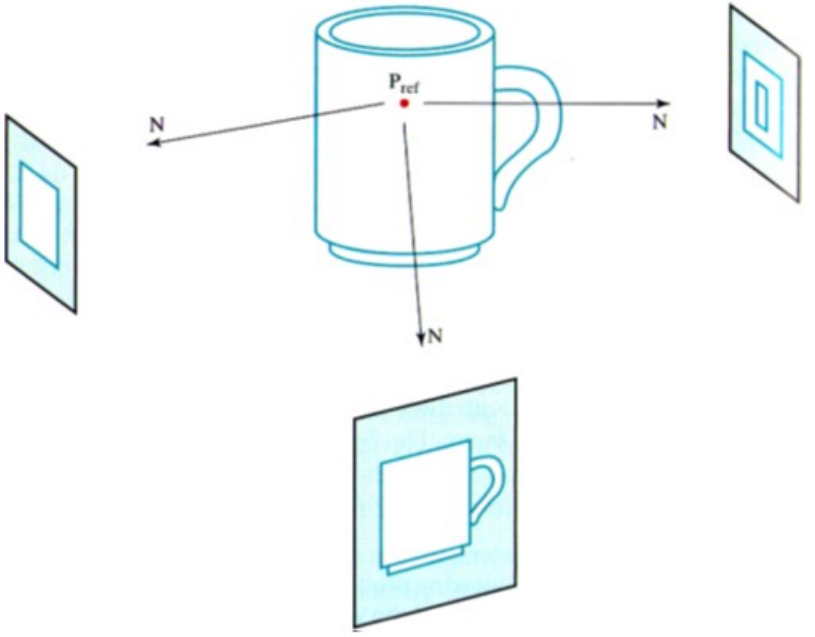


Gerando Efeitos de Visão 3D

- Efeitos de **mover a câmera (pam)** podem ser obtidos fixando 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

Transformação do Sistema de Coordenadas do 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 em dois passos:
  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 do Mundo para o de Visão

- Se a origem do sistema de visão for $P_0 = (x_0, y_0, z_0)$ a matriz de translação será:

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

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

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

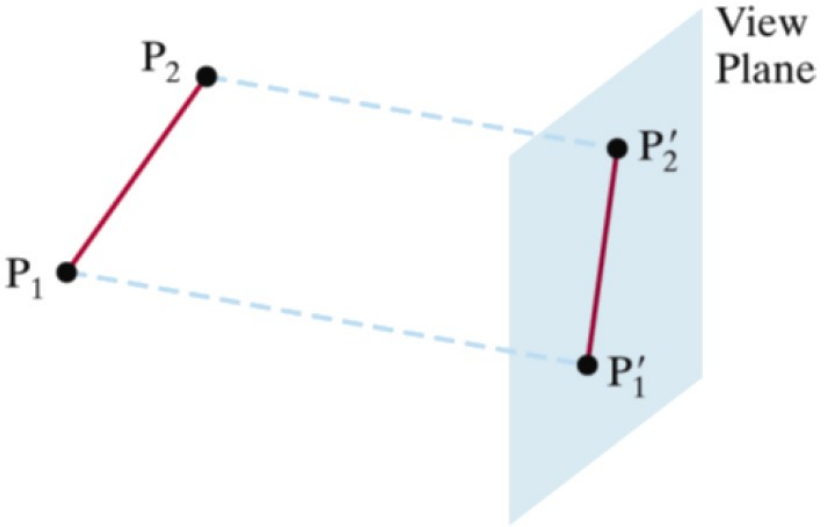
Transformação do Sistema de Coordenadas do Mundo para o de Visão

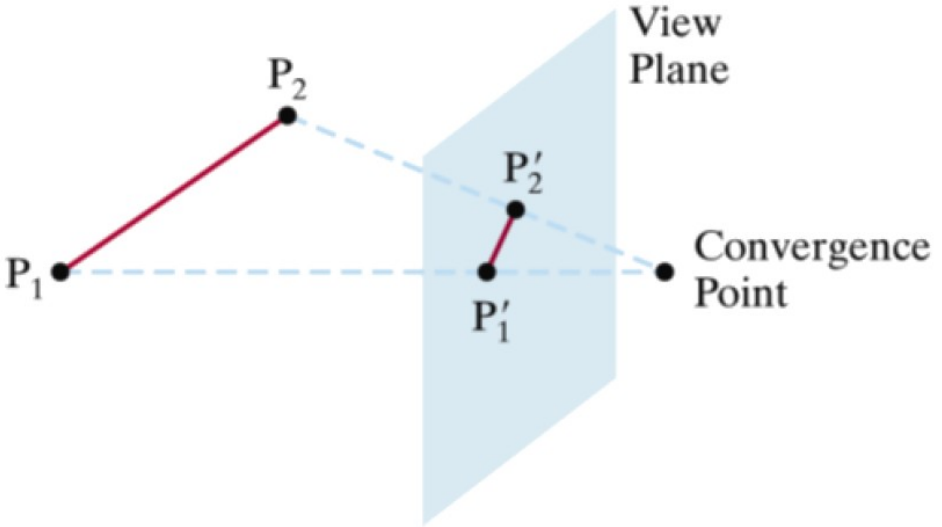
- A matriz de transformação é portanto:

$$M_{WC,VC} = R \cdot T = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação do Sistema de Coordenadas do Mundo para o de Visão

- 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 ao vetor N do plano de projeção
  - **Projeção Perspectiva:** as coordenadas são transferidas para o plano de projeção ao longo de linhas que convergem a um ponto



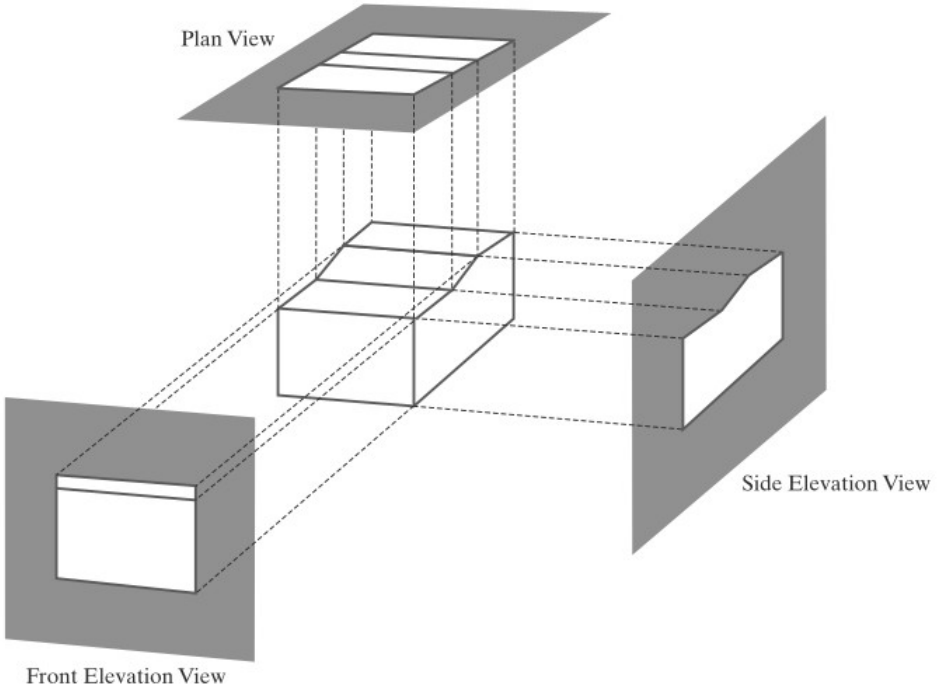


Sumário

Sumário

Projeções Ortogonais

- **Projeção Ortogonal (ou Ortográfica)**
  - Transformação da descrição dos objetos a um plano de projeção ao longo de linhas paralelas ao vetor normal N do plano de projeção
  - Frequentemente usada para produzir a visão frontal, lateral e superior de um objeto
  - Preserva tamanhos e ângulos: útil para desenhos arquitetônicos e de engenharia.



