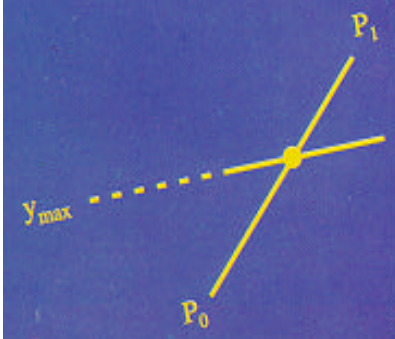
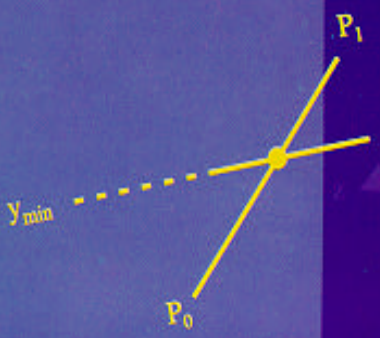
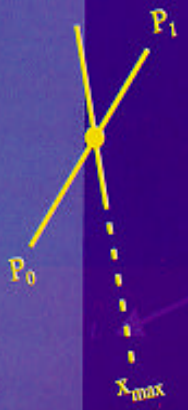


INTRODUÇÃO À COMPUTAÇÃO GRÁFICA

FERNANDO LUIZ B. RIBEIRO





COPPE/UFRJ

ÍNDICE

INTRODUÇÃO

A computação gráfica consiste na sintetização da imagem de objetos reais ou imaginários a partir de seus modelos descritivos. O processamento de imagens, por outro lado, trata do processo inverso: a reconstrução de modelos 2D ou 3D a partir de suas imagens. Técnicas tais como filtragem [10] e reconhecimento de padrões [11,12] estão relacionadas ao processamento de imagem. Embora tenham sido tratadas em separado até recentemente, a interseção destas duas áreas vem aumentando de modo crescente. São inúmeras as aplicações da computação gráfica, como por exemplo:

- Desenvolvimento de interfaces gráficas (window managers).

- Em aplicações CAD (computer-aided design), a computação gráfica é utilizada no projeto de componentes mecânicos, elétricos, incluindo projetos de estruturas, automóveis, aviões, navios, plantas de arquitetura, etc.

- Visualização científica.

- Produção de desenhos animados, comerciais, efeitos especiais na televisão e no cinema.

- Cartografia.

Na área de computação científica, onde predominam os métodos dos elementos finitos e de diferenças finitas, a utilização de programas para a visualização de dados numéricos (visualização científica) tornou-se obrigatória. Estes programas possibilitam a representação gráfica estática e animada de grandezas escalares, vetoriais e de partículas.

Os principais dispositivos de saída gráfica são os monitores de vídeo. Estes dispositivos podem ser classificados em duas categorias:

- dispositivos VETORIAIS

- dispositivos tipo RASTER

Tanto na computação gráfica propriamente dita quanto no processamento de imagem, os dispositivos de vídeo mais utilizados hoje são os dispositivos do tipo *raster*.

Um dispositivo tipicamente vetorial consiste em um processador de controle, um buffer de memória e um CRT (tubo de raios catódicos). O buffer armazena os comandos para o traçado de pontos, linhas e caracteres. Estes comandos são ciclicamente interpretados pelo processador de controle e enviados ao CRT, de modo que o canhão de elétrons se mova de um ponto para outro formando segmentos de reta na tela (Figura 1).

Além dos componentes de um dispositivo vetorial, os sistemas do tipo raster possuem ainda uma controladora de vídeo. A imagem é formada por uma matriz de pixels representando a tela. A matriz de pixels é armazenada no buffer, que por sua vez é

rastreado permanentemente pela controladora de vídeo, linha por linha (Figura 2). Para cada pixel, a intensidade do canhão de elétrons é dada por sua cor. Em sistemas coloridos, três canhões correspondentes às cores primárias vermelho, verde e azul são controlados de modo a fornecer a cor desejada. Em sistemas mais simples, como os PC's, o papel desempenhado pelo processador de controle é realizado via software. O termo *bitmap* é empregado para designar o conteúdo binário do buffer de memória. Em sistemas monocromáticos, cada pixel da tela é representado por por um bit do buffer (0-apagado ou 1-aceso). Em consequência, para representar uma tela monocromática de resolução 1024x1024 pixels são necessários 131072 bytes ou 128 Kb. Os sistemas com representação de 8, 16 e 24 bits por pixel permitem a utilização simultânea, respectivamente, de 256 (2^8), 65536 (2^{16}) e 16777216 (2^{24}) cores. O termo *frame buffer* também é empregado para designar o buffer de memória.

