Transformações de *Viewing* 3D

SCC0250 - Computação Gráfica

Profa. Maria Cristina F. Oliveira

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

9 de outubro de 2023

Sumário

Sumário

Retomando

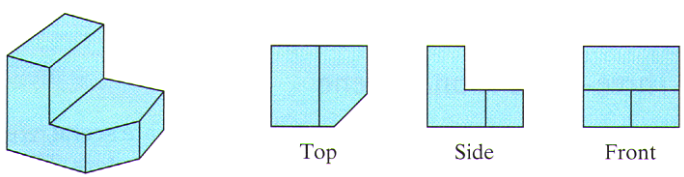
Viewing Pipeline 3D

As funções de *viewing* processam a descrição geométrica da cena por meio de um conjunto de transformações a fim de gerar uma visão específica da cena a ser exibida na superfície do dispositivo de saída

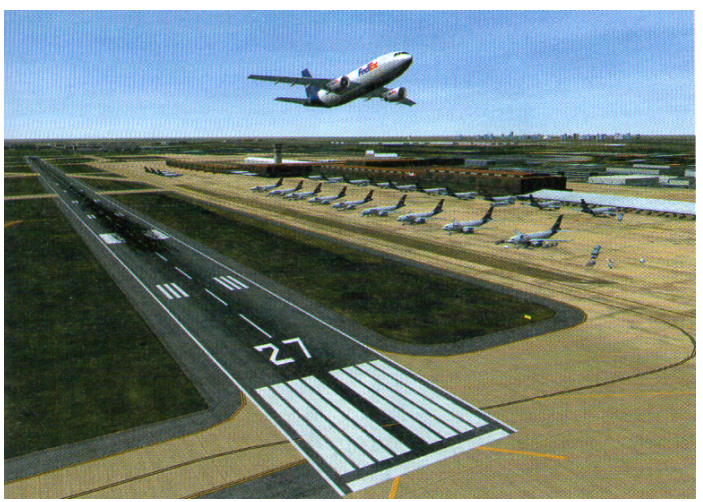
- *View matrix* + *modelview matrix*
- Transformação de projeção
- Identificação de partes visíveis da cena
- Cálculo dos efeitos de iluminação

Transformações de Projeção

- Há duas maneiras de projetar uma cena:
  - **Projeção Paralela:** os pontos dos objetos são projetados ao longo de linhas paralelas – muito usado em desenhos arquitetônicos e de engenharia (preserva comprimentos e ângulos)
  - **Projeção Perspectiva:** os pontos dos objetos são projetados ao longo de linhas convergentes – cenas mais realísticas (objetos distantes da posição de observação parecem menores)



(a) Projeção Paralela

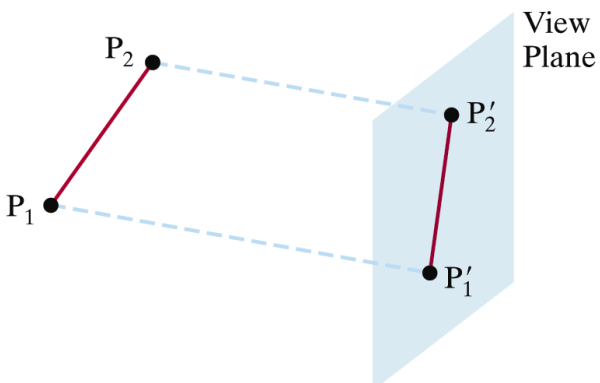


(b) Projeção Perspectiva

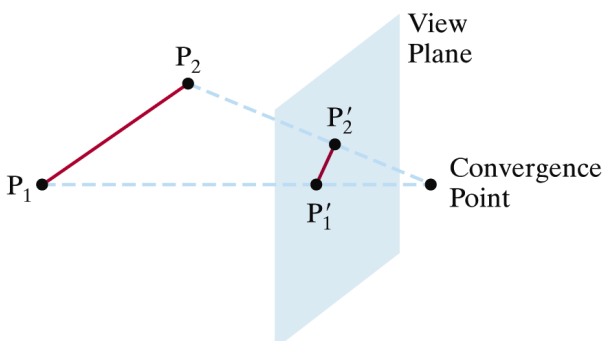
Transformações de Projeção

- Após a transformação para o VCS, a próxima etapa do *Viewing Pipeline 3D* é a projeção da cena

- Em geral, os pacotes gráficos admitem
  - **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 para um ponto



(c) Projeção Paralela



(d) Projeção Perspectiva

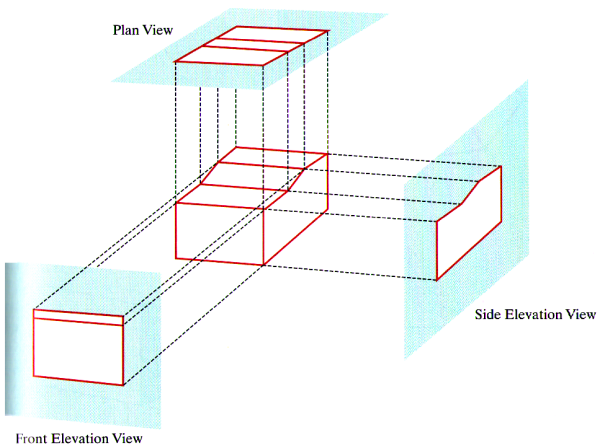
Sumário

Projeções Ortogonais

Projeção Ortogonal (ou Ortográfica)

- As coordenadas dos objetos são projetadas sobre um plano de projeção, ao longo de linhas paralelas ao vetor normal $\mathbf{N}$ (direção de observação)

- As linhas de projeção são perpendiculares ao plano de projeção
- Utilizada para produzir as visões frontal, lateral e superior de um objeto
- Preserva comprimentos e ângulos: útil para desenhos arquitetônicos e de engenharia



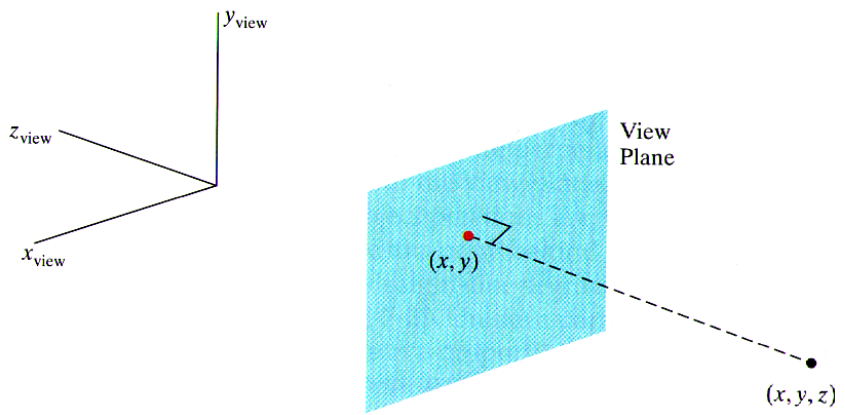
Coordenadas de Projeções Ortogonais

- Assumindo uma direção de projeção paralela ao eixo z_{view}, as equações da transformação de projeção ortogonal para um ponto de coordenadas (x, y, z) são dadas por:

$$x_p = x$$

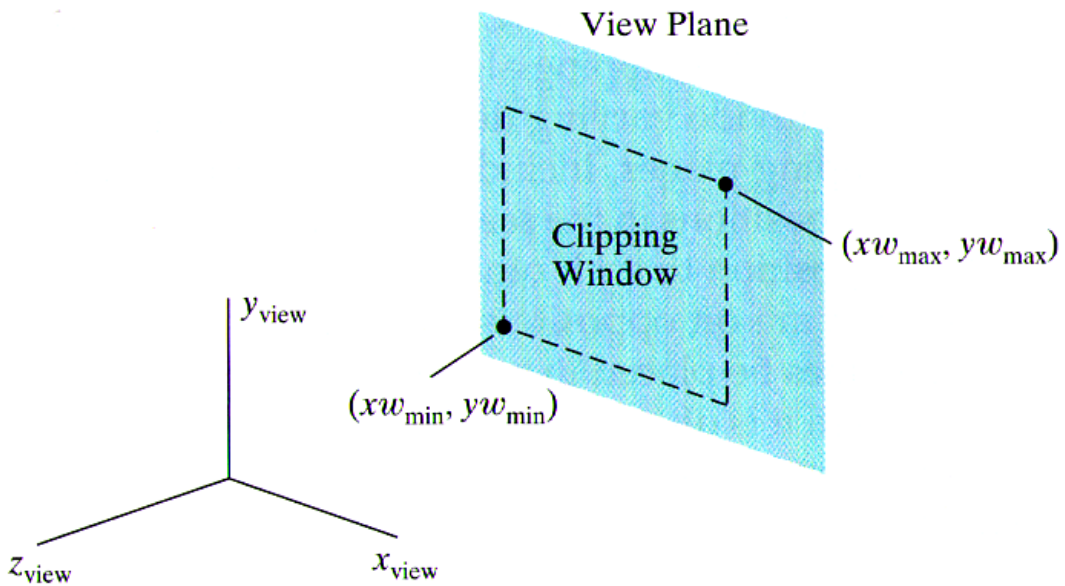
$$y_p = y$$

- O valor de z é preservado para uso futuro pelos procedimentos para determinar visibilidade



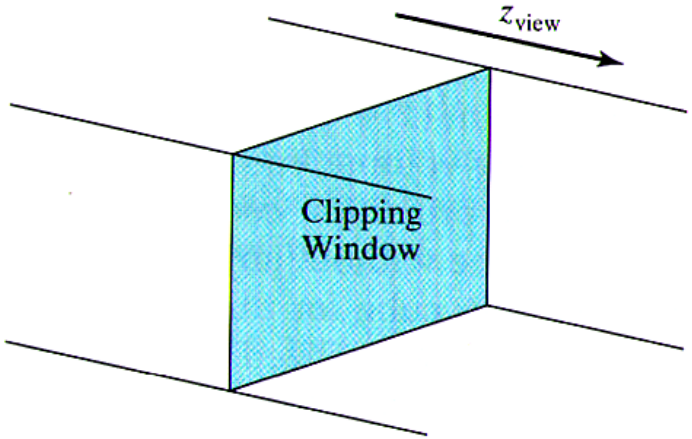
Janela de Recorte e Volume de Projeção Ortogonal

- Uma **Janela de Recorte** determina qual região da cena 3D será efetivamente transferida para o plano de projeção
  - É necessário determinar os limites dessa janela no plano de projeção, com as arestas paralelas aos eixos x_{view} e y_{view}

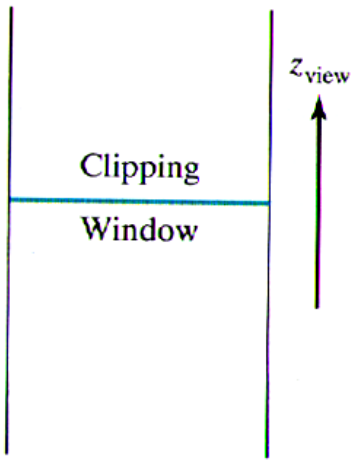


Janela de Recorte e Volume de Projeção Ortogonal

- As arestas da *Janela de Recorte* definem os limites, nas direções x_{view} e y_{view}, da região da cena a ser exibida, formando um **volume de observação (*viewing volume*)** característico da projeção ortogonal



Side View
(a)

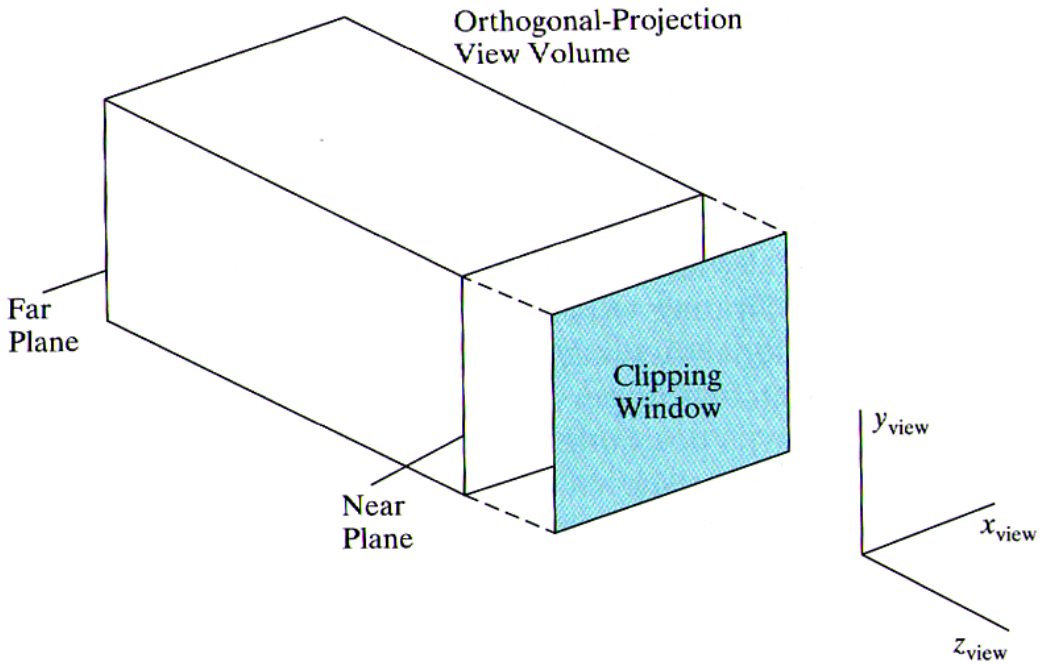


Top View
(b)

Janela de Recorte e Volume de Projeção Ortogonal

- A extensão desse volume na direção z_{view} é delimitada por por dois planos paralelos ao plano de projeção, denominados **planos de recorte *near* / *far***
- Assim, define-se um volume finito da cena, sendo que objetos que estão na frente ou atrás dessa região são eliminados da renderização da cena
- Supondo a direção de observação ao longo do eixo z_{view} negativo, temos $z_{far} < z_{near}$

Janela de Recorte e Volume de Projeção Ortogonal



Transformação de Normalização para Projeção Ortogonal

- Como em uma projeção ortogonal qualquer coordenada (x, y, z) é mapeada para (x, y), as coordenadas dos pontos no interior do volume de observação coincidem com as coordenadas de projeção. Assim, podem ser mapeadas diretamente para um **volume de visão normalizado**, sem que precisem ser reprojetadas.