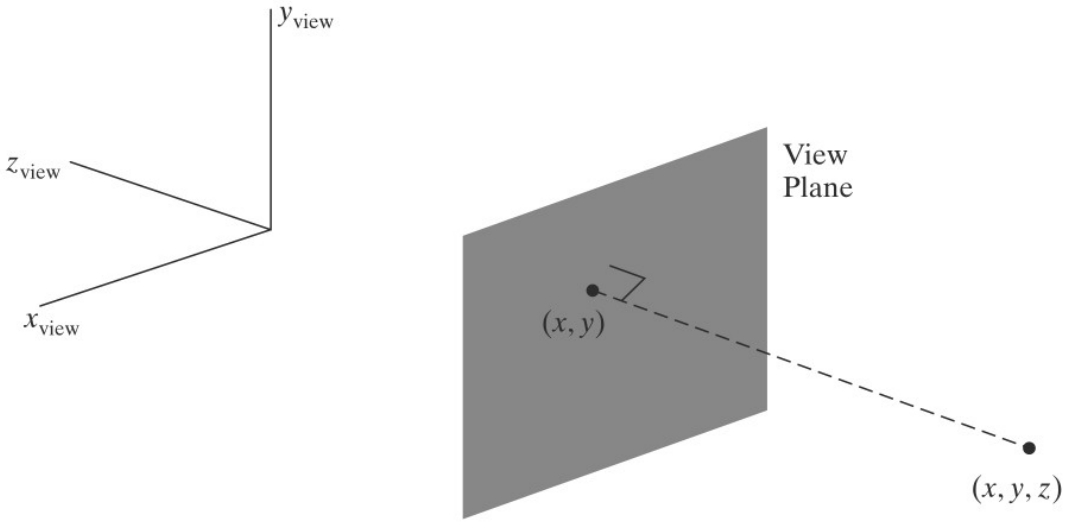
Coordenadas de Projeções Ortogonais

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

$$x_p = x$$

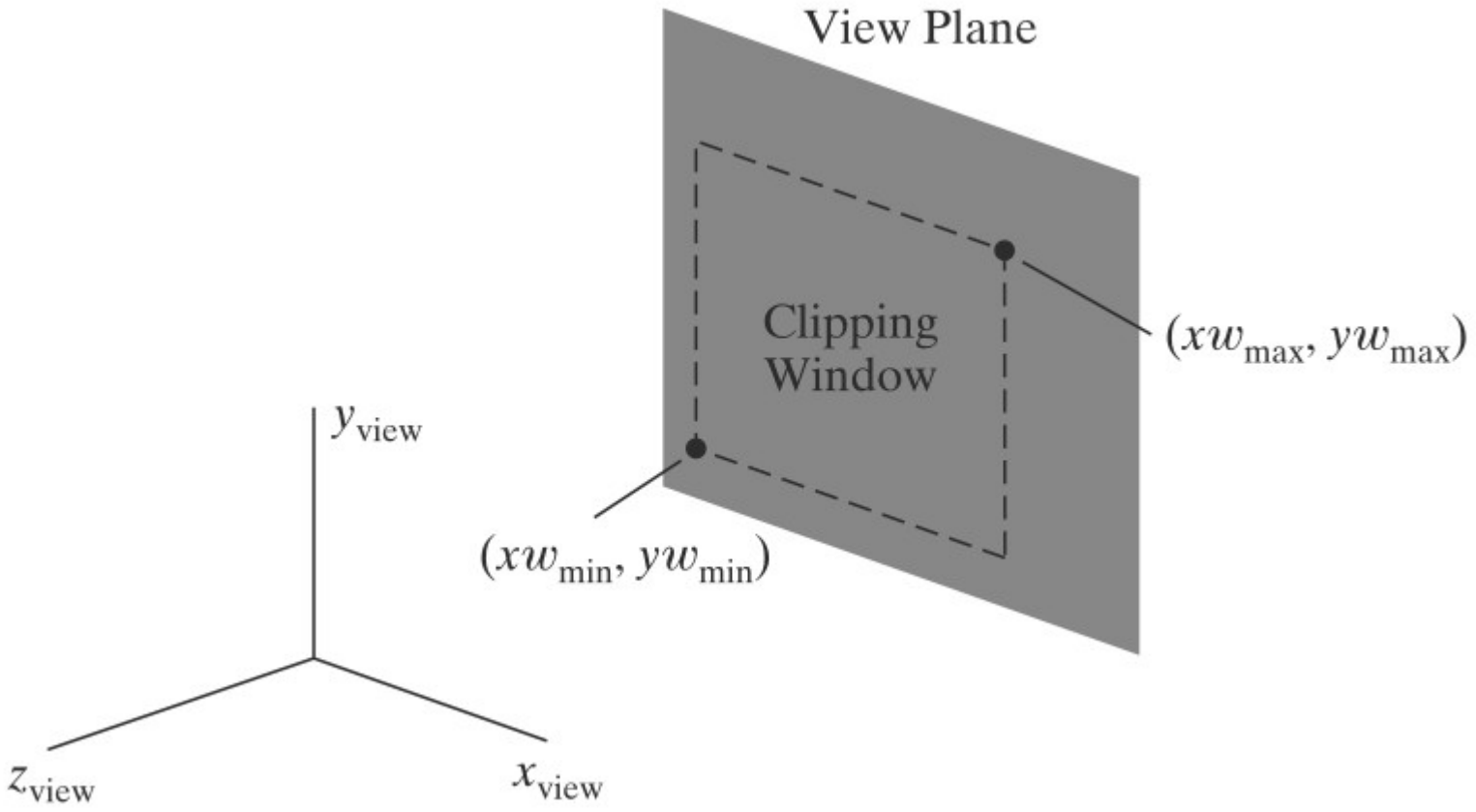
$$y_p = y$$

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



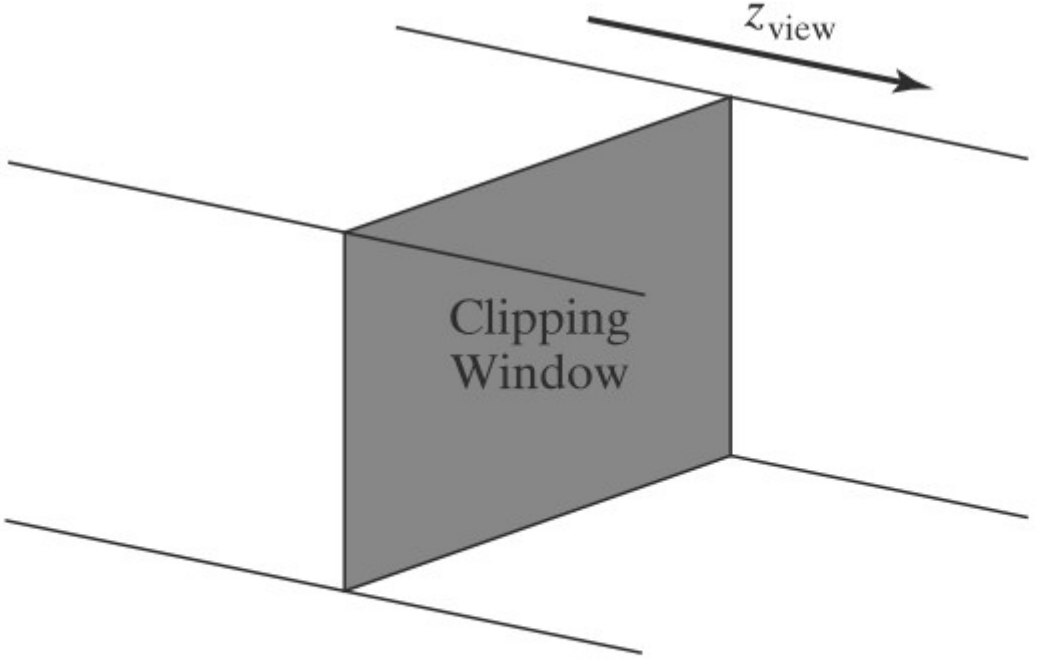
Janela de Recorte e Volume de Projeção Ortogonal

- 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}

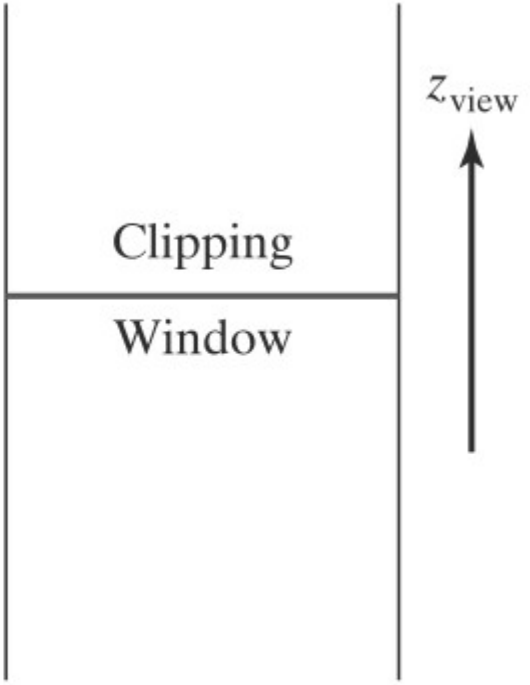


Janela de Recorte e Volume de Projeção Ortogonal

- 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**



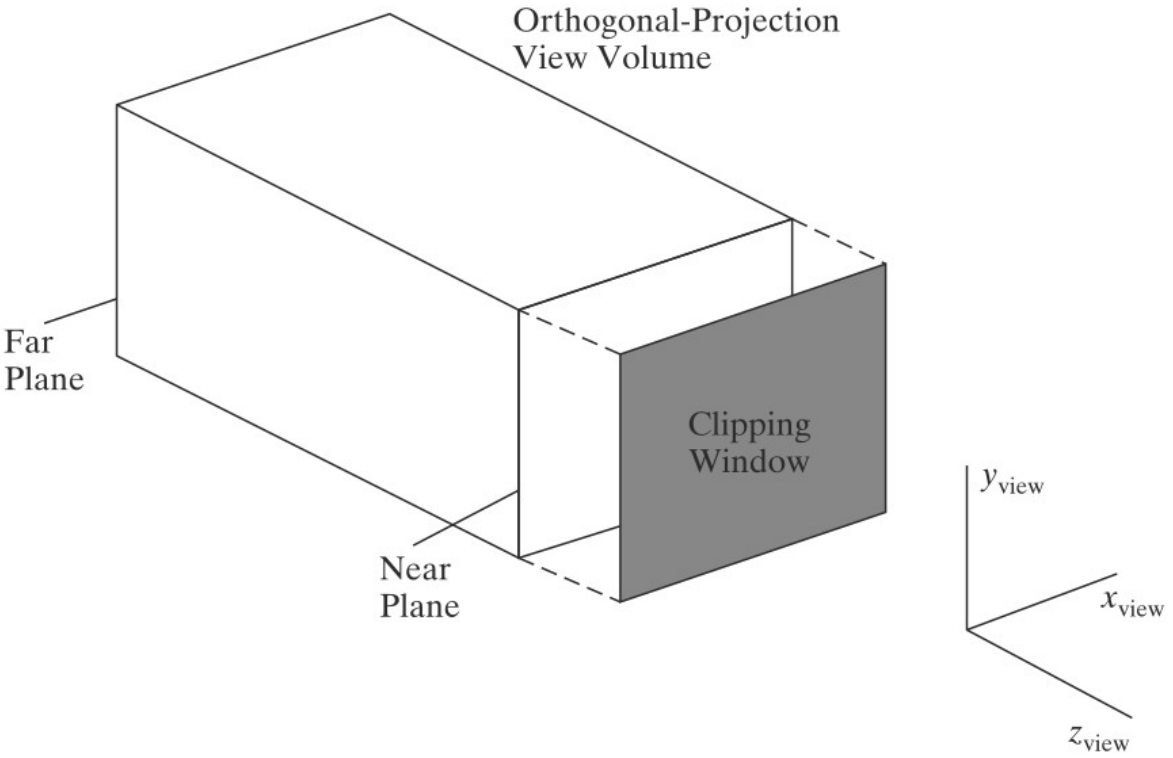
Side View
(a)



Top View
(b)

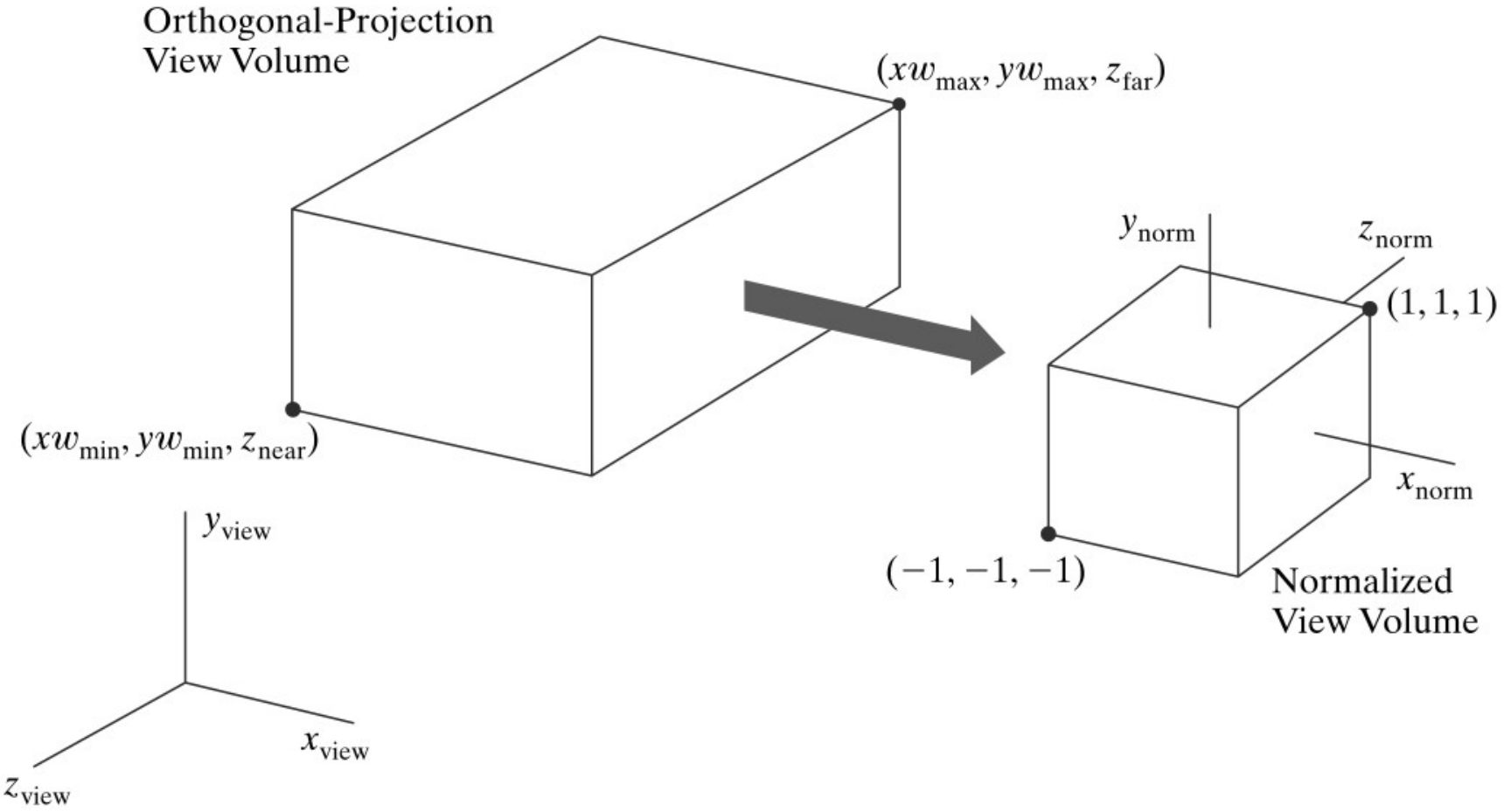
Janela de Recorte e Volume de Projeção Ortogonal

- Para se limitar a extensão desse volume na direção zview, dois planos de fronteira paralelos aos planos de visão, chamados planos de recorte near/far, são considerados:
  - 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