Devido à representação discreta da tela nos sistemas tipo raster, tornou-se imprescindível o desenvolvimento de técnicas adequadas (técnicas de raster) para o traçado de primitivas tais como linhas, polígonos, curvas, preenchimento de áreas e também para a manipulação de imagens armazenadas sob o formato de *bitmap*. Este assunto é abordado no capítulo 1 do presente texto, onde os algoritmos discutidos são implementados em linguagem C de programação. No capítulo 2 são estudadas transformações geométricas em duas e três dimensões, introduzindo-se o conceito de coordenadas homogêneas, que permitem uma padronização das operações matriciais de transformação essenciais para a representação gráfica dos objetos. As ferramentas descritas neste capítulo são utilizadas no capítulo 3, que trata da visualização tridimensional de objetos. Este capítulo inicia com um breve estudo sobre projeções, apresentando em seguida um modelo de visualização tridimensional. O modelo apresentado é similar ao do padrão PHIGS (*Programmer's Hierarchical Interactive Graphics System*), e compreende a definição de um volume de visualização e a caracterização das transformações necessárias para a representação de um objeto qualquer em um dispositivo de saída gráfica, seja este vetorial ou do tipo raster. Ao final do capítulo apresenta-se um algoritmo clássico para o recorte de polígonos em três dimensões. O capítulo 4 apresenta as principais formas de representação paramétrica de curvas e superfícies, encerrando com a descrição de estruturas de dados para o tratamento de superfícies poliédricas. Finalmente, o capítulo 5 aborda o tema da eliminação de superfícies ocultas, descrevendo os algoritmos de *prioridades, Z-buffer e ray-tracing*.

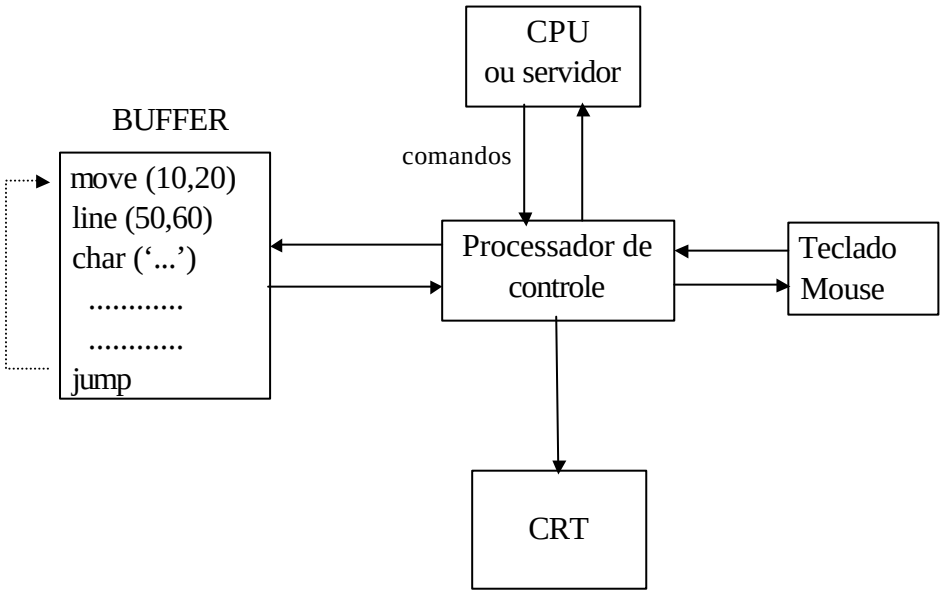


Figura 1 - Dispositivo de vídeo vetorial.

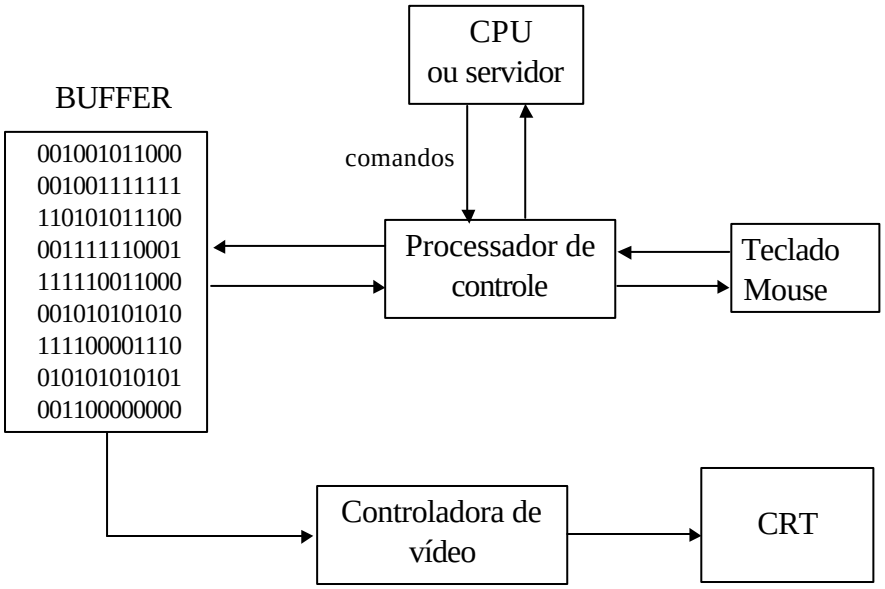


Figura 2 - Dispositivo do tipo raster.

1 TÉCNICAS DE RASTER

1.1 CONVERSÃO DE SEGMENTOS DE RETA

A conversão de segmentos de reta (ou simplesmente linhas) em um grid 2D de pixels consiste em determinar quais os pixels que fornecem a melhor aproximação do segmento (ver Figura 1.1). Este processo é chamado conversão por varredura.

As características essenciais de um algoritmo de conversão de linhas devem ser, em primeiro lugar, que o segmento convertido comece e termine exatamente sobre as extremidades $P_0(x_0, y_0)$ e $P_1(x_1, y_1)$. Esta característica implica que as coordenadas (x_0, y_0) e (y_0, y_0) devem ser definidas como inteiros. Em segundo, que os pixels selecionados quando o segmento tem orientação $P_0 \rightarrow P_1$ sejam os mesmos selecionados no caso em que o segmento tem orientação inversa, ou seja, os pixels ligados devem independer da orientação da reta.