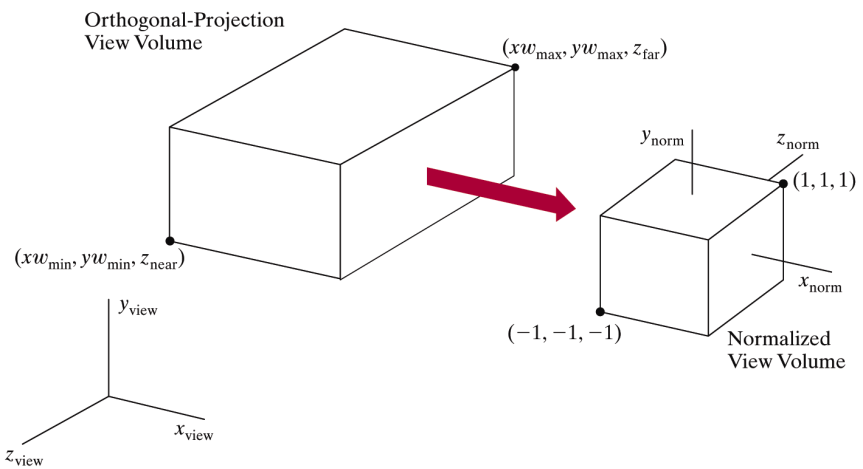


Figura: Transformação de normalização em um sistema de referência dado pela mão esquerda (normalmente o sistema da tela)

Transformação de Normalização para Projeção Ortogonal

- Essa transformação de normalização é semelhante à obtida no caso 2D, incluindo os valores da coordenada z, em que o intervalo z_{near} a z_{far} é normalizado para o intervalo -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 a descrição da cena do WCS para o VCS produz a matriz completa da transformação que obtém as coordenadas da projeção ortogonal no volume normalizado

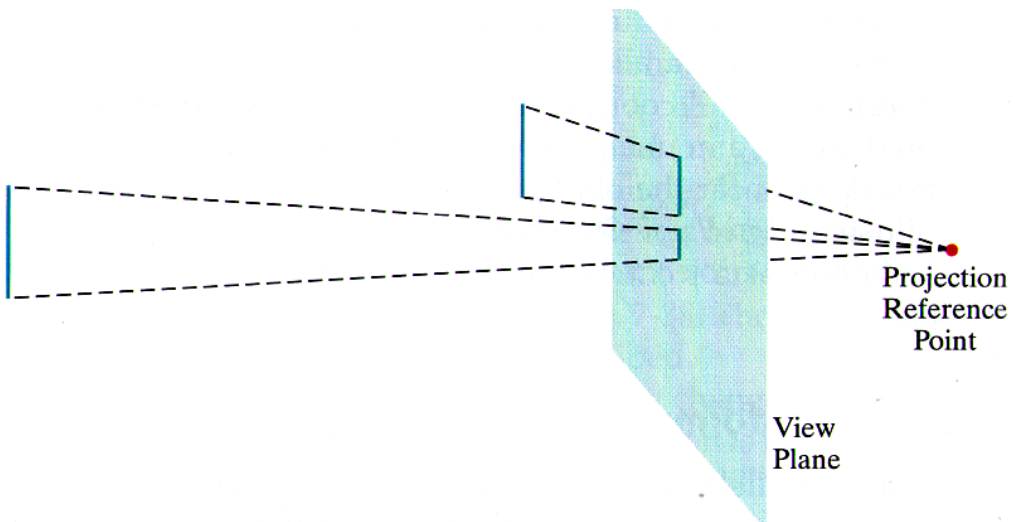
$$\mathbf{M}_{ortho,norm} \cdot \mathbf{M}_{WC,VC}$$

- Operações como recorte e identificação de superfícies visíveis, entre outras, podem ser executadas de maneira mais eficiente em relação a esse volume normalizado

Sumário

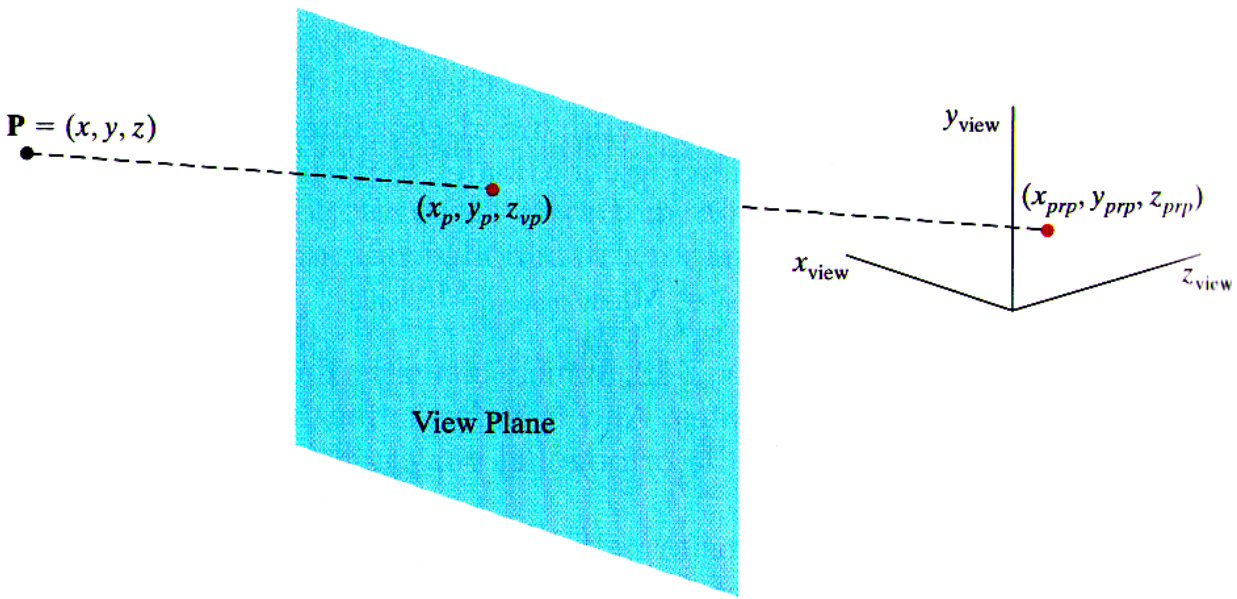
Projeções Perspectivas

- Para renderizar cenas mais realistas do que as obtidas utilizando projeções paralelas deve-se considerar que os raios de luz refletidos na cena seguem caminhos convergentes
- Isso pode ser aproximado projetando os objetos no plano de observação ao longo de linhas convergentes a uma posição denominada **ponto de referência de projeção** (ou **centro de projeção**)



Transformação de Coordenadas de Projeção Perspectiva

- Algumas bibliotecas gráficas permitem escolher o ponto de referência de projeção $(x_{prp}, y_{prp}, z_{prp})$



Transformação de Coordenadas de Projeção Perspectiva

- Considerando que o ponto (x, y, z) é projetado na posição (x_p, y_p, z_{vp}) no plano de projeção, podemos descrever qualquer ponto ao longo da 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 projeção $z' = z_{vp}$, então podemos determinar u nesse ponto:

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

Transformação de Coordenadas de Projeção Perspectiva

- Substituindo esse valor de u nas equações de x' e y' obtemos as equações da 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 definir uma matriz de transformação perspectiva a partir das equações derivadas anteriormente, uma vez que os denominadores dos coeficientes de x e y estão expressos em 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 trabalhando no espaço de coordenadas homogêneas

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

- Em que o parâmetro homogêneo é dado por:

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

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

- É imediato definir os elementos da matriz para obter as coordenadas homogêneas x_h e y_h como nas equações anteriores, mas a matriz deve preservar a informação de profundidade (valor z), do contrário o parâmetro homogêneo h vai distorcer as coordenadas z

- Isso é possível definindo os valores para a transformação de z de modo a normalizar as coordenadas z_p da projeção
  - Isso pode ser feito de várias formas, por exemplo:

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

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

Matriz de Transformação de Projeção Perspectiva

- Essa matriz obtém as coordenadas homogêneas do ponto projetado, portanto, o resultado deve ser dividido pelo fator homogêneo h obter as coordenadas no espaço cartesiano

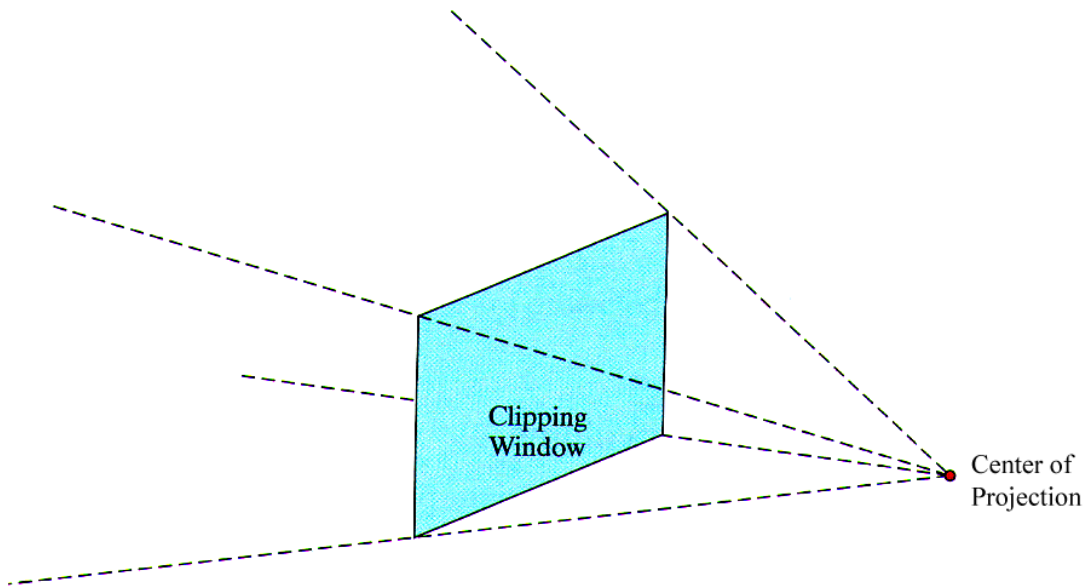
- Em outras palavras, seja $\mathbf{P} = (x, y, z, 1)$ o ponto a ser projetado
- As coordenadas homogêneas do ponto projetado são dadas por:

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

- Essas coordenadas $\mathbf{P}_h = (x_h, y_h, z_h, h)$ devem ser divididas por h para obter as coordenadas cartesianas correspondentes

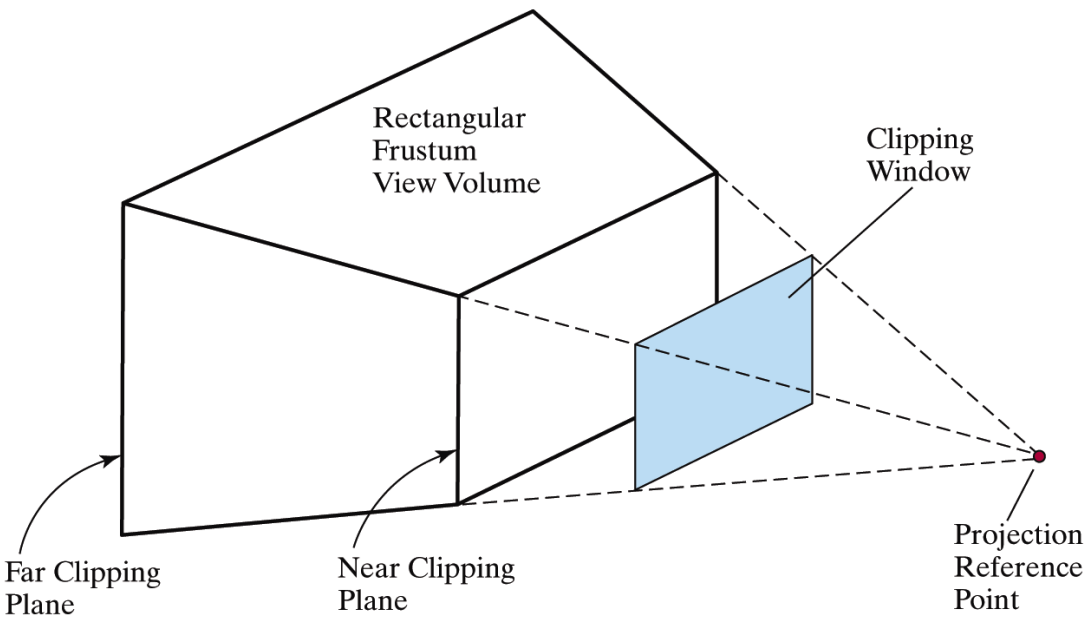
Volume de Projeção Perspectiva

- Uma projeção perspectiva define um volume de visão dado por uma pirâmide infinita cujo ápice está no centro de projeção, normalmente chamada de **pirâmide de visão**
  - Objetos fora dessa pirâmide são eliminados pelas rotinas de recorte