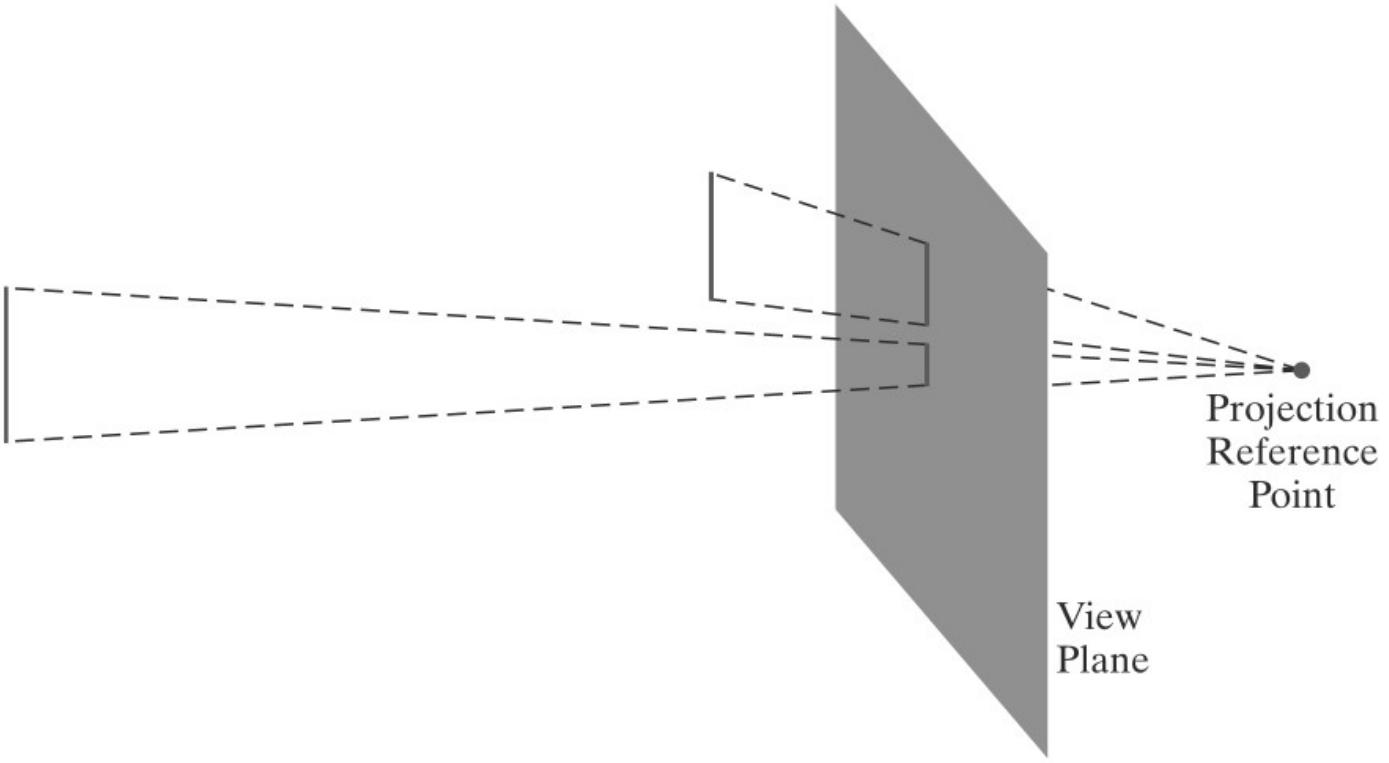
- Como qualquer posição (x, y, z) em uma projeção ortogonal é mapeada para (x, y), as coordenadas dentro do volume de visão são as coordenadas de projeção, assim essas podem ser mapeadas para um volume de visão normalizado sem precisar ser reprojetadas



Sumário

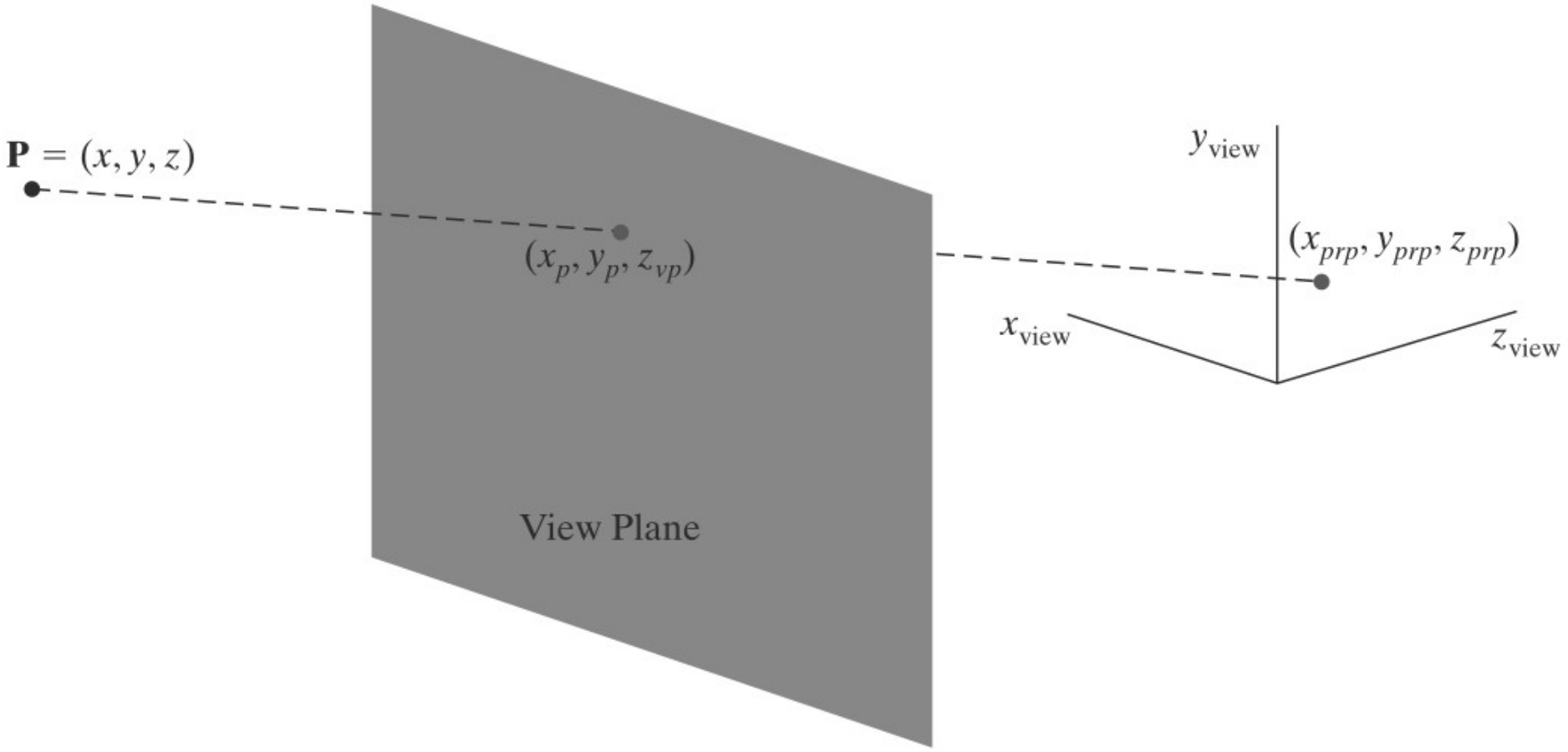
Projeções Perspectivas

- 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_{prp}, y_{prp}, z_{prp}).



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:

$$x' = x - (x - x_{prp})u$$
$$y' = y - (y - y_{prp})u$$
$$z' = z - (z - z_{prp})u$$
$$0 \leqslant u \leqslant 1$$

- A posição (x', y', z') representa qualquer ponto ao longo da linha de projeção
- Para u = 0 estamos em P = (x, y, z). Já para u = 1 temos o centro de projeção (x_{prp}, y_{prp}, z_{prp})

Transformação de Coordenadas de Projeção Perspectiva

- No plano de visão, z' = z_{vp} , então podemos encontrar u substituindo na última equação:

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

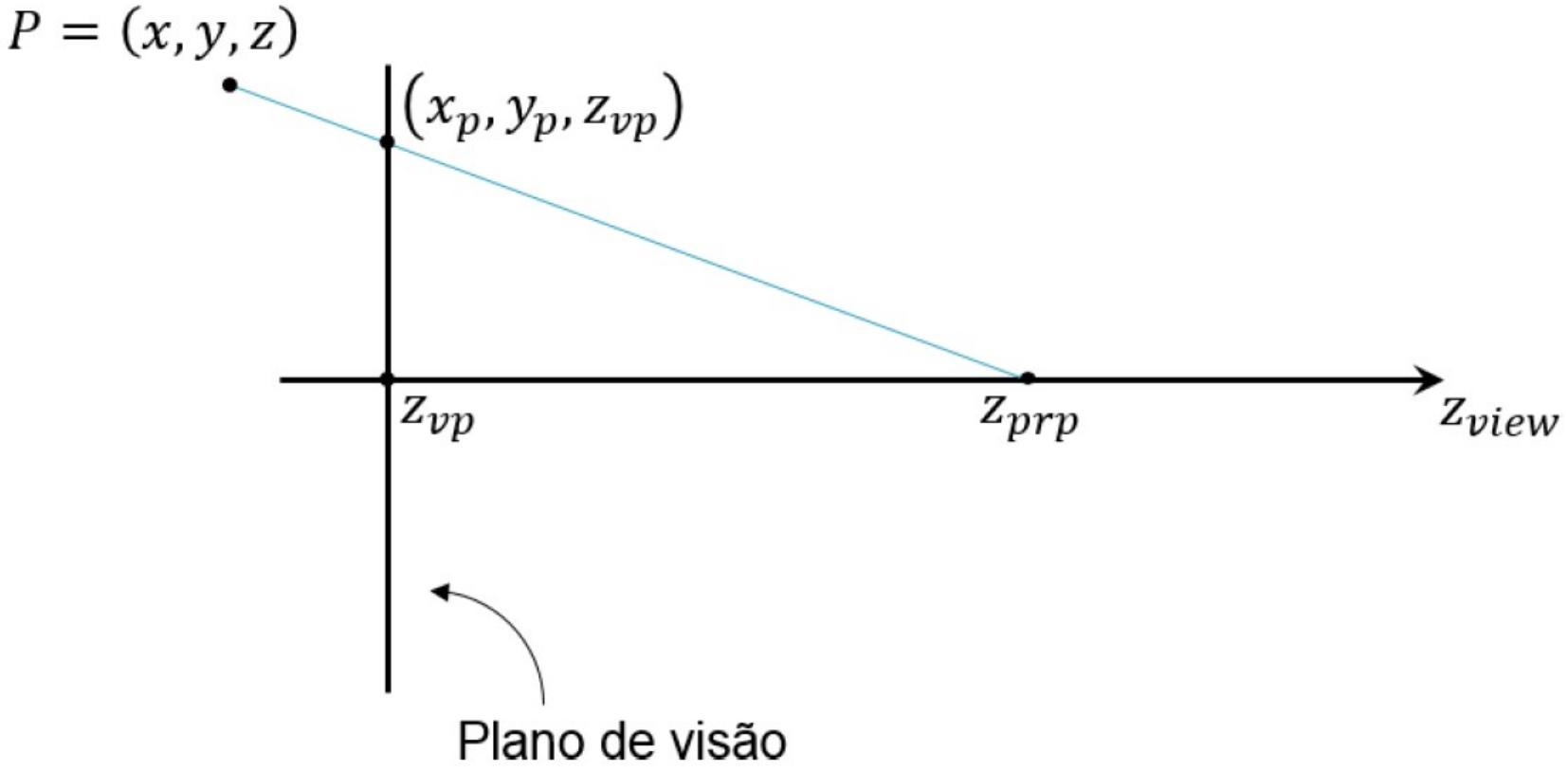
- Substituindo esse valor de u para as equações de x' e y' obtemos as **equações genéricas de projeção perspectiva**:

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

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

Transformação de Coordenadas de Projeção Perspectiva

- Para facilitar o cálculo, o ponto de referência da projeção pode ser limitado a posição ao longo do eixo z_{view}:



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

Transformação de Coordenadas de Projeção Perspectiva

- 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 $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$

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

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

Exemplo

- Dado um ponto P = (0.7, 0.4, −4), que já passou pela transformação do sistema de coordenadas do mundo para o de visão, calcule sua projeção perspectiva dado os seguintes parâmetros:
  - $z_{vp} = -2$
  - $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 3)$

Exemplo – Solução

- Dado um ponto P = (0.7, 0.4, −4), que já passou pela transformação do sistema de coordenadas do mundo para o de visão, calcule sua projeção perspectiva dado os seguintes parâmetros:
  - z_{vp} = -2
  - (x_{prp}, y_{prp}, z_{prp}) = (0, 0, 3)

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = 0.7\left(\frac{3 - (-2)}{3 - (-4)}\right) = 0.5$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = 0.4\left(\frac{3 - (-2)}{3 - (-4)}\right) = 0.2857$$

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 coincidindo com o centro de projeção)