A seguir são descritos dois algoritmos, o algoritmo básico incremental [1,2] e o algoritmo do ponto médio [1], sendo este último equivalente ao algoritmo de Bresenham [2,3].

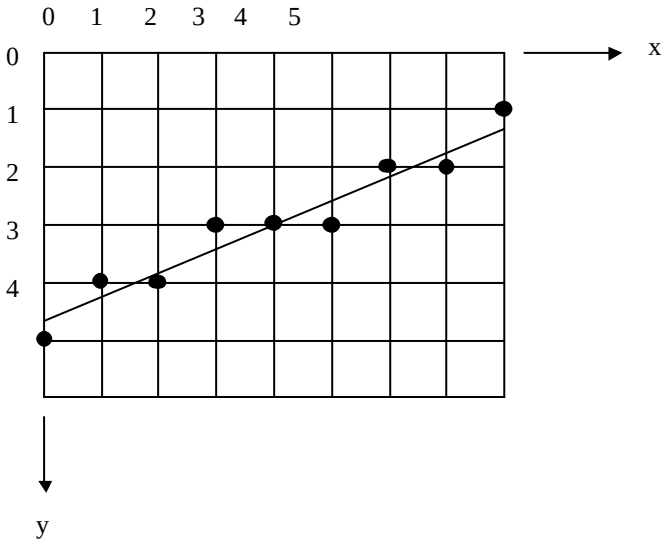


Figura 1.1 - Conversão de segmentos de reta em pixels.

1.1.1 Algoritmo básico incremental

O algoritmo básico incremental, também conhecido como DDA (digital differencial analyzer), assume as seguintes hipóteses:

a) O ponto extremo (x_0, y_0) encontra-se à esquerda de (x_1,y_1).

b) A inclinação $m = (y_1-y_0) / (x_1-x_0)$ é tal que : $|m| \leq 1$.

A Figura 1.2 mostra este algoritmo, escrito em linguagem C. A função *line* chama a função fictícia *writepixel*, que desempenha o papel de ligar um pixel com uma determinada cor. Esta função não faz parte do padrão C, e depende do compilador utilizado. As declarações de funções (protótipos de funções) e argumentos são feitas seguindo o padrão C++. Assume-se que o atributo cor possa ser representado por uma variável inteira *color*. Para inclinações $|m| > 1$ os papéis desempenhados por x e y devem ser invertidos. O algoritmo DDA tem a desvantagem de tratar as variáveis y e m como reais.

```
include <math.h>

/*  Declaração de funções:  */

int round(float x);

/*  Rotina para a conversão de linhas pelo método DDA :        */

void line (int x0,int y0,int x1,int y1,int color)
{
        int x, round( );
        float dx, dy, y, m;

        dx = x1 - x0;
        dy = y1 - y0;
        m  = dy / dx;

        y = y0;
        for ( x = x0; x < = x1; x++)  {
                writepixel(x,round(y),color);
                y = y + m;
        }
}

/*  Rotina para o arredondamento de um número real:      */

int round(float x)
{
        return( (int)floor(x+0.5) );
}
```

Figura 1.2 - Algoritmo para conversão de linhas pelo método DDA.

1.1.2 Algoritmo do ponto médio

O algoritmo do ponto médio utiliza apenas aritmética inteira e produz o mesmo resultado, ou seja, gera os mesmos pixels, que o algoritmo de Bresenham.

Considere inicialmente uma reta de inclinação $0 \leq m \leq 1$, sendo (x_0,y_0) o ponto extremo esquerdo inferior e (x_1,y_1) a extremidade direita superior (Figura 1.3). Assumindo que o pixel $P(x_p,y_p)$ tenha sido selecionado, o próximo passo consiste em escolher entre os pixels E (um incremento à direita de P) e NE (um incremento à direita e um incremento acima de P), como mostra a Figura 1.4.

Seja Q o ponto de interseção entre a linha que se deseja traçar e a vertical do grid $x = x_p + 1$. A técnica do ponto médio consiste em verificar em que lado da reta o ponto médio M se situa. Se o ponto M se encontra acima da linha, então o pixel E é o mais próximo da linha. Se M está abaixo da reta, então o pixel NE é o mais próximo da reta.

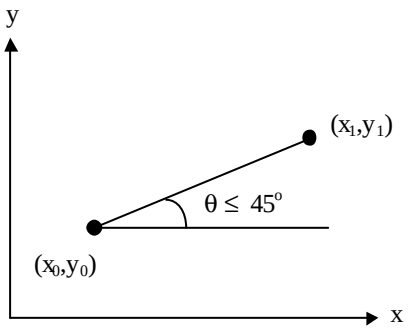


Figura 1.3 - Reta no 1º octante.

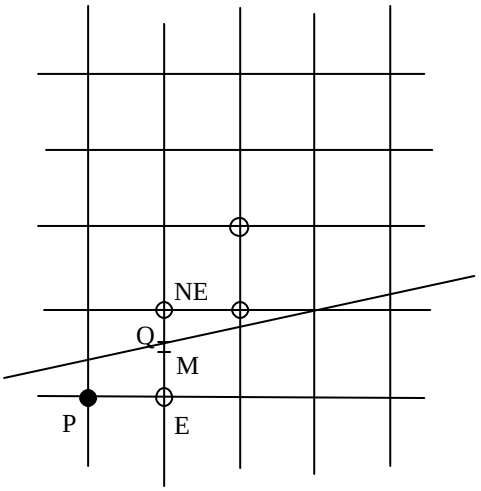


Figura 1.4 - Critério de seleção de um pixel.