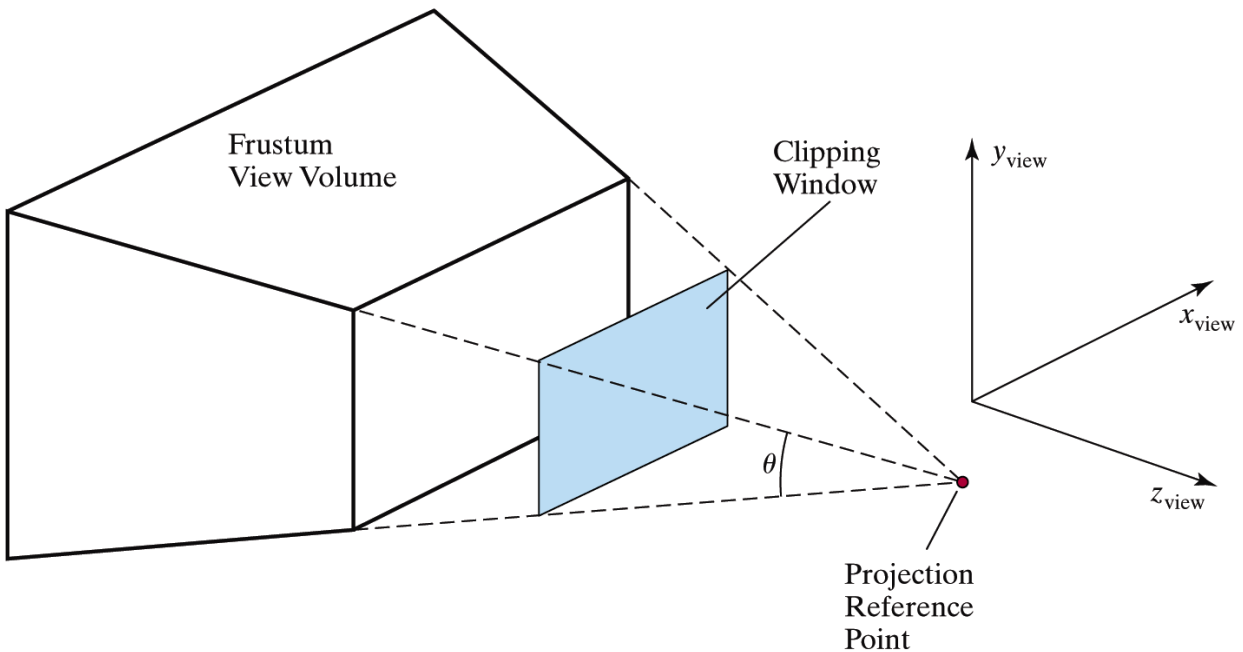
Volume de Projeção Perspectiva

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



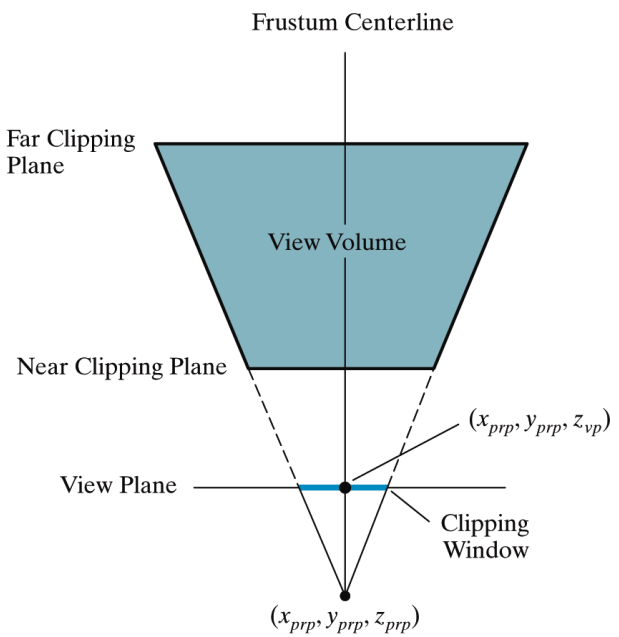
Frustum Simétrico de Projeção Perspectiva

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



Frustum Simétrico de Projeção Perspectiva

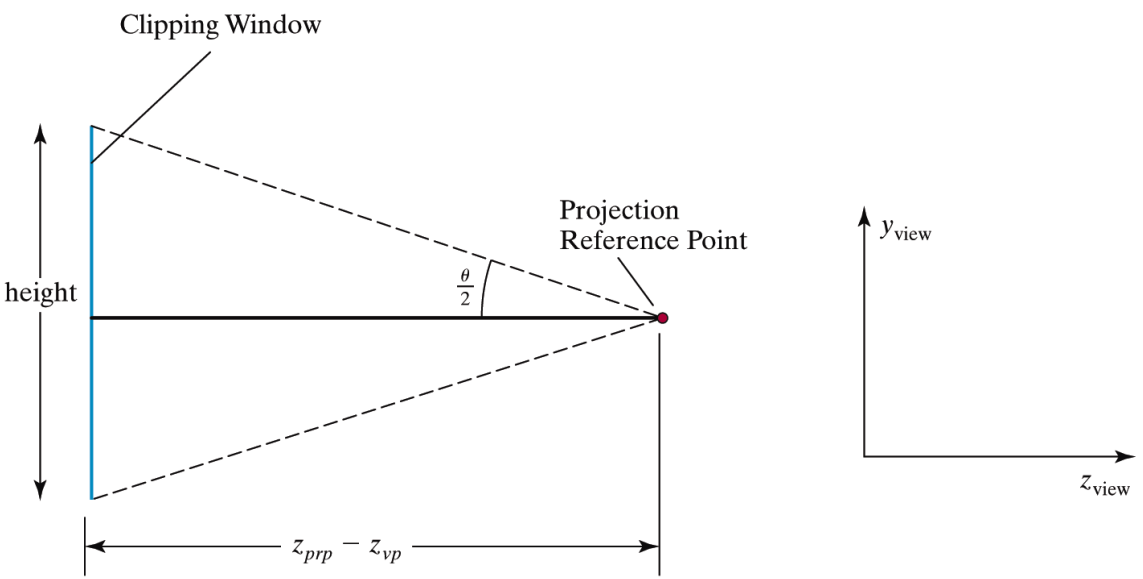
- Quando trabalhamos com **projeções perspectivas simétricas**, a linha que liga o centro de projeção ao ponto no centro do plano de projeção, perpendicular a esse, é a linha central do ***frustum simétrico***



Frustum Simétrico de Projeção Perspectiva

- Assim, dado um ponto de referência $(x_{prp}, y_{prp}, z_{prp})$ e a posição z_{vp} aonde foi posicionado o plano de projeção, o ângulo θ que definie o campo de visão da câmera 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

- A largura da janela pode ser obtida considerando um parâmetro adicional, que é a razão de aspecto $aspect = (width/height)$

- Assim podemos substituir o valor $(z_{prp} - z_{vp})$ na diagonal da matriz $\mathbf{M}_{pers}$ por

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

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

- Ou por

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

Frustum Simétrico de Projeção Perspectiva

- Assim temos

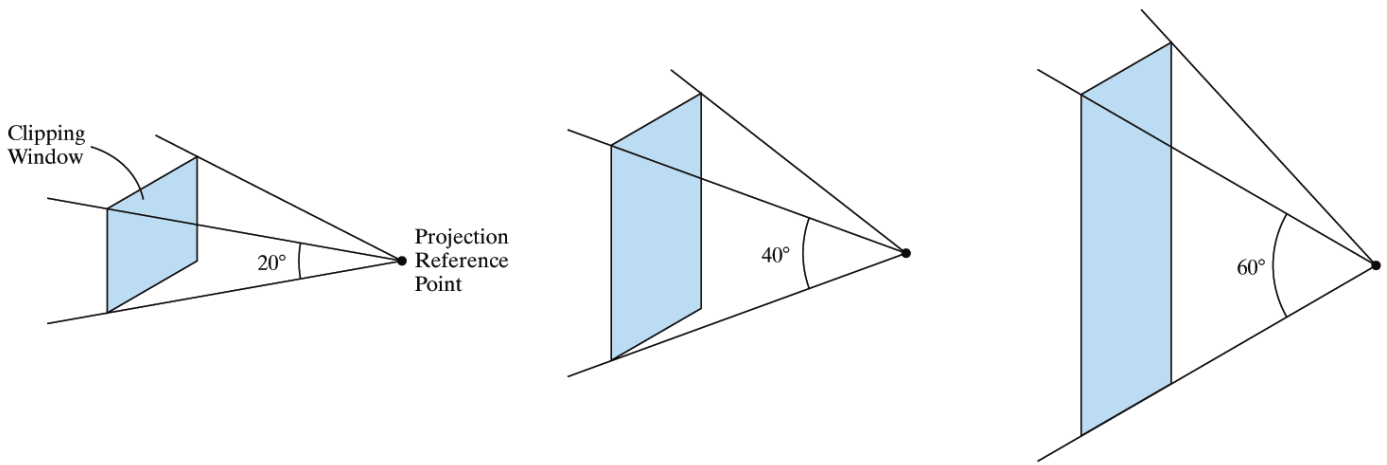
$$\mathbf{M}_{pers} = \begin{bmatrix} \frac{height}{2}\cot\left(\frac{\theta}{2}\right) & 0 & -x_{prp} & x_{prp}z_{vp} \\ 0 & \frac{height}{2}\cot\left(\frac{\theta}{2}\right) & -y_{prp} & y_{prp}z_{vp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