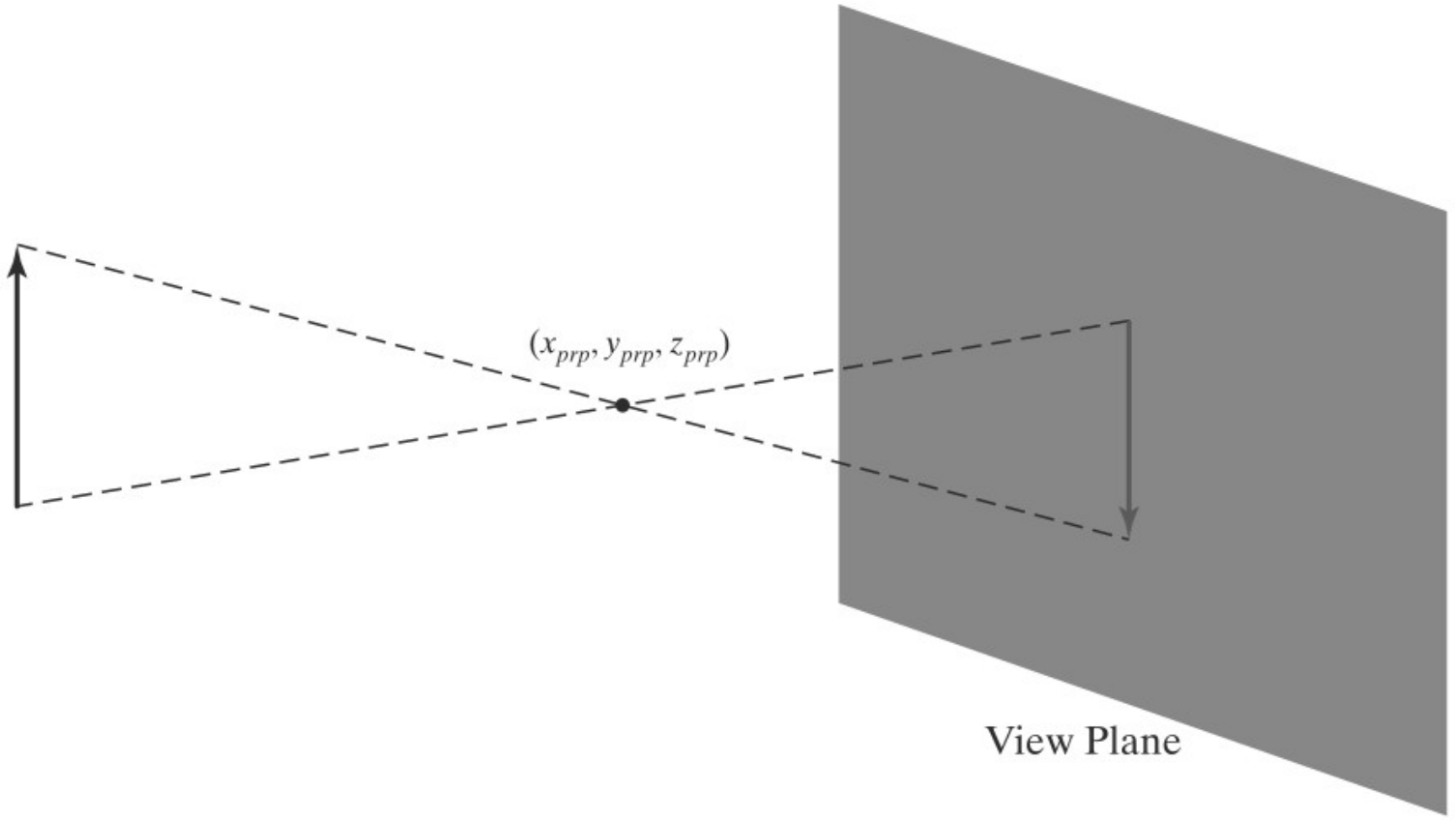


Figura: Os objetos são invertidos se o ponto de referência está entre o plano de visão e a cena.

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.

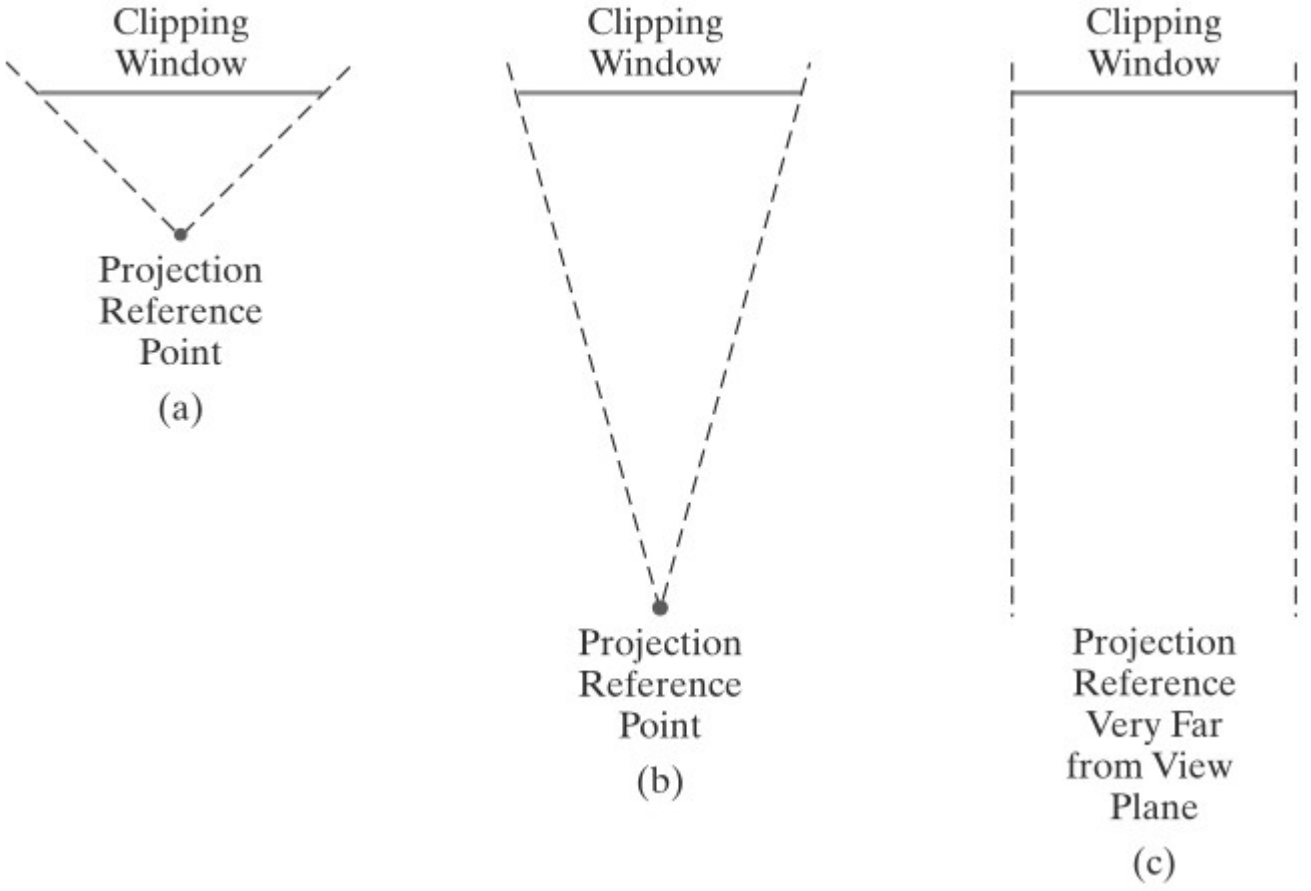


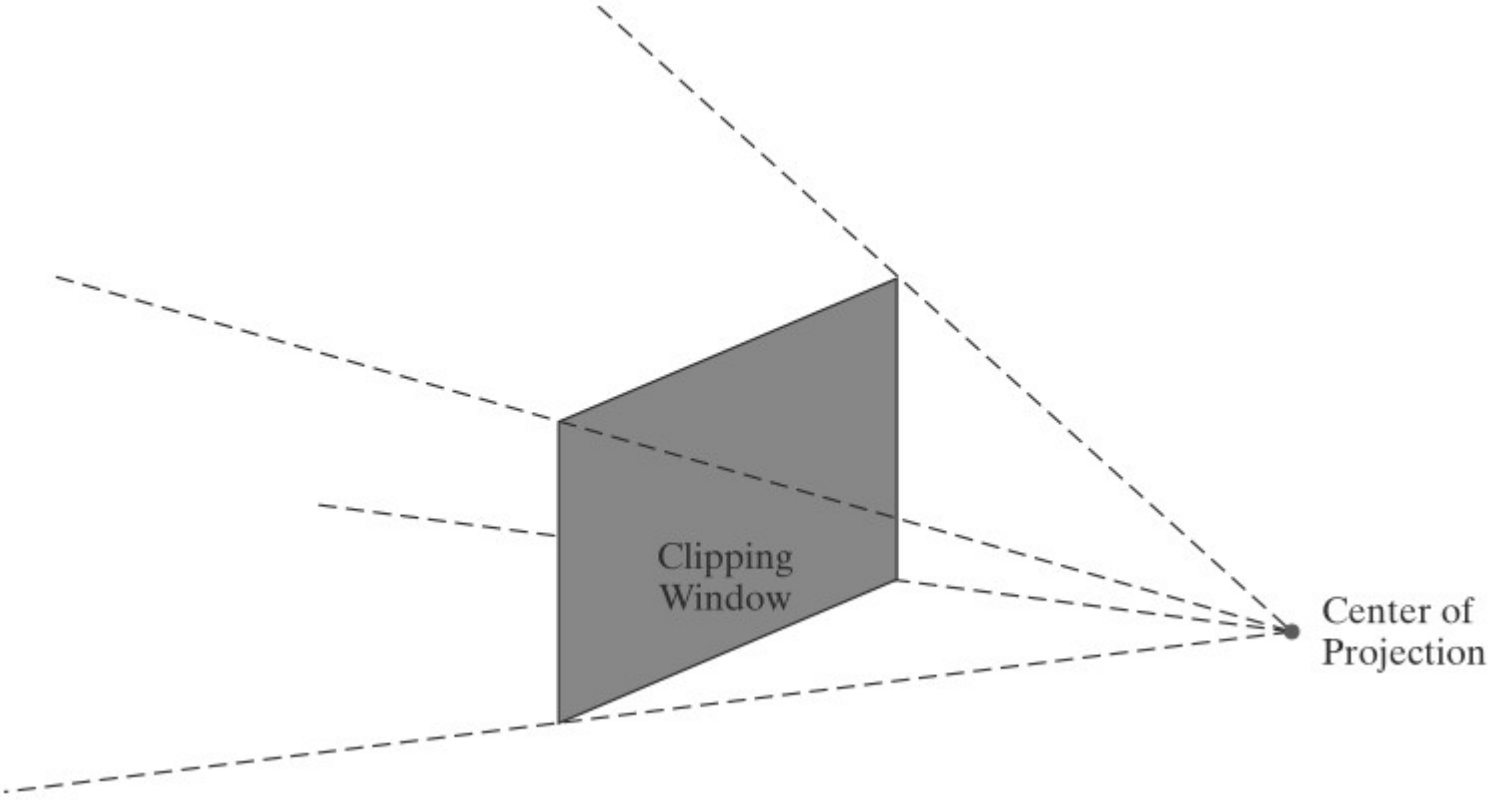
Figura: Se o centro de projeção está próximo ao plano de projeção, os efeitos da perspectiva são enfatizados. Objetos mais próximos ao plano aparecerão muito maiores do que os distantes.

Pontos de Fuga para Projeções Perspectivas

- **Na projeção perspectiva:**
  - Linhas paralelas ao plano de projeção são projetadas como linhas paralelas
  - Linhas paralelas na cena, mas 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**