A linha pode passar entre NE e E, ou ambos os pixels podem estar de um mesmo lado da linha, mas, em qualquer das hipóteses, o teste do ponto médio seleciona o pixel mais próximo. O erro, ou seja, a distância vertical entre o pixel e a linha, será sempre ≤ 1/2. Representando a reta na forma,

$$F(x,y) = ax+by+c = 0 \tag{1.1}$$

e comparando com a equação:

$$y = \frac{dy}{dx} x + B \tag{1.2}$$

pode-se verificar que os coeficientes de (1.1), a, b e c são iguais a:

$$a = dy$$

$$b = - dx$$

$$c = B\,dx$$

sendo $dx = x_1 - x_0$ e $dy = y_1 - y_0$. A função F(x,y) é igual a zero para os pontos da reta, positiva para os pontos abaixo da reta, e negativa para os pontos acima da reta. Para aplicar o critério do ponto médio deve-se, portanto, utilizar uma variável de teste d igual a:

$$d = F(M) = F(x_p+1, y_p+1/2) \tag{1.3}$$

Utilizando a expressão (1.1) obtém-se :

$$d = a \,.\, (x_p+1) + b \,.\, (y_p+1/2) + c \tag{1.4}$$

Se d > 0, seleciona-se o pixel NE, e se d ≤ 0 seleciona-se o pixel E. O ponto médio M e o valor de d para a linha vertical seguinte do grid, $x = x_p + 2$, dependem ambos da escolha entre E e NE. Se E foi escolhido, o ponto M é incrementado em 1 na direção x:

$$d_{new} = F(x_p+2, y_p+1/2) = a \,.\, (x_p+2) + b \,.\, (y_p+1/2) + c \tag{1.5}$$

Fazendo,

$$d_{old} = a \,.\, (x_p+1) + b \,.\, (y_p+1/2) + c \tag{1.6}$$

e subtraindo d_{old} de d_{new} obtém-se a diferença incremental ΔE:

$$\Delta E = d_{new} - d_{old} = a \,.\, [(x_p+2)-(x_p+1)] = a = dy \tag{1.7}$$

Portanto, o próximo valor de d pode ser calculado de uma forma incremental fazendo-se:

$$d_{new} = d_{old} + \Delta E = d_{old} + a \tag{1.8}$$

Se o pixel NE foi o escolhido, o ponto M é incrementado em uma unidade em ambas as direções x e y, e então:

$$d_{new} = F(x_p + 2, y_p + 3/2) = a . (x_p + 2) + b . (y_p + 3/2) + c \qquad (1.9)$$

Neste caso, a diferença incremental ΔNE é dada por:

$$\Delta NE = d_{new} - d_{old} = a + b = dy - dx \qquad (1.10)$$

e, em consequência,

$$d_{new} = d_{old} + \Delta NE = d_{old} + a + b \qquad (1.11)$$

Em resumo, o algoritmo do ponto médio seleciona, em cada passo, um entre dois pixels, dependendo do sinal da variável de teste. A variável de teste é então atualizada somando-se ΔE ou ΔNE, conforme o pixel escolhido no passo anterior. O primeiro pixel é simplesmente a extremidade (x_0, y_0), e o primeiro valor de d é calculado através da expressão:

$$d_{start} = F (x_0+1, y_0+1/2) = a (x_0+1) + b (y_0+1/2) + c =$$

$$= F (x_0, y_0) + a + b / 2 \qquad (1.12)$$

Como (x_0, y_0) é um ponto da reta, então a expressão acima se reduz a:

$$d_{start} = a + b/2 = dy - dx/2 \qquad (1.13)$$

Para eliminar a fração em (1.13), pode-se multiplicar a função F(x, y) por 2, observando-se que a multiplicação de F por uma constante positiva não altera o sinal da variável de teste:

$$F(x, y) = 2 (a x + b y + c) \qquad (1.14)$$

Pode-se então escrever:

$$d_{start} = 2 a + b = 2 dy - dx \qquad (1.15)$$

$$\Delta E = 2 a = 2 dy \qquad (1.16)$$

$$\Delta NE = 2 (a + b) = 2 (dy - dx) \qquad (1.17)$$

O algoritmo da Figura 1.5 sintetiza o desenvolvimento acima. Na generalização do algoritmo do ponto médio para qualquer inclinação deve-se assegurar que o segmento de reta P_0P_1 contém os mesmos pixels que o segmento P_1P_0. A única situação em que a escolha do pixel depende da direção da reta é quando a linha passa exatamente pelo ponto

médio M, ou seja, quando d = 0. Portanto, quando se caminha da direita para a esquerda, deve-se selecionar o pixel SW para d = 0, como mostra a Figura 1.6. A Figura 1.7 ilustra as oito possibilidades a serem consideradas na generalização do algoritmo.

```
/*  Rotina para conversão de linhas pelo método do ponto médio :    */

midpoint_line ( int x0, int y0, int x1, int y1, int color )
{
        int dx, dy, incrE, incrNE, d, x, y ;

        dx  =  x1  -  x0 ;
        dy  =  y1  -  y0 ;
        d   =  2 * dy  -  dx ;
        incrE    =  2 * dy ;
        incrNE  =  2 * ( dy  -  dx ) ;
        y  =  y0 ;

        for ( x = x0 ;  x < x1 ;  x++ )  {
                writepixel ( x, y, color );
                if ( d > 0 )  {
                        d  =  d + incrNE ;
                        y++;  }
                else
                        d  =  d + incrE ;
        }
        writepixel ( x, y, color ) ;
}
```

Figura 1.5 - Conversão de linhas pelo método do ponto médio.