- Ou

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

Frustum Simétrico de Projeção Perspectiva

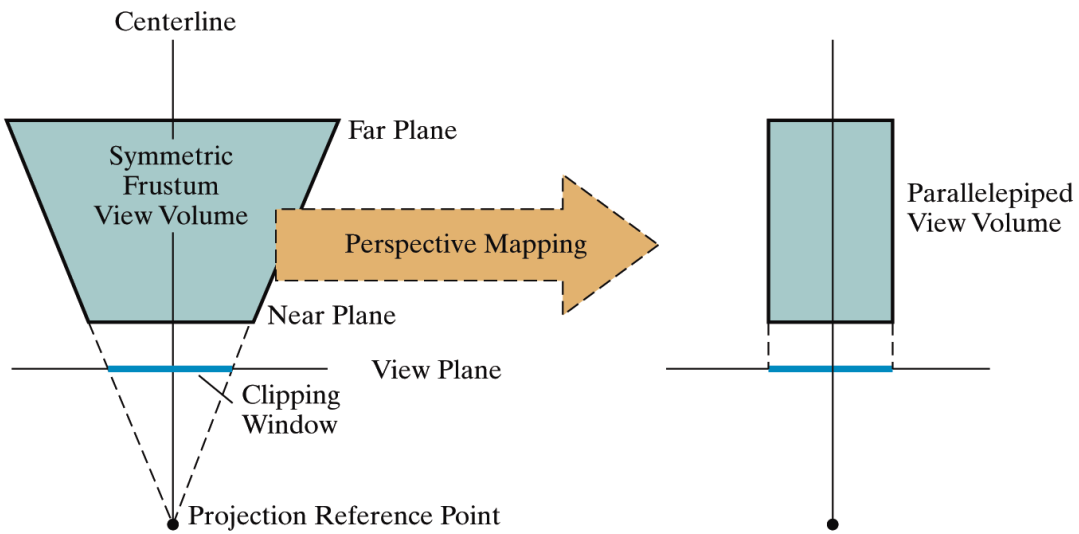
- Diminuir o ângulo do campo de visão diminui a janela de recorte
  - Equivale a afastar o centro de projeção do plano de projeção
  - *Zoom-in* em uma pequena região da cena
- Aumentar o ângulo do campo de visão aumenta a janela de recorte
  - Equivale a aproximar o centro de projeção do plano de observação
  - *Zoom-out* da cena



Frustum Simétrico de Projeção Perspectiva

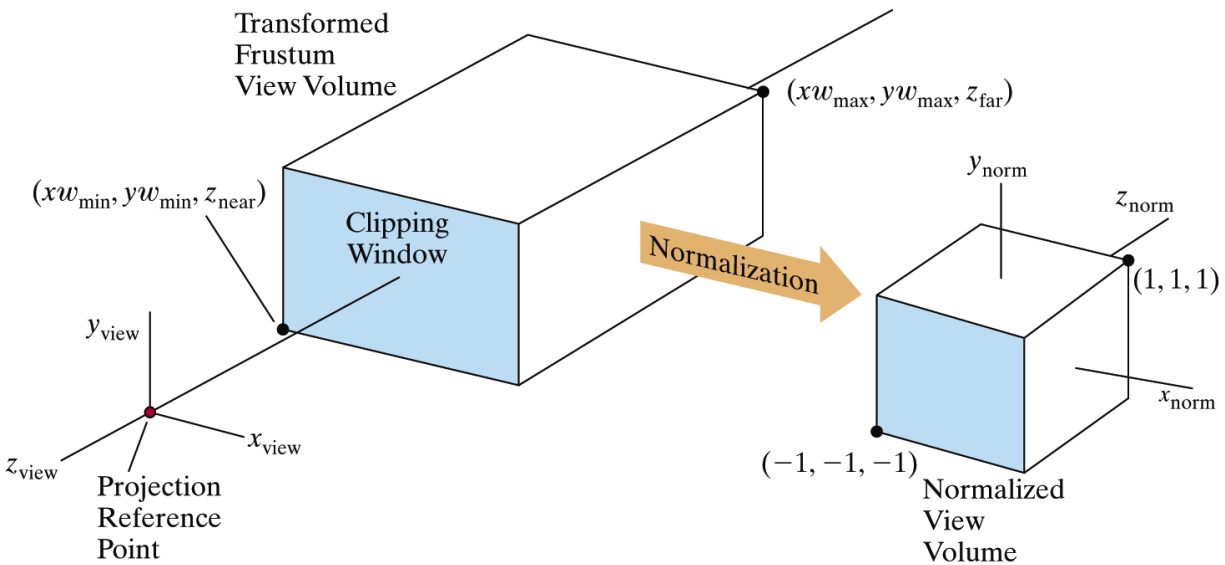
Observação importante

- Para um *frustum* simétrico, a transformação perspectiva mapeia o *viewing frustum* em um paralelepípedo retangular (equivalente ao *viewing volume* de uma projeção ortogonal)



Transformação de Projeção Perspectiva Normalizada

- A última etapa da projeção perspectiva é mapear o paralelepípedo obtido para um **viewing volume normalizado**
  - O procedimento é equivalente ao aplicado para a projeção paralela



Transformação de Projeção Perspectiva Normalizada

- Os parâmetros para a normalização da coordenada z já foram incluídos na matriz de projeção perspectiva, mas ainda precisam ser definidos

- Também é necessário definir os parâmetros para a normalização das coordenadas x e y
  - Não precisa de translação, porque a linha central do paralelepípedo retangular coincide com o eixo z_{view}, basta uma escala com relação à 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

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

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

Transformação de Projeção Perspectiva Normalizada

- Considerando o centro de projeção na origem, i.e., $(x_{prp}, y_{prp}, z_{prp}) = (0, 0, 0)$ e o plano de projeção coincidente com o plano de recorte próximo, *near* $z_{vp} = z_{near}$, então podemos reescrever a matriz final