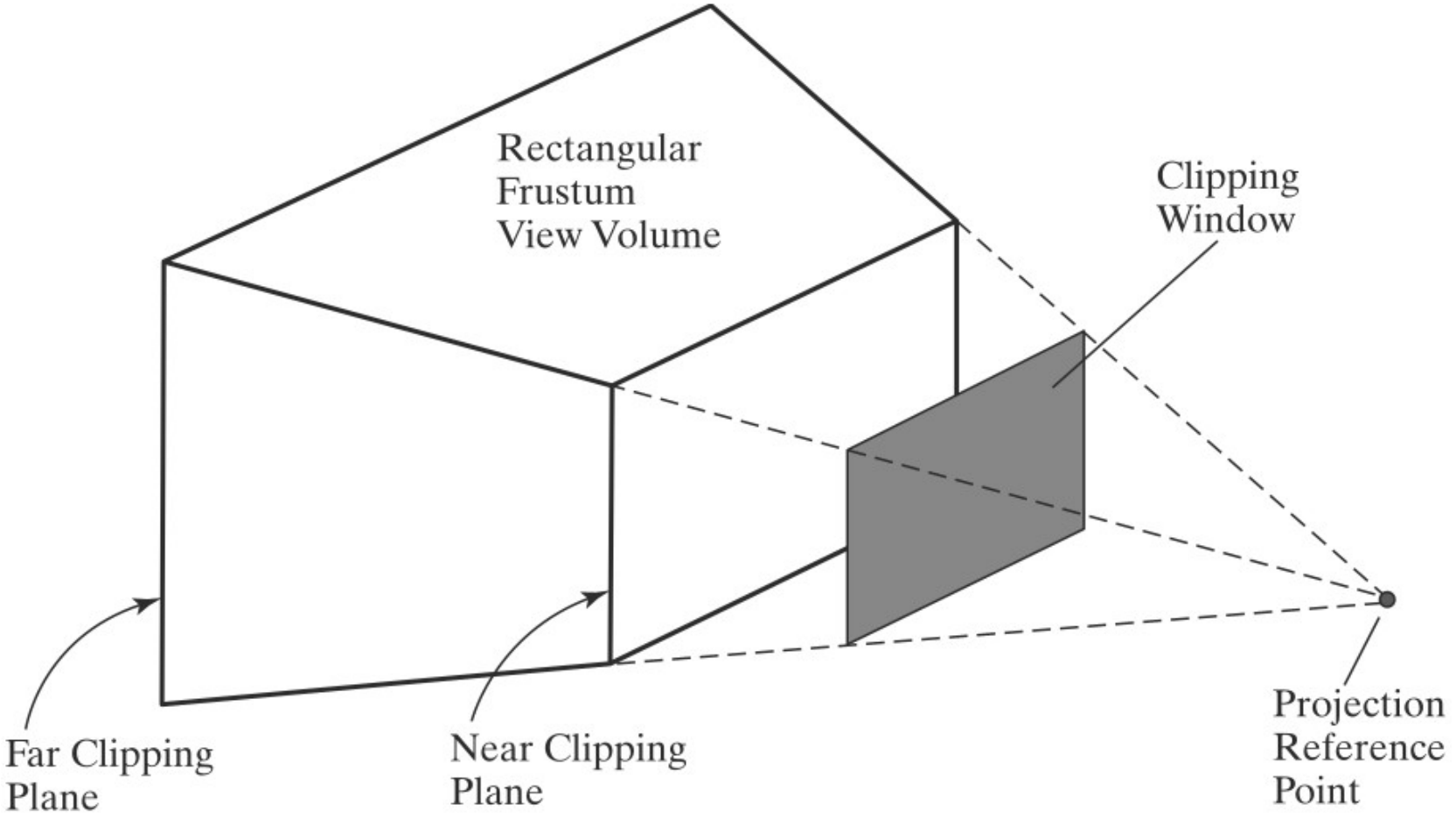
Volume de Projeção Perspectiva

- 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



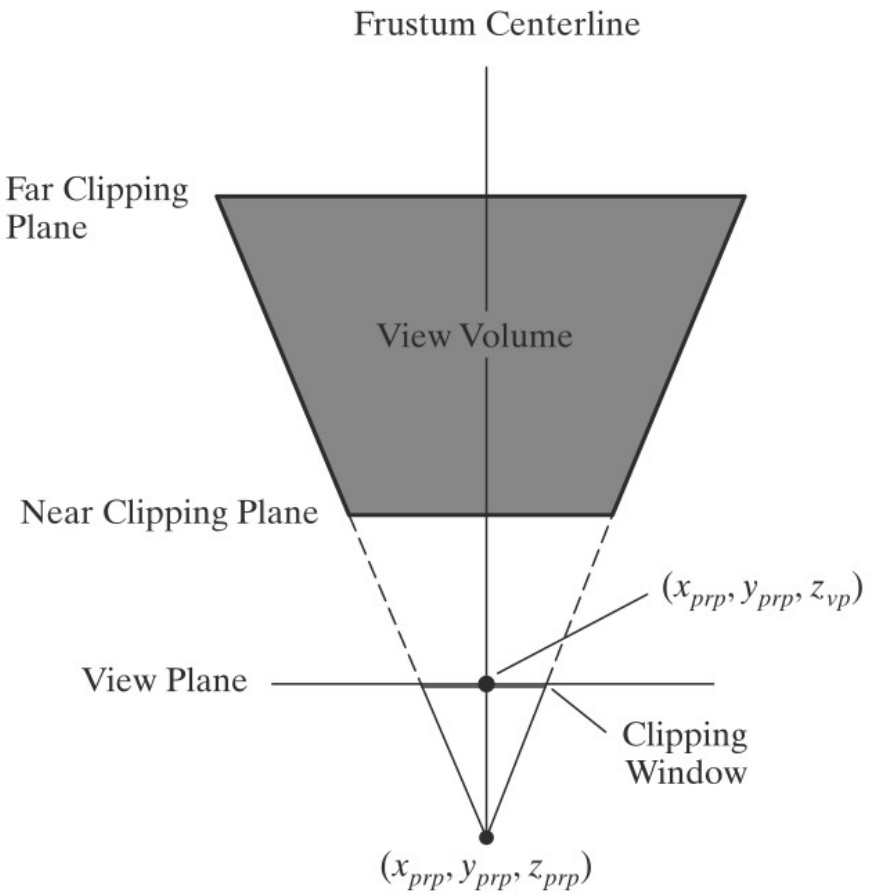
Volume de Projeção Perspectiva

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



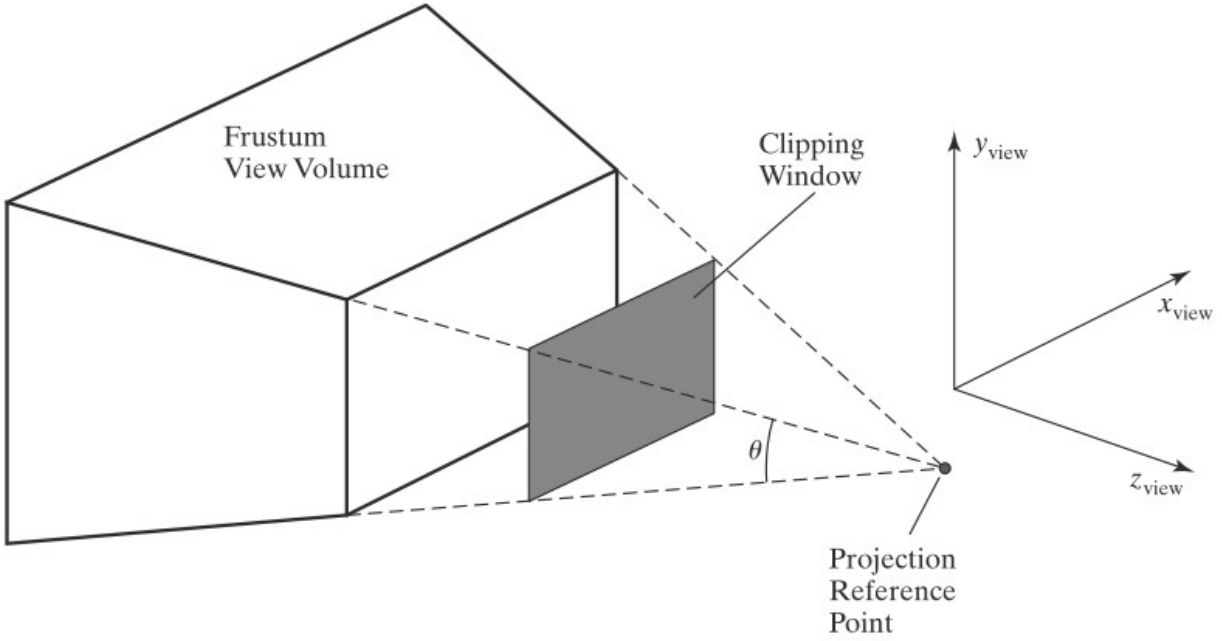
Frustum Simétrico de Projeção Perspectiva

- 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

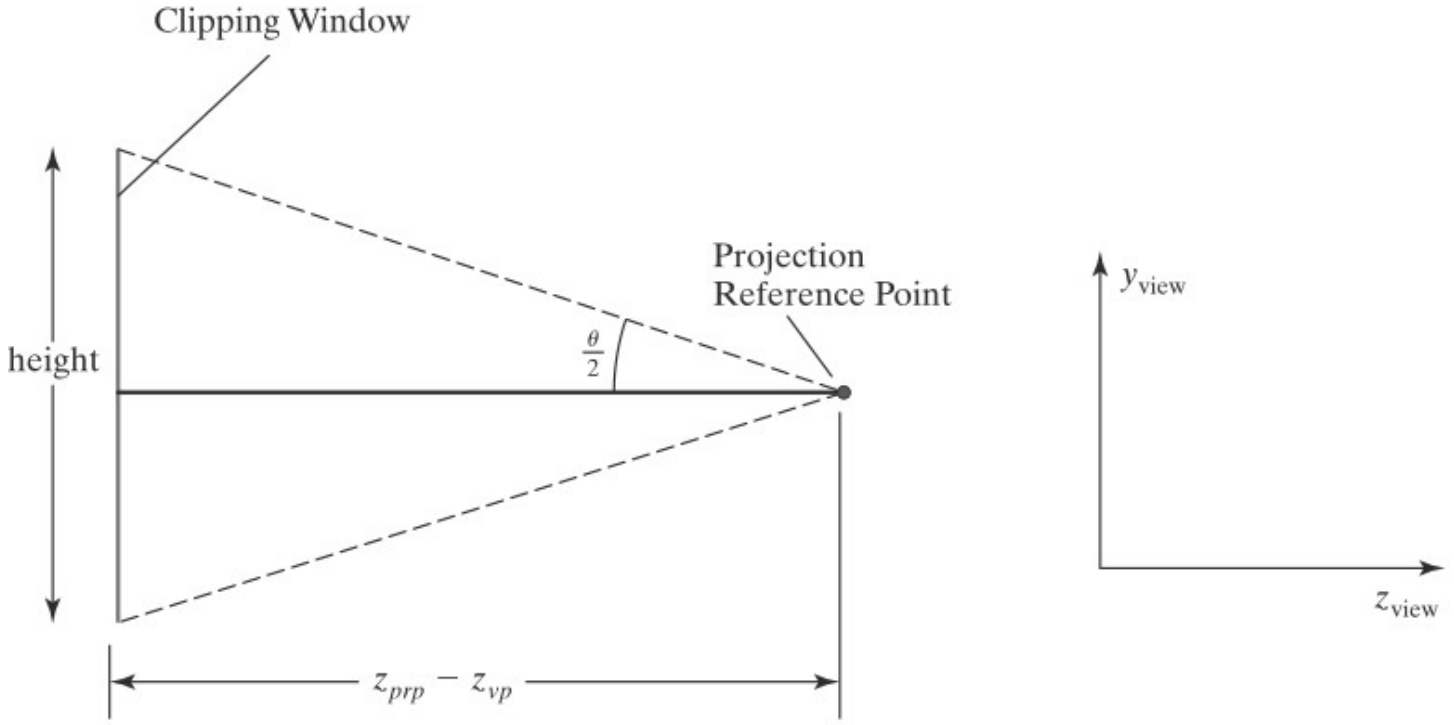
- Uma projeção perspectiva pode também ser aproximada considerando o **cone de visão**, definido pelo **ângulo do campo de visão**, de uma câmera:
  - Grandes ângulos de campo de visão correspondem a lentes grandes-angulares



  - 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

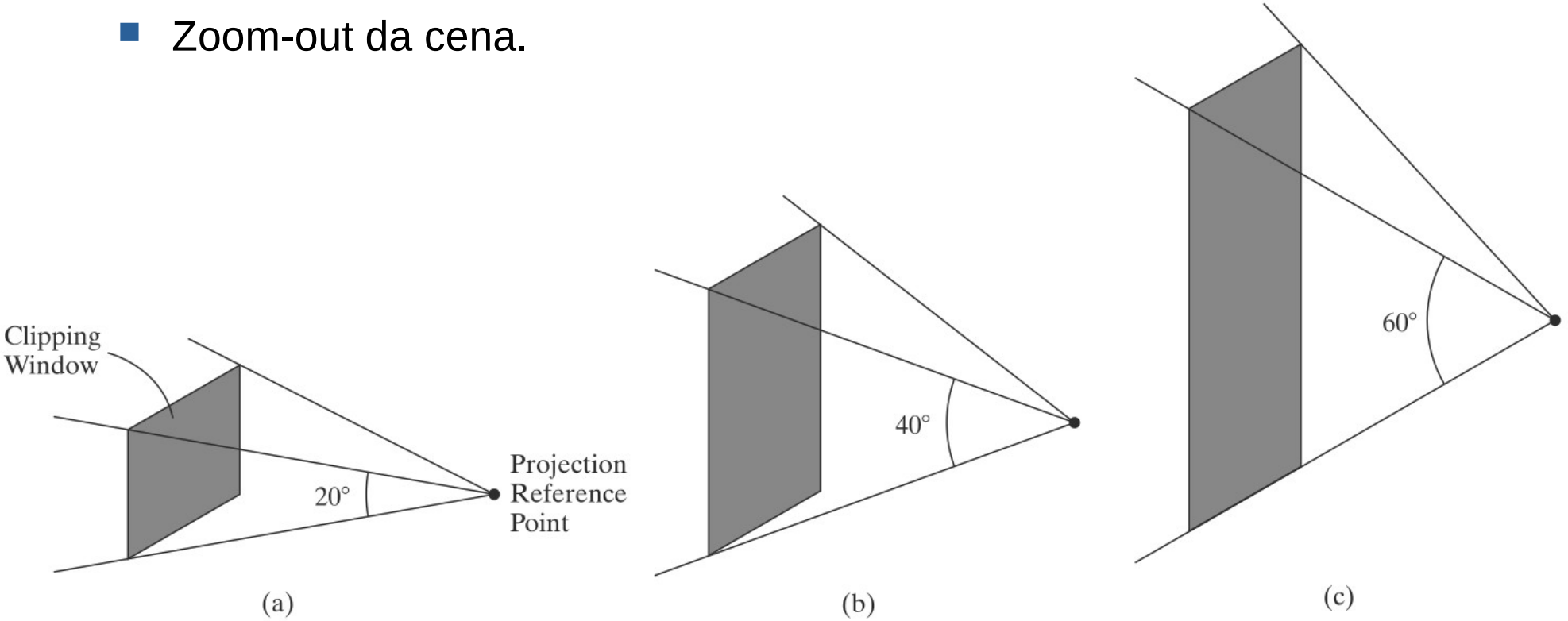
- Dado um ponto de referência e a posição do plano de visão, o **ângulo do campo de visão** determina a altura da janela de recorte