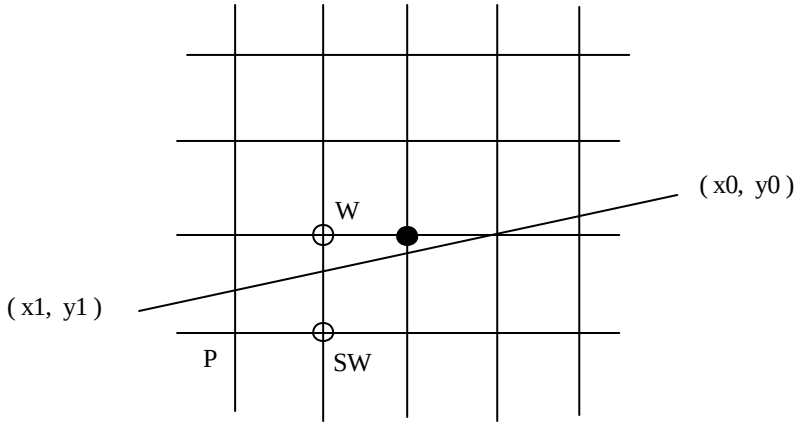
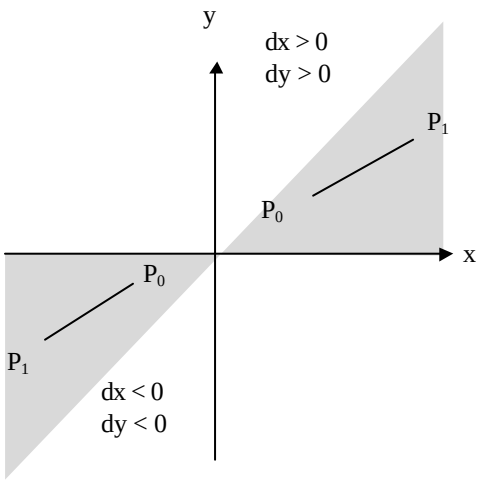
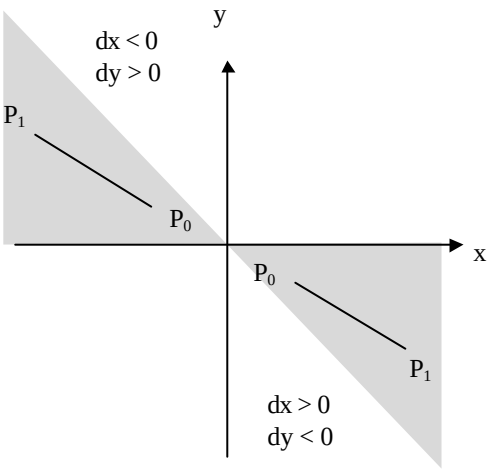


Figura 1.6 - Sentido direita-esquerda na conversão de linhas.

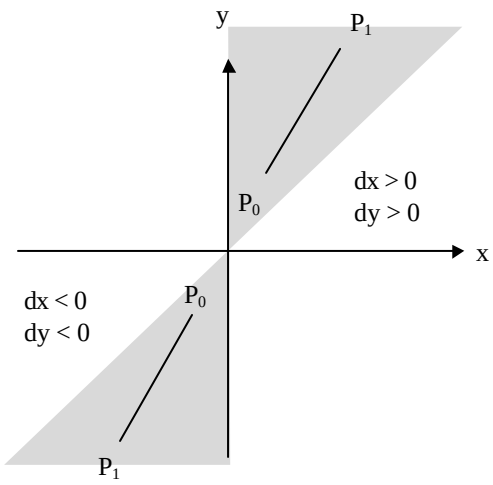
(a) $0 \leq m \leq 1$



(b) $-1 \leq m \leq 0$



(c) $m > 1$



(d) $m < -1$

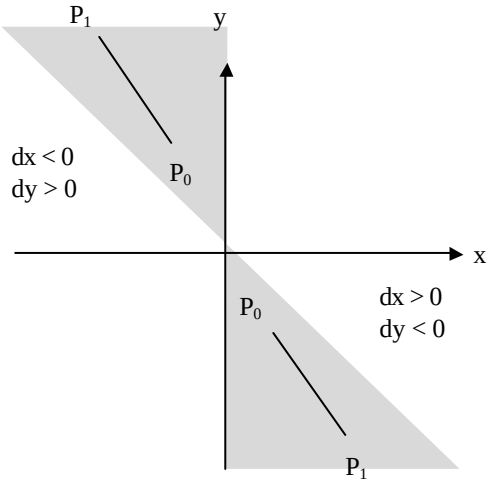


Figura 1.7 - Inclinação de retas nos oito octantes.