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

Transformação de Projeção Perspectiva Normalizada

- Com esta transformação obtemos as coordenadas homogêneas do ponto projetado

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

- 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 obter a projeção normalizada no *viewing volume* canônico, queremos definir uma transformação que mapeia o ponto $(x_{wmin}, y_{wmin}, z_{near})$ em $(-1, -1, -1)$, e o ponto $(x_{wmas}, y_{wmax}, z_{far})$ em $(1, 1, 1)$

- 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 de visão 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 dos vértices dados em coordenadas do mundo (WCS) para o volume d eprojeção perspectiva normalizado é dada pela composição

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

- Uma vez aplicada essa matriz, as coordenadas foram transformadas e as rotinas de recorte podem ser executadas em relação ao volume canônico

Sumário

Transformação de Viewport e Coordenadas de Tela 3D

- Uma vez definido o *viewing volume*, seu conteúdo pode ser mapeado para a janela dada em coordenadas do dispositivo (*viewport*)

- O processo é semelhante ao visto no caso 2D, porém é preciso preservar a informação de profundidade para os testes de visibilidade no momento de *rendering*
  - Os valores da variável z são normalizados no intervalo $[0, 1]$, assumindo que a tela está em $z = 0$

$$\mathbf{M}_{normviewvol,3Dscreen} = \begin{bmatrix} \frac{xv_{max}-xv_{min}}{2} & 0 & 0 & \frac{xv_{max}+xv_{min}}{2} \\ 0 & \frac{yv_{max}-yv_{min}}{2} & 0 & \frac{yv_{max}+yv_{min}}{2} \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformação de Viewport e Coordenadas de Tela 3D

- Em coordenadas normalizadas, a coordenada $z_{norm} = -1$ vai ser mapeada para a coordenada na tela $z_{screen} = 0$
  - A *Viewport* 3D é definida como o volume de $(xv_{min}, yv_{min}, 0)$ a $(xv_{max}, yv_{max}, 1)$

- As posições x e y são enviadas para o **frame buffer** (contém a 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

Funções OpenGL para Viewing 3D

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

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

Sumário

Função de Projeção Ortogonal OpenGL

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

```
1 gl.glMatrixMode(GL.GL_PROJECTION);
```

- Para 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 posições dos planos de recorte *near/far*

Função de Projeção Ortogonal OpenGL

- Ass ume que o plano de projeção coincide 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* adota os seguintes valores para essa função

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

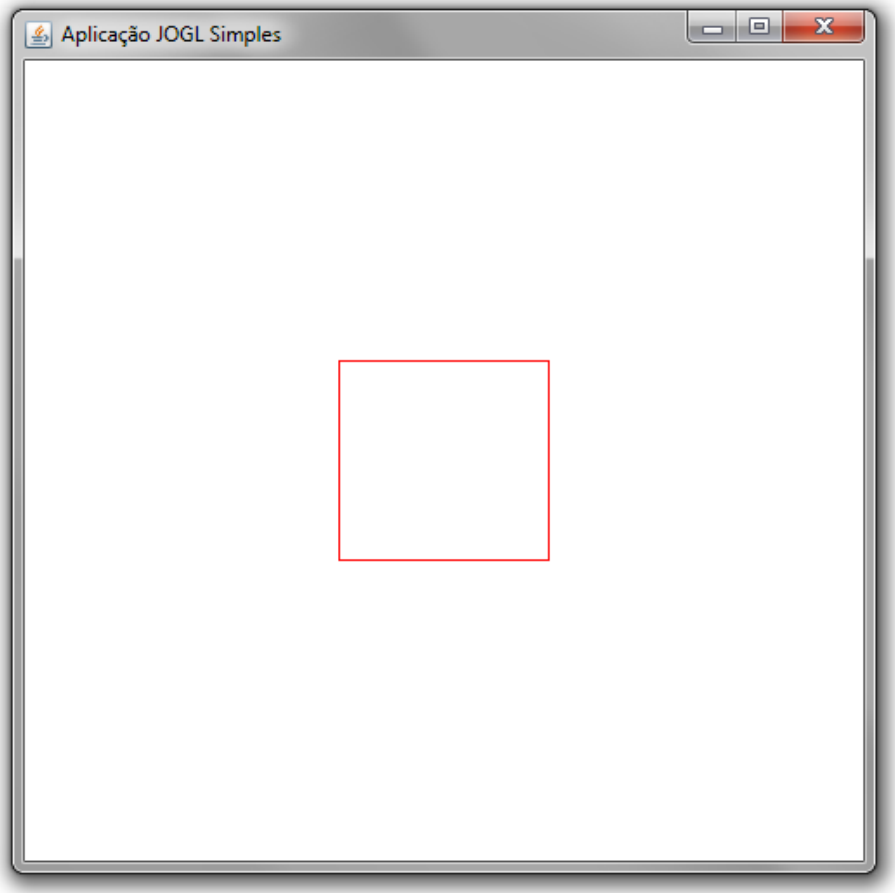
Função de Projeção Ortogonal OpenGL

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

Função de Projeção Ortogonal OpenGL



Sumário

Função para Projeção Perspectiva Simétrica OpenGL

- Para criar um *viewing frustum* simétrico em relação à direção de observação (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 (altura e largura) da janela de recorte
- $0^0 \leq theta \leq 180^0$ define o ângulo do campo de visão, $aspect$ é o valor da razão de aspecto $width/height$ da janela de recorte
- Os parâmetros $dnear$ e $dfar$ especificam as distâncias entre o ponto de observação (origem do sistema de coordenadas) e os planos de recorte *near* e *far*

Função para Projeção Perspectiva Simétrica OpenGL

- O ponto de referência da projeção (centro de projeção, posição da câmera) é a origem do VCS
- O plano de projeção coincide com o plano de recorte *near*

- Os planos de recorte *near/far* devem ser posicionados ao longo da direção negativa do eixo z_{view}, e não podem estar atrás da posição de obervação
- Os valores devem ser $dnear > 0$ e $dfar > 0$