- 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)*

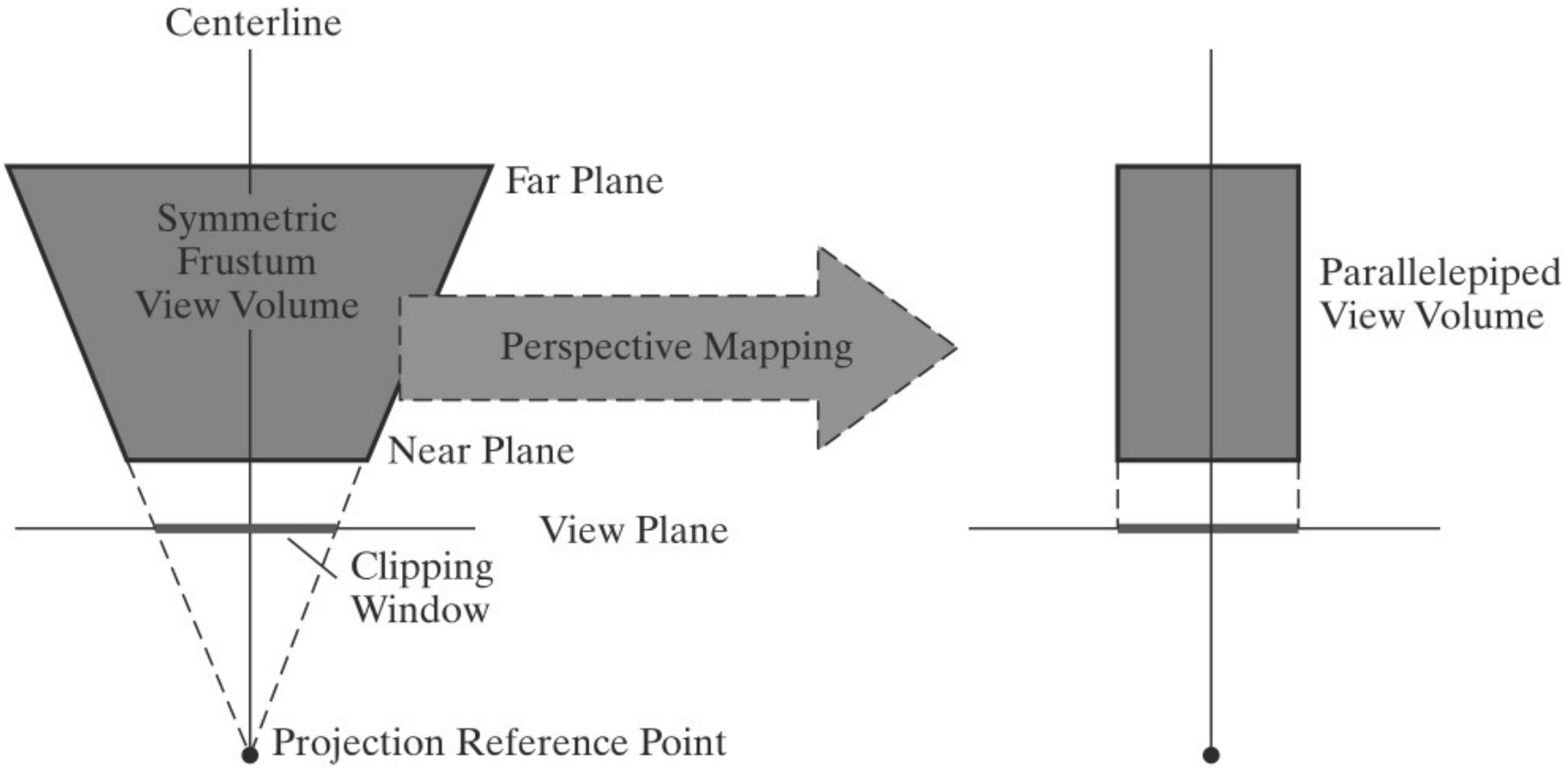
Frustum Simétrico de Projeção Perspectiva

- 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.



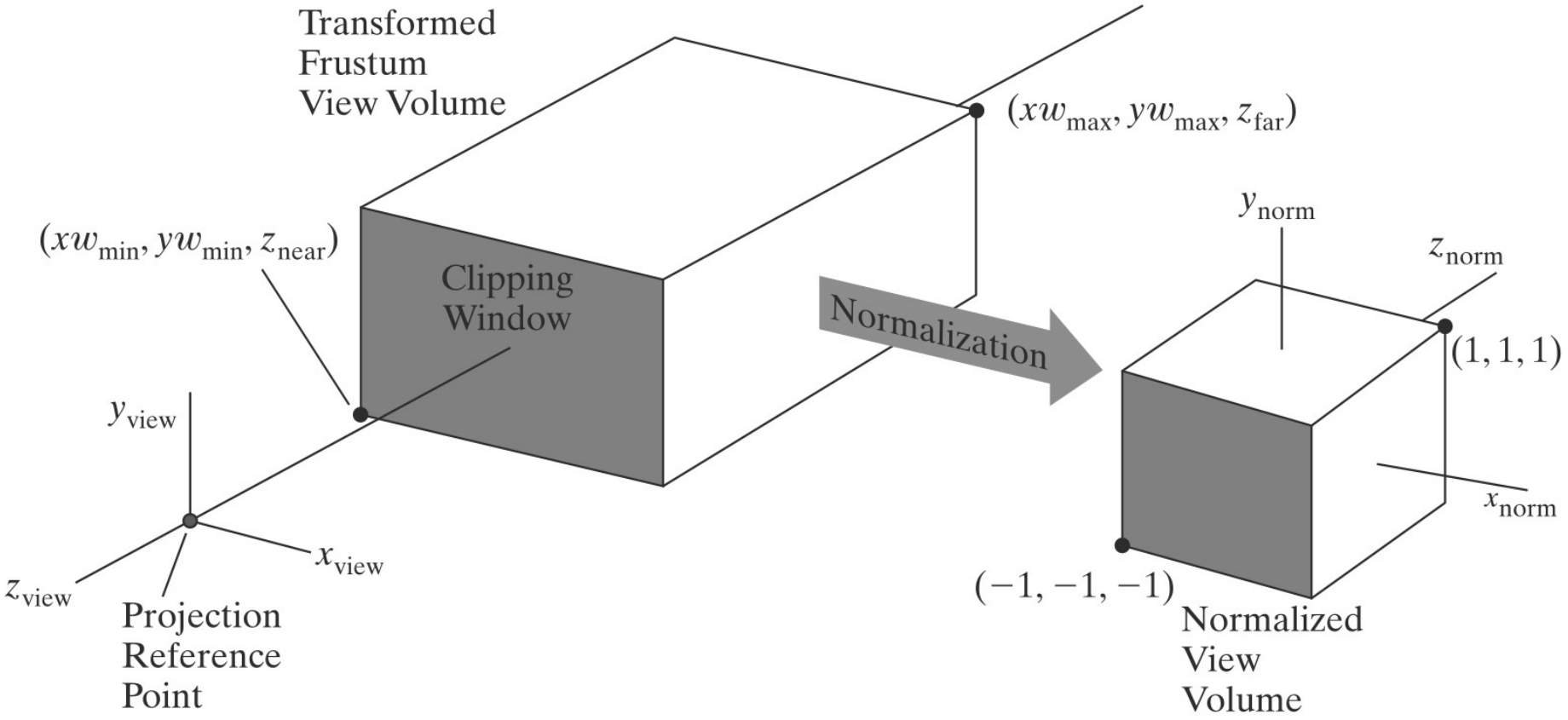
Frustum Simétrico de Projeção Perspectiva

- **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**



Sumário

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
- 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

Função de Projeção Ortogonal OpenGL

- A **biblioteca OpenGL** inclui funções para:
  - Transformação do Sistema de Coordenadas de Mundo para o de Visão;
  - Projeção ortogonal;
  - Projeção perspectiva simétrica (e oblíqua).
- A **biblioteca OpenGL Utility (GLU)** inclui funções para:
  - Especificar os parâmetros de visão;
  - Definir a transformação de projeção perspectiva simétrica.

Sumário

Função de Projeção Ortogonal OpenGL

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

```
glMatrixMode(GL_PROJECTION);
```

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

```
glOrtho(GLdouble xwmin, GLdouble xwmax, GLdouble ywmin,
    GLdouble ywmax, 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*

Funções OpenGL para Viewing 3D

- 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, o plano de recorte far estará na posição $z_{far} = -55$
  - Quaisquer valores podem ser atribuídos, desde que *dnear* < *dfar*.
- Por padrão, a OpenGL usa os seguintes valores para essa função:

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

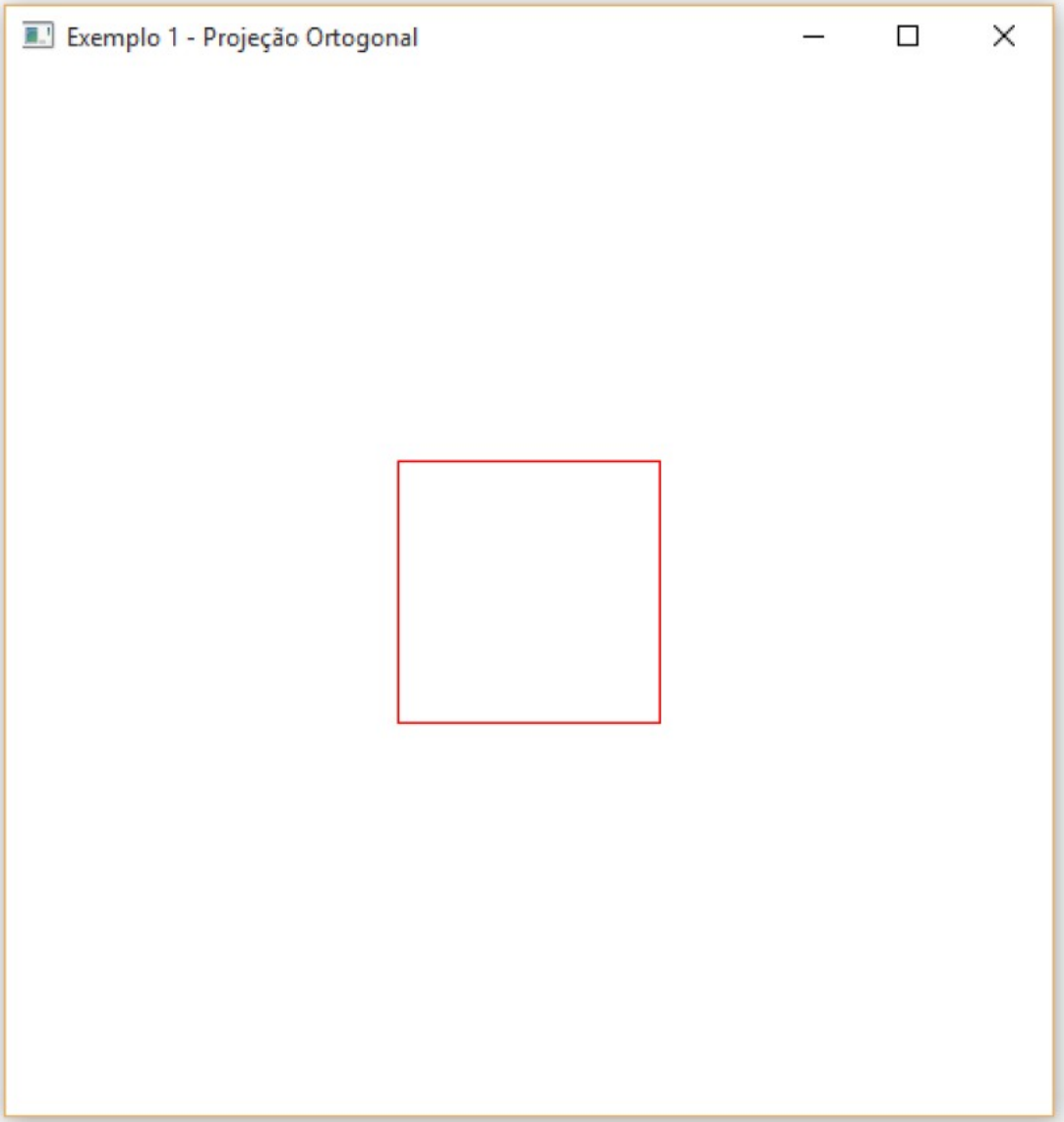
Função de Projeção Ortogonal OpenGL – Exemplo 1