1.2 PREENCHIMENTO DE POLÍGONOS

O algoritmo descrito a seguir permite a conversão por varredura de polígonos côncavos e convexos [1,2]. O algoritmo opera calculando os intervalos das linhas de varredura que residem no interior do polígono, como mostra a Figura 1.8.

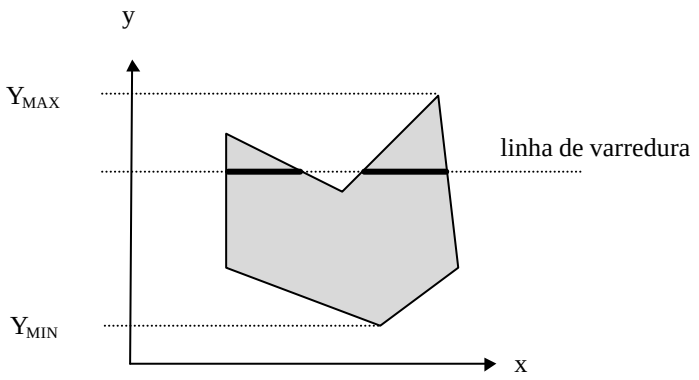


Figura 1.8 - Linhas de varredura para preenchimento de polígonos.

Seja um polígono de n arestas, das quais somente p arestas não são horizontais. Deve-se inicialmente construir uma tabela contendo, para cada aresta não horizontal, os valores mínimos e máximos de y, o valor de x correspondente ao ponto de y mínimo (x_{min}), e o inverso da inclinação (Figura 1.9).

Em seguida, para cada linha de varredura y ($Y_{MIN} \leq y < Y_{MAX}$), estas arestas são ordenadas de acordo com suas coordenadas x_{min} , em ordem crescente, e são ativados os pixels dos segmentos de reta horizontais entre pares de arestas de paridade par (Figura 1.10). Os valores de x_{min} são atualizados a cada incremento em y. Para ordenar as arestas pode-se utilizar um ordenador do tipo bolha (bubble sort).

aresta	y_{min}	y_{max}	x de $y_{min} = x_{min}$	$m = \Delta x / \Delta y$
0				
1				
p - 1				

Figura 1.9 - Lista de arestas não horizontais.

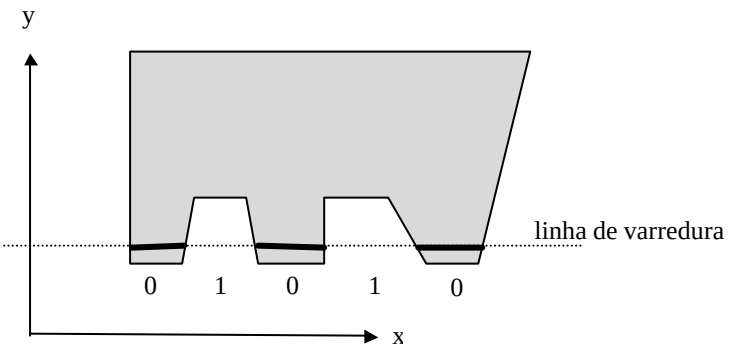


Figura 1.10 - Paridade de segmentos de reta horizontais entre arestas.

Este algoritmo é representado pela função *fillarea* da Figura 1.13. São chamadas as funções *ordena_xmin*, que ordena as arestas, e *horizontal*, que ativa os pixels de um segmento de reta horizontal entre pares de arestas. A função *writepixel*, já comentada antes, é utilizada pela função *horizontal*. Os parâmetros de entrada da função *fillarea* são dois arranjos, *x* e *y*, contendo as coordenadas dos vértices do polígono, e o número de vértices *nv*. A única restrição é que os vértices do polígono devem ser armazenados em um determinado sentido, horário ou anti-horário, formando uma poligonal fechada. Assim como na rasterização de linhas, os vértices são especificados através de variáveis inteiras. As coordenadas x das extremidades dos segmentos horizontais a serem rasterizados são tratadas como variáveis reais, que são arredondadas para inteiros na chamada da função *horizontal*.

Inicialmente, o algoritmo percorre todas as arestas, formando a tabela da Figura 1.9. Em seguida, verifica as arestas que interceptam cada linha de varredura, marcando a paridade de cada segmento horizontal entre interseções. Os vértices correspondentes aos valores máximos de y não são levados em conta nesta verificação, para evitar os erros que ocorreriam em situações como as da Figura 1.11. Na Figura 1.11a, se o vértice C da aresta BC fosse considerado como interseção da linha de varredura de coordenada y_C, o segmento CD teria paridade ímpar e não seria convertido. Na Figura 1.11b, o vértice E seria interpretado como um segmento de paridade ímpar, e o segmento EC' seria convertido, na linha de varredura y_E.

Pelo fato de não considerar os vértices com valores máximos em y como interseções, os pontos e segmentos pertencentes à linha de varredura $y = Y_{MAX}$ não são convertidos por este algoritmo, tal como este se encontra implementado. Portanto, as arestas EF e GH das Figuras 1.11a e 1.11b não são convertidas. Este problema pode ser resolvido tratando-se a linha de varredura $y = Y_{MAX}$ isoladamente. Para esta linha, os vértices das arestas seriam considerados como interseções.

Outro inconveniente deste algoritmo é a ordenação de todas as arestas, a cada linha de varredura. A eficiência do algoritmo pode ser melhorada construindo-se uma lista de arestas ativas, ou seja, uma lista que forneça para cada linha de varredura as arestas que a interceptam [1, 2].