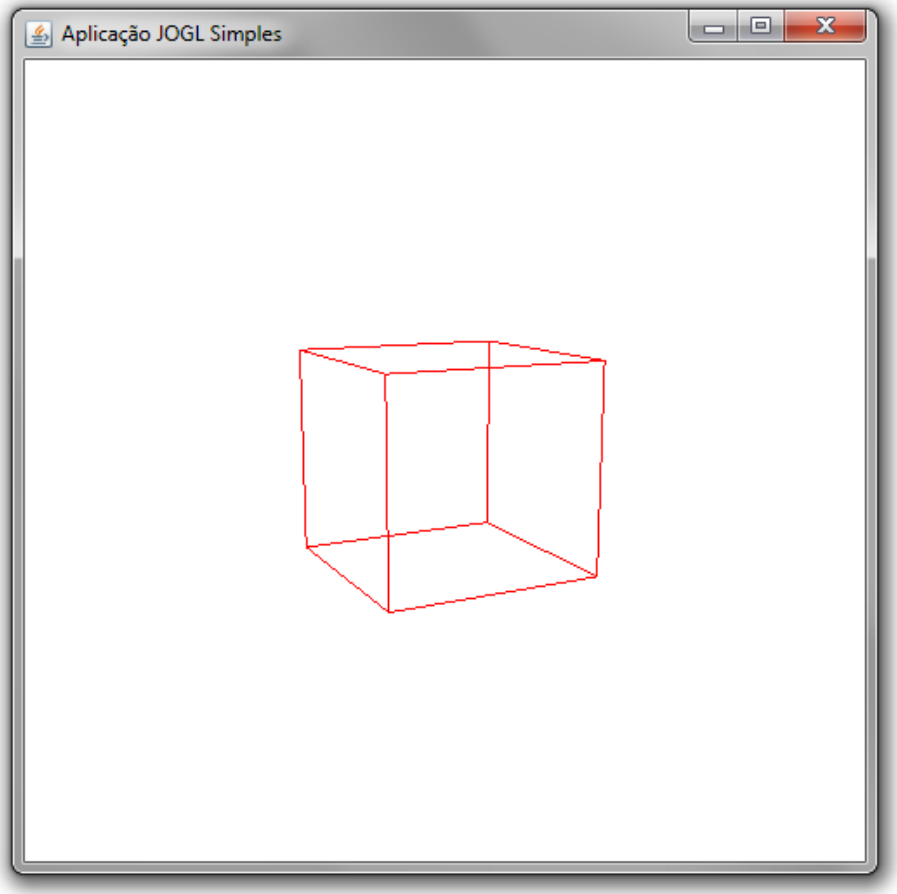
Função para Projeção Perspectiva Simétrica OpenGL

```
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  a de projeo
9    gl.glLoadIdentity(); //carrega a matrix de identidade
10   glu.gluPerspective(45.0, 1.0, 0.1, 10.0); //define uma projeo 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  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, //posio da cmera
13                 0.0, 0.0, 0.0, //para onde a cmera 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   //fora o desenho das primitivas
21   gl.glFlush();
22 }
```

Função para Projeção Perspectiva Simétrica OpenGL



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

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

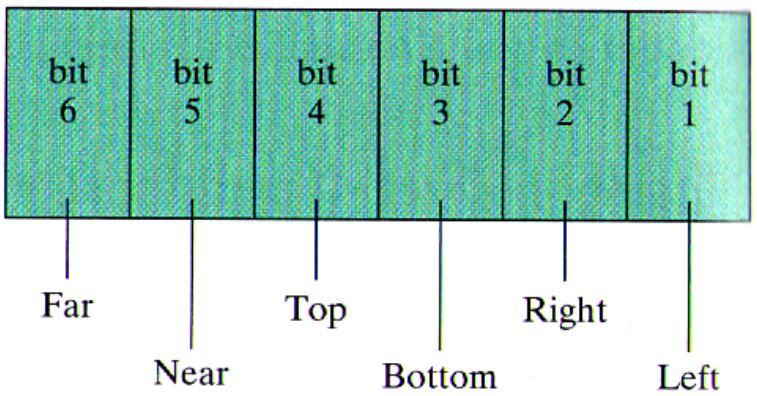
$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \mathbf{M} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 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

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

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

Código de Região 3D

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

$$bit_1 = 1 \text{ se } h + x_h < 0 \text{ (esquerda)}$$

$$bit_2 = 1 \text{ se } h - x_h < 0 \text{ (direita)}$$

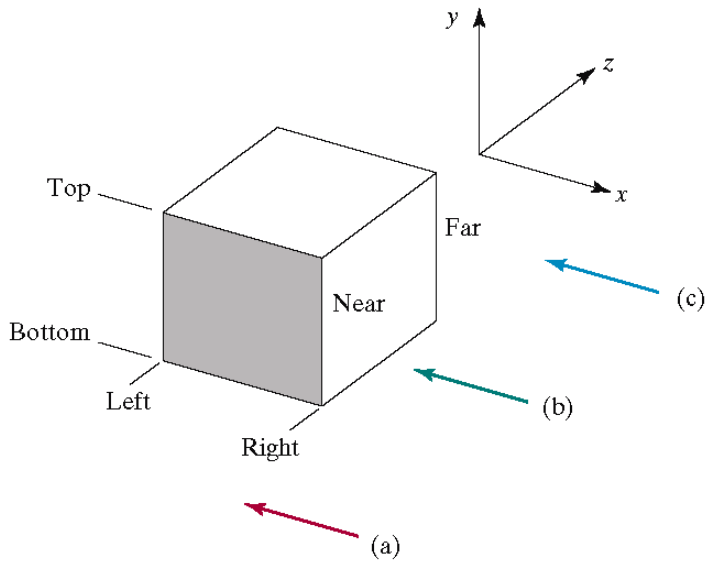
$$bit_3 = 1 \text{ se } h + y_h < 0 \text{ (inferior)}$$

$$bit_4 = 1 \text{ se } h - y_h < 0 \text{ (superior)}$$

$$bit_5 = 1 \text{ se } h + z_h < 0 \text{ (near)}$$

$$bit_6 = 1 \text{ se } h - z_h < 0 \text{ (far)}$$

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

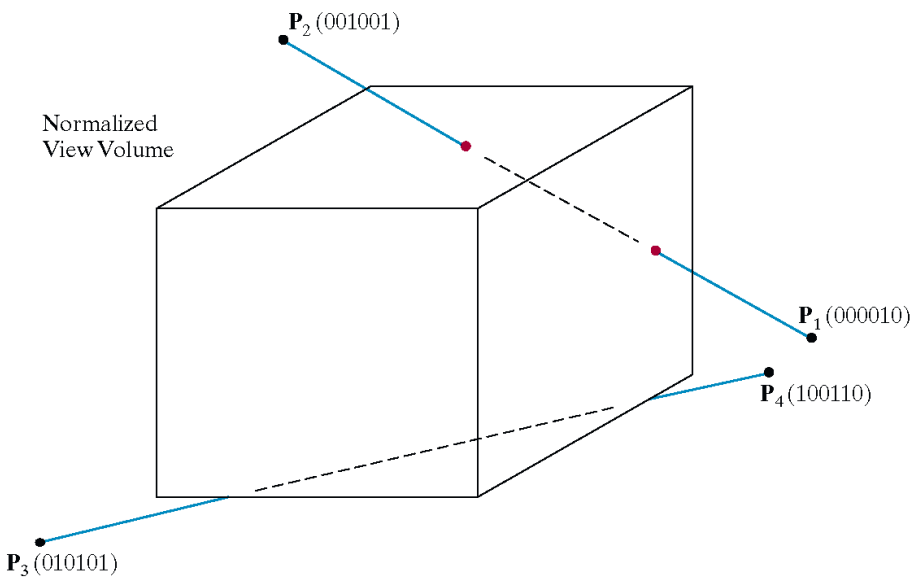
Region Codes
Between Near and Far Planes
(b)

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

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

- Ou seja

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

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

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

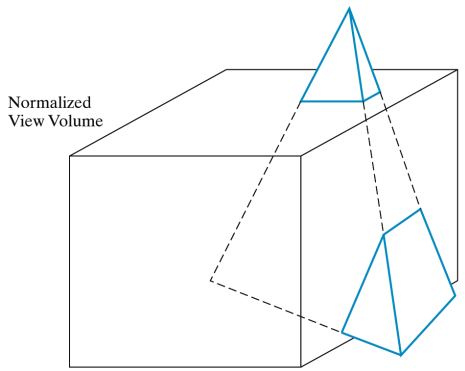
$$h = h_1 + (h_2 - h_1)u$$

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