```
#include <GL/freeglut.h>

int init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
    glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
    glMatrixMode(GL_PROJECTION); //define que a matrix é a de projeção
    glLoadIdentity(); //carrega a matrix de identidade
    glOrtho(-2.0, 2.0, -2.0, 2.0, 0.5, 2.0); //define uma projeção ortogonal
}

void display(){
    //limpa o buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //define que a matrix é a de modelo
    glMatrixMode(GL_MODELVIEW);

    //desenha um cubo
    glColor3f(1.0f, 0.0f, 0.0f);
    glutWireCube(1.0f);

    //força o desenho das primitivas
    glFlush();
}
// Continua...
```

Função de Projeção Ortogonal OpenGL – Exemplo 1

```
int main(int argc, char** argv){
    glutInit(&argc,argv);                          //inicializa o GLUT
    glutInitDisplayMode(GLUT_SINGLE| GLUT_RGB); //configura o modo de
                                                   //display
    glutInitWindowPosition(200,0);     //seta a posição inicial da janela
    glutInitWindowSize(500,500);       //configura a largura e altura da
                                       // janela de exibição
    //cria a janela de exibição
    glutCreateWindow("Exemplo 1 - Projeção Ortogonal");

    init();                            //executa função de inicialização
    glutDisplayFunc(display);
    glutMainLoop();                    //mostre tudo e espere
    return 0;
}
```

Função de Projeção Ortogonal OpenGL – Exemplo 1



Sumário

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:

```
glMatrixMode(GL_MODELVIEW);
```

Função de Transformação de Visão OpenGL

- Para especificar os **parâmetros de visão** usamos a função:

```
gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

- Essa função define:
  - A origem do sistema de visão $P_0 = (x_0, y_0, z_0)$ no sistema de coordenadas de mundo (a localização da câmera)
  - A posição de referência $P_{ref} = (x_{ref}, y_{ref}, z_{ref})$ (para onde a câmera aponta)
  - O vetor view-up $V = (V_x, V_y, V_z)$

Função de Transformação de Visão OpenGL

- A direção positiva do eixo z_{view} do sistema de coordenadas de visão está na direção $N = P_0 - P_{ref}$
  - 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 $M_{WC,VC}$
  - Matriz que alinha o sistema de visão com o sistema de coordenadas de mundo.
- Por padrão os parâmetros de visão da `gluLookAt(...)` são:
  - $P_0 = (0, 0, 0)$
  - $P_{ref} = (0, 0, -1)$
  - $V = (0, 1, 0)$

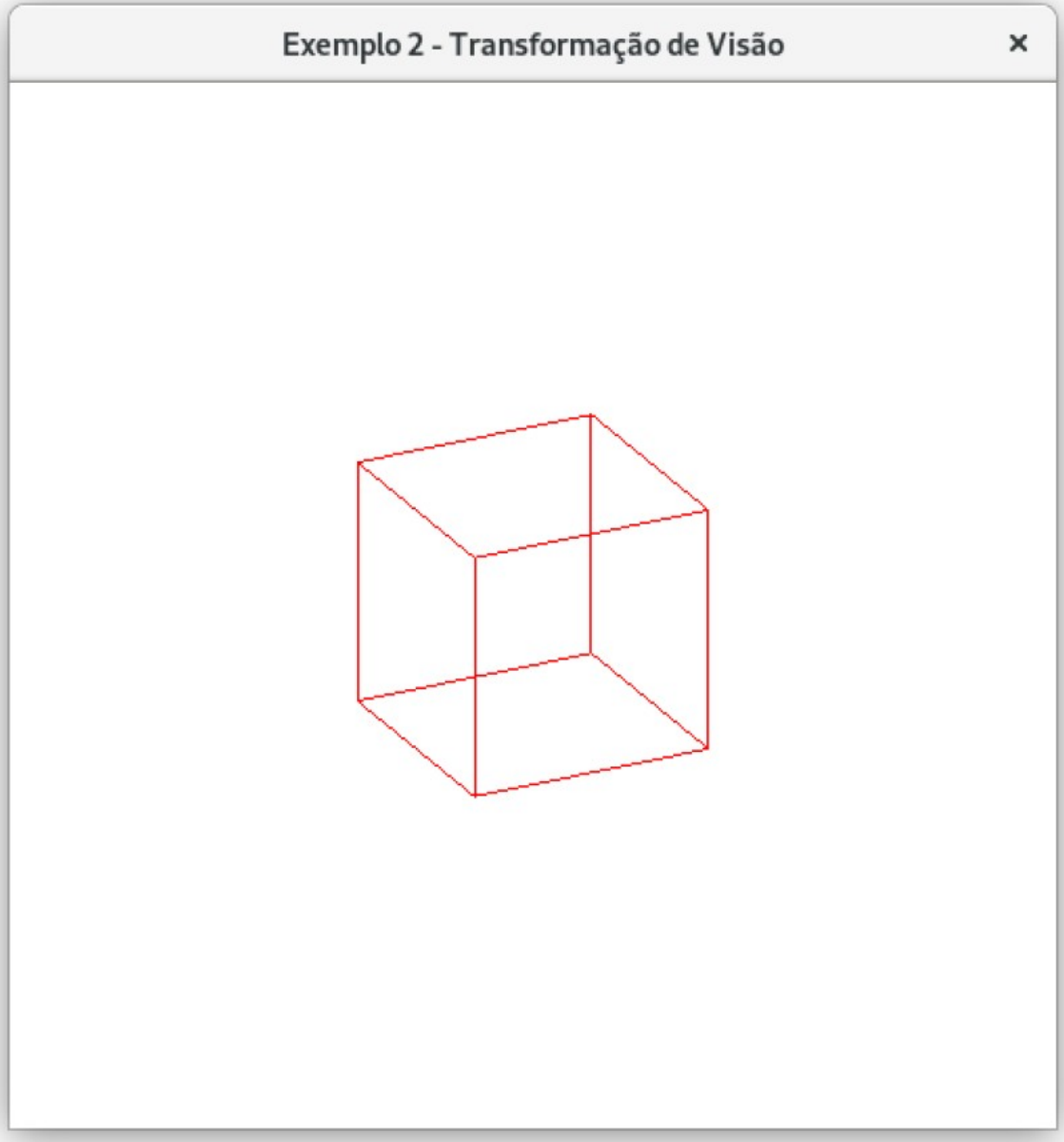
Função de Projeção Ortogonal OpenGL – Exemplo 2

```
#include <GL/freeglut.h>

int init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
    glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
    glMatrixMode(GL_MODELVIEW); //define que a matrix é a model view
    glLoadIdentity(); //carrega a matrix de identidade
    gluLookAt(1.0, 0.5, 0.5, //posição da câmera
              0.0, 0.0, 0.0, //para onde a câmera aponta
              0.0, 1.0, 0.0); //vetor view-up
    glMatrixMode(GL_PROJECTION); //define que a matrix é a de projeção
    glLoadIdentity(); //carrega a matrix de identidade
    //define uma projeção ortogonal
    glOrtho(-2.0, 2.0, -2.0, 2.0, 0.25, 2.5);
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  //limpa o buffer
    glMatrixMode(GL_MODELVIEW);     //define que a matrix é a de modelo
    glColor3f(1.0f, 0.0f, 0.0f);    //desenha um cubo
    glutWireCube(1.0f);
    glFlush();              //força o desenho das primitivas
}
// Continua...
```

Função de Projeção Ortogonal OpenGL – Exemplo 2

```
int main(int argc, char** argv){
    glutInit(&argc,argv);                          //inicializa o GLUT
    glutInitDisplayMode(GLUT_SINGLE| GLUT_RGB); //configura o modo de
                                                   //display
    glutInitWindowPosition(200,0);     //seta a posição inicial da janela
    glutInitWindowSize(500,500);       //configura a largura e altura da
                                       // janela de exibição
    //cria a janela de exibição
    glutCreateWindow("Exemplo 2 – Transformação de Visão");

    init();                            //executa função de inicialização
    glutDisplayFunc(display);
    glutMainLoop();                    //mostre tudo e espere
    return 0;
}
```

Função de Projeção Ortogonal OpenGL – Exemplo 2



Sumário

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:

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

- Os parâmetros ***theta*** e ***aspect*** definem o tamanho e posição da janela de recorte:
  - $0^o \leq$ `theta` $\leq 180^o$ define o ângulo do campo de visão
  - aspect é o valor da razão de aspecto *width/height* 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*

Função para Projeção Perspectiva Simétrica OpenGL

- O ponto de visão (posição da câmera) 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 *dnear* > 0 e *dfar* > 0

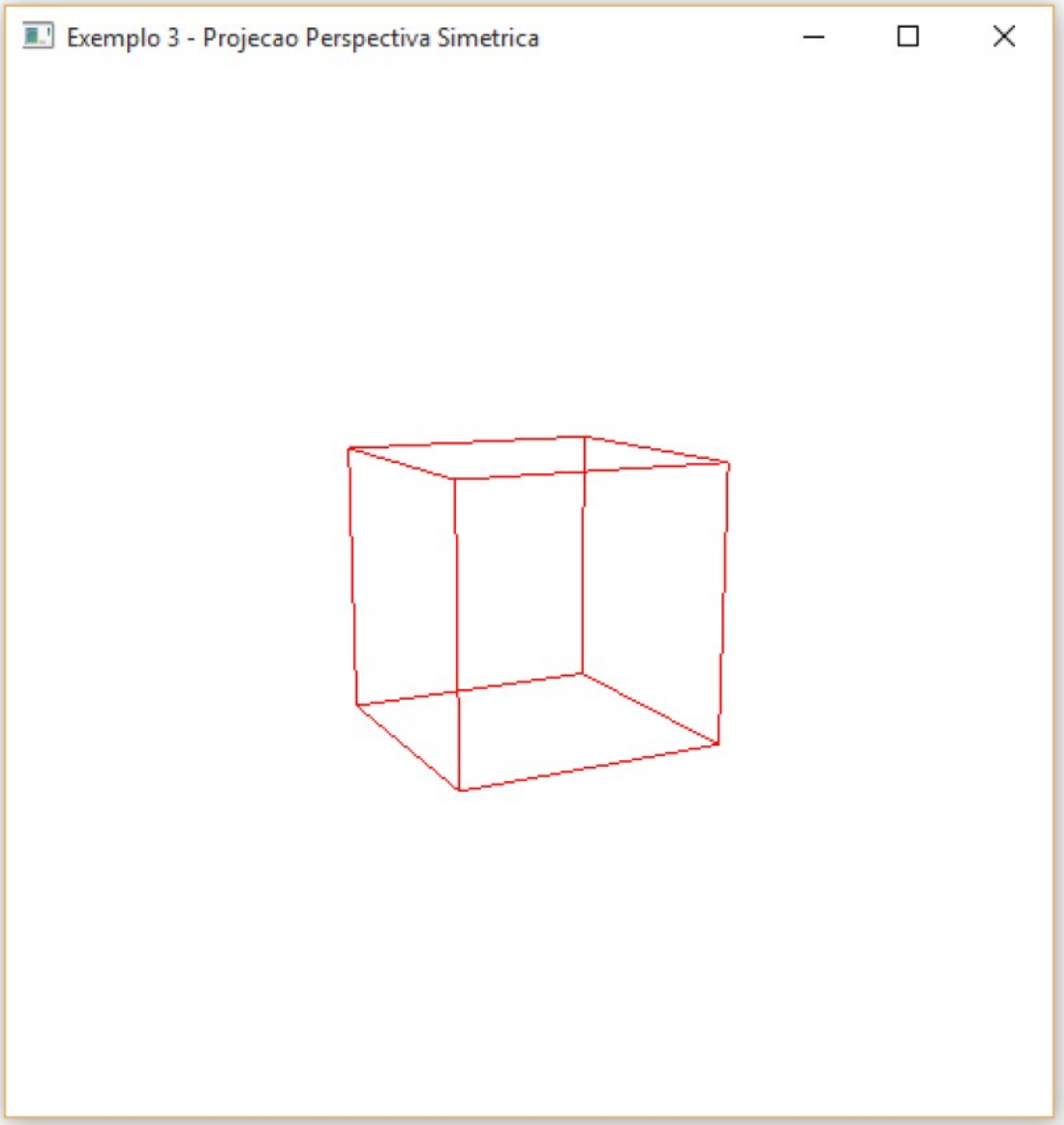
Função de Projeção Ortogonal OpenGL – Exemplo 3

```
#include <GL/freeglut.h>

int init(){
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //define a cor de fundo
    glEnable(GL_DEPTH_TEST); //habilita o teste de profundidade
    glMatrixMode(GL_MODELVIEW); //define que a matrix é a model view
    glLoadIdentity(); //carrega a matrix de identidade
    gluLookAt(4.0, 1.0, 2.0, //posição da câmera
              0.0, 0.0, 0.0, //para onde a câmera aponta
              0.0, 1.0, 0.0); //vetor view-up
    glMatrixMode(GL_PROJECTION); //define que a matrix é a de projeção
    glLoadIdentity(); //carrega a matrix de identidade
    gluPerspective(45.0, 1.0, 0.1, 10.0); //define uma projeção perspectiva
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  //limpa o buffer
    glMatrixMode(GL_MODELVIEW);     //define que a matrix é a de modelo
    glColor3f(1.0f, 0.0f, 0.0f);    //desenha um cubo
    glutWireCube(1.0f);
    glFlush();              //força o desenho das primitivas
}
// Continua...
```