Em cantos formados por arestas com inclinações menores que 1, em valor absoluto, o preenchimento fica comprometido, como mostra a Figura 1.12. Como os incrementos são feitos em y, as inclinações mais adequadas são maiores que 1. No entanto, este problema é corrigido se os contornos são convertidos por um algoritmo de segmentos de reta.

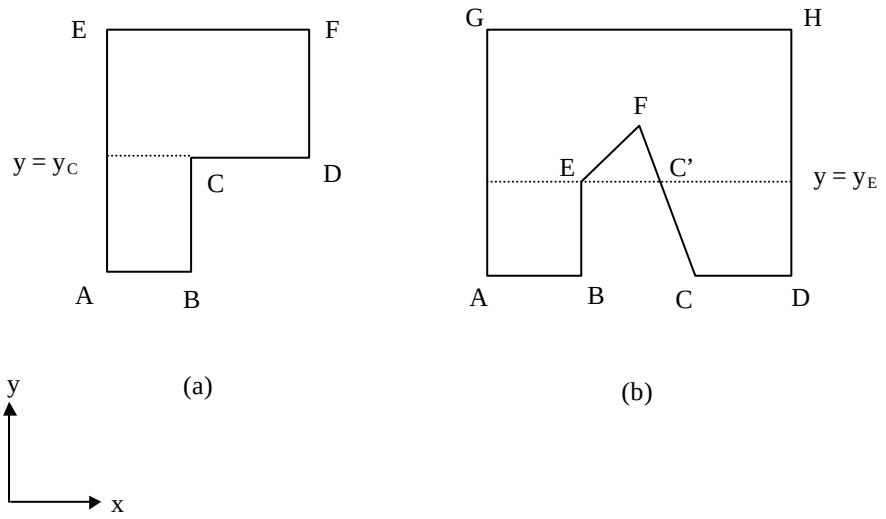


Figura 1.11 - Exemplos de erros que ocorrem quando os vértices de valores máximos em y são considerados. O segmento CD em (a) não seria convertido e o segmento EC' em (b) seria convertido.

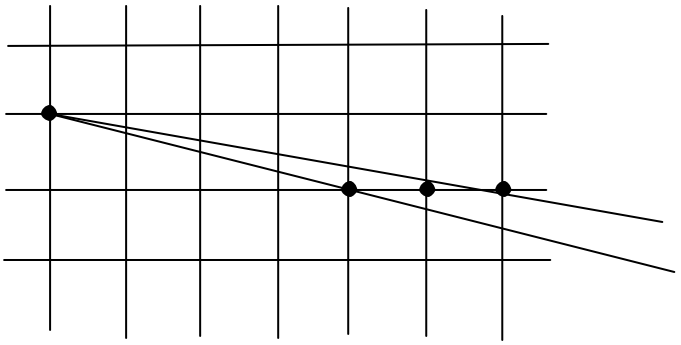


Figura 1.12 - Exemplo de falha no preenchimento de cantos: os pixels correspondentes às interseções das três verticais subsequentes ao vértice esquerdo com as arestas não são convertidos.

```
#include <stdlib.h>
#define MAX 100

/*  Declaração de funções :     */
        void OrdenaXmin(int p, float xi[], int ymin[], int ymax[], float m[]);
        void horizontal(int x0, int x1, int y);
        int round(float x);

/* Rotina para preenchimento de polígonos :   */
void fillarea(int nv,int x[],int y[])
{
        int YMIN, YMAX, yi, ymin[MAX], ymax[MAX], i, j, p = 0;
        float xi[MAX], m[MAX];
        char par, MASK = 1;

        YMIN = YMAX = y[0];
        i = nv-1;
        for (j = 0; j < nv; j++) {
                YMIN  = min(YMIN,y[j]);
                YMAX = max(YMAX,y[j]);
                if (y[j] > y[i]) {
                        ymin[p] = y[i];
                        ymax[p] = y[j];
                        xi[p]   = x[i];
                        m[p]    = (float)(x[j]-x[i])/(y[j]-y[i]);
                        p++; }
                else if (y[i] > y[j]) {
                        ymin[p] = y[j];
                        ymax[p] = y[i];
                        xi[p]   = x[j];
                        m[p]    = (float)(x[i]-x[j])/(y[i]-y[j]);
                        p++; }
                i = j;
        }
        for (yi = YMIN; yi < YMAX; yi++)    {
                OrdenaXmin(p,xi,ymin,ymax,m);
                par = 0;
                for (j = 0; j < p; j++)             {
                        if (yi >= ymin[j] && yi < ymax[j]) {
                                if (!par)
                                        i   = j;
                                else {
                                        horizontal(round(xi[i]),round(xi[j]),yi);
                                        xi[i] = xi[i] + m[i];
                                        xi[j] = xi[j] + m[j]; }
                                par = MASK^par; }
                }
        }
}
void horizontal(int x0, int x1, int y)
{
        int x;

        for (x = x0; x <= x1; x = x++)   writepixel(x, y, interior_color);
}
```

Figura 1.13 - Algoritmo para preenchimento de polígonos.

1.3 OPERAÇÕES COM BLOCOS

A maneira mais eficiente de fazer com que imagens (ícones, menus, etc) apareçam e desapareçam da tela rapidamente é através do armazenamento na memória dos bitmaps (ou pixmaps) correspondentes a estas imagens.