Função de Projeção Ortogonal OpenGL – Exemplo 3

```
int main(int argc, char** argv){
    glutInit(&argc,argv);                          //inicializa o GLUT
    glutInitDisplayMode(GLUT_SINGLE| GLUT_RGB); //configura o modo de
                                                   //display
    glutInitWindowPosition(200,0);     //seta a posição inicial da janela
    glutInitWindowSize(500,500);       //configura a largura e altura da
                                       // janela de exibição
    //cria a janela de exibição
    glutCreateWindow("Exemplo 3 – Projeção Perspectiva Simétrica");

    init();                            //executa função de inicialização
    glutDisplayFunc(display);
    glutMainLoop();                    //mostre tudo e espere
    return 0;
}
```

Função de Projeção Ortogonal OpenGL – Exemplo 3



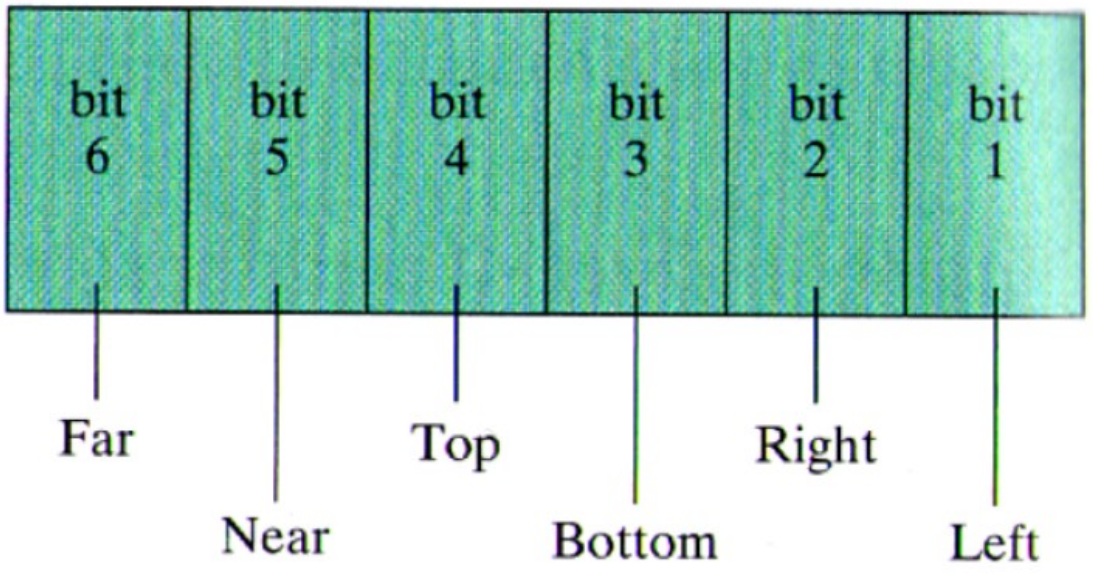
Sumário

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

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.

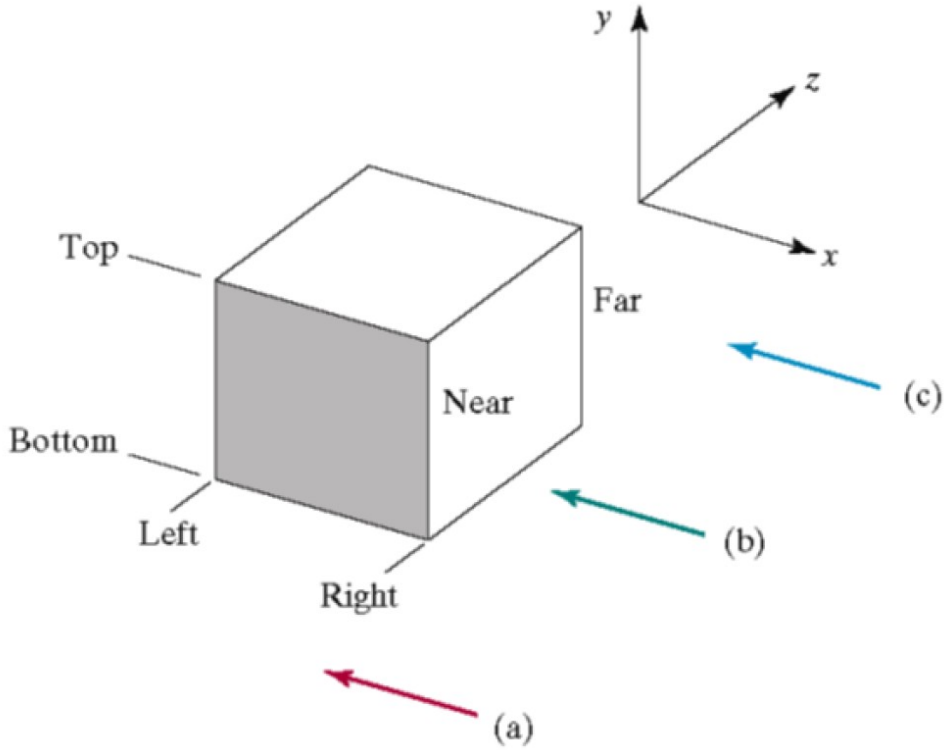
Código de Região 3D

- Sejam os planos que limitam o volume de visão, representados pelas coordenadas:
  - xw_{min} (esquerda)
  - xw_{max} (direita)
  - yw_{min} (embaixo)
  - yw_{max} (acima)
  - zw_{min} (frente)
  - zw_{max} (trás)

Código de Região 3D

- Podemos definir os valores dos bits como:
  - $bit_1 = 1$ se $x < xw_{min}$
  - $bit_2 = 1$ se $x > xw_{max}$
  - $bit_3 = 1$ se $y < yw_{min}$
  - $bit_4 = 1$ se $y > yw_{max}$
  - $bit_5 = 1$ se $z < zw_{min}$
  - $bit_6 = 1$ se $z > zw_{max}$
- Qualquer sequencia de bits **diferente de 000000**, indica um ponto que esta fora da região de visão

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

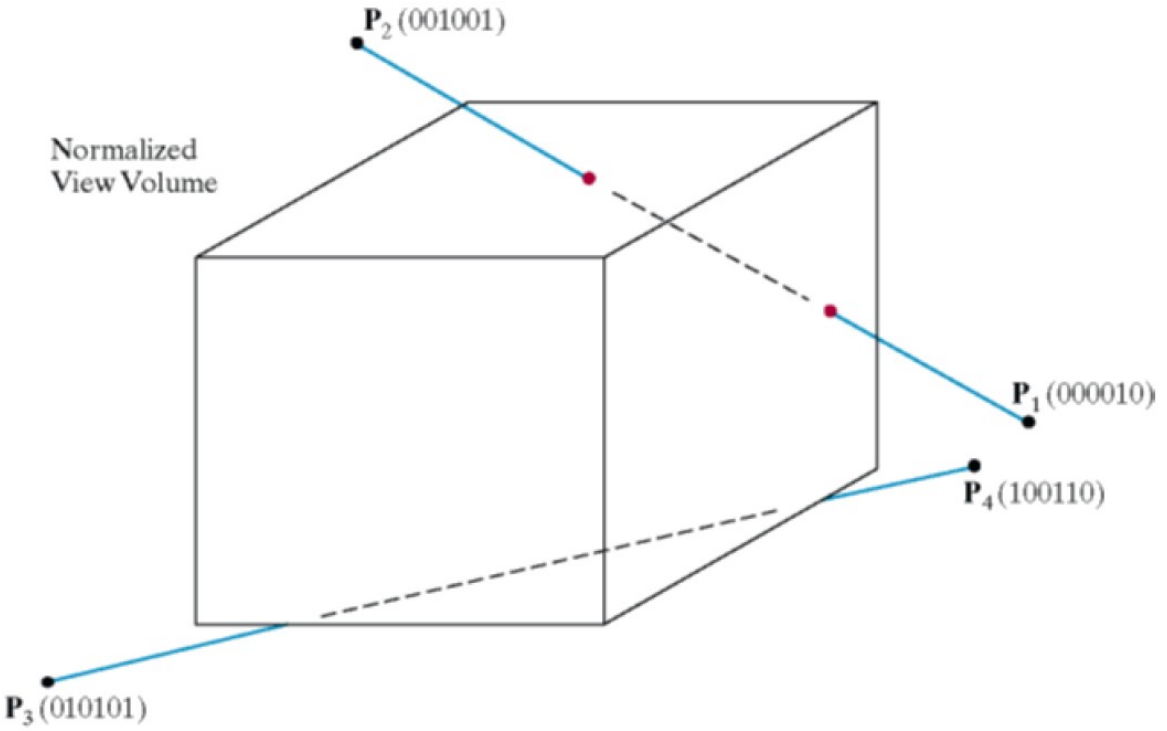
Region Codes
Between Near and Far Planes
(b)

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:
  - Linhas **completamente dentro** tem seus pontos definidos como 000000
  - Linhas que tenham 1 nas mesmas posições dos pontos finais estão **completamente fora** da janela de recorte
  - Se uma linha falha nesses testes, suas equações são avaliadas para calcular as **intersecções**

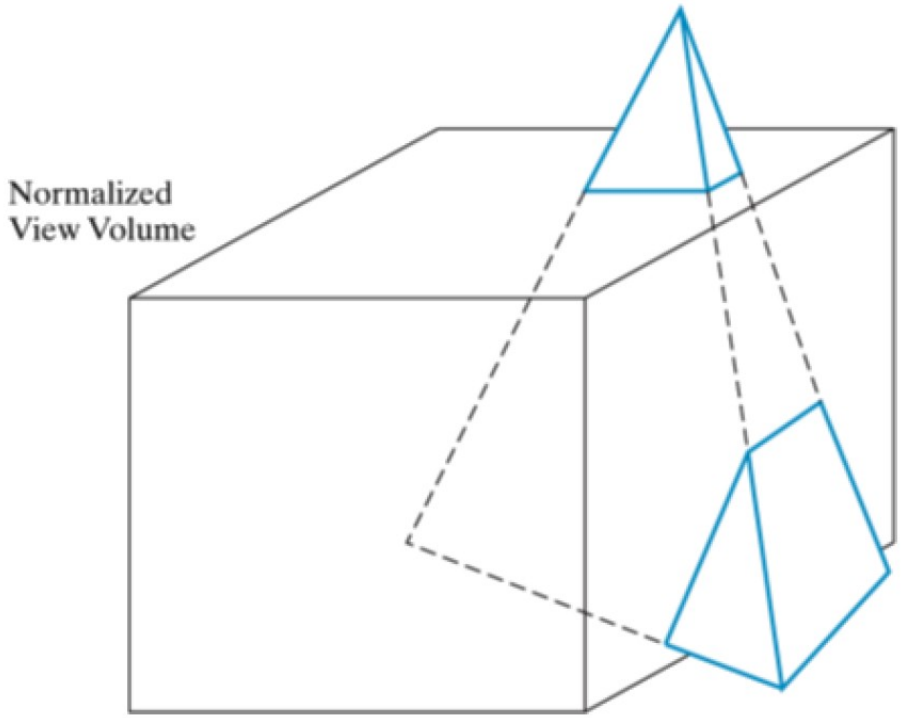


Recorte de Ponto e Linha 3D

- 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.

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