Na medida em que as imagens são requisitadas na tela, os blocos de pixels são copiados na posição e tamanho adequados. Frequentemente, utiliza-se a palavra *canvas* para designar uma estrutura de dados que armazena uma imagem na forma de uma matriz de pixels. Tais estruturas contém também informações a respeito do tamanho e atributos da imagens. Para operações com estas matrizes são necessárias funções que realizam a transferência de blocos de pixels de uma região para outra (Figura 1.14).

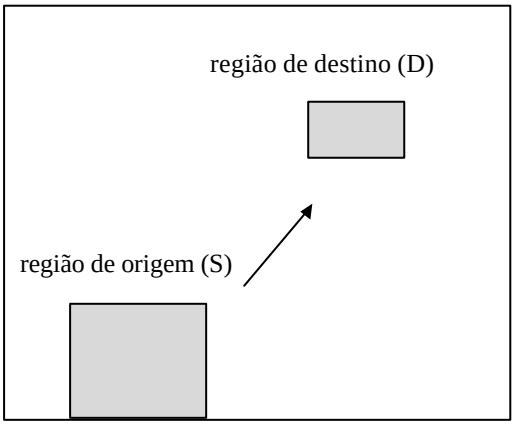


Figura 1.14 - Transferência de bitmaps.

1.3.1 Modos de escrita

As funções de transferência de blocos de pixels executam operações lógicas entre os pares de pixels correspondentes da região de origem (S) e da região de destino (D). Estas operações são simbolizadas por:

$$D \leftarrow (S \text{ op } D) \tag{1.18}$$

onde op é um operador booleano que representa o modo de escrita dos pixels. Os operadores mais utilizados são:

REPLACE
OR
XOR
AND

Os efeitos produzidos por estes modos são ilustrados na Figura 1.15.

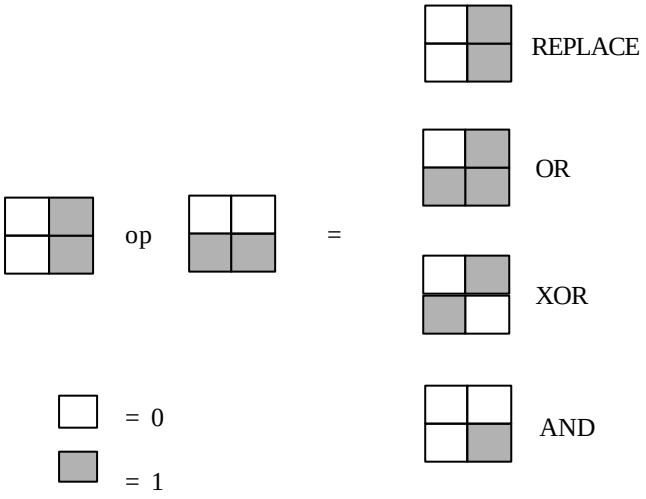


Figura 1.15 - Efeitos produzidos pelos modos de escrita.

O modo REPLACE é um modo destrutivo, uma vez que as imagens são excritas sobre o que já se encontra na tela. É o modo normal para o traçado de primitivas e menus. Pode também ser utilizado para deletar primitivas, redesenhando-as com a cor de fundo. O modo OR é parte destrutivo e parte não-destrutivo. O modo XOR é muito usado na implementação de objetos dinâmicos, tais como cursores, janelas deslizantes, etc. Uma primeira operação XOR inverte a imagem da região de destino, e uma segunda operação XOR restaura o pixel original. O modo AND pode ser utilizado para desativar seletivamente pixels da região de destino.

1.3.2 Escalonamento de imagem

O modo mais simples de mudar a escala de uma imagem armazenada no formato de bitmap é através da "replicação de pixels". Esta técnica consiste em substituir cada pixel por um bloco NxN de pixels. Consequentemente, a imagem é ampliada por um fator inteiro N (Figura 1.16). À medida em que as imagens aumentam, tornam-se cada vez mais pobres visualmente. Para reduzir a imagem são necessárias técnicas mais sofisticadas [1, 4, 5].

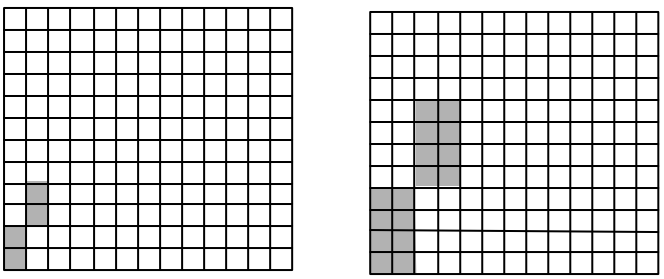


Figura 1.16 - Replicação de pixels.

1.4 PADRÕES DE PREENCHIMENTO

Os padrões de preenchimento são definidos em um pequeno bitmap MxN. Considerando-se, por exemplo, um determinado padrão armazenado na matriz *padrão*(M,N), o pixel *padrão*(0,0) é considerado coincidente com a origem da tela, e a repetição do padrão é obtida através do emprego de aritmética modular. Sendo assim, o pixel (x, y) de uma primitiva terá uma cor k igual a:

$$k = padrão[i][j]; \tag{1.19}$$

Os índices i e j da matriz acima são calculados através das expressões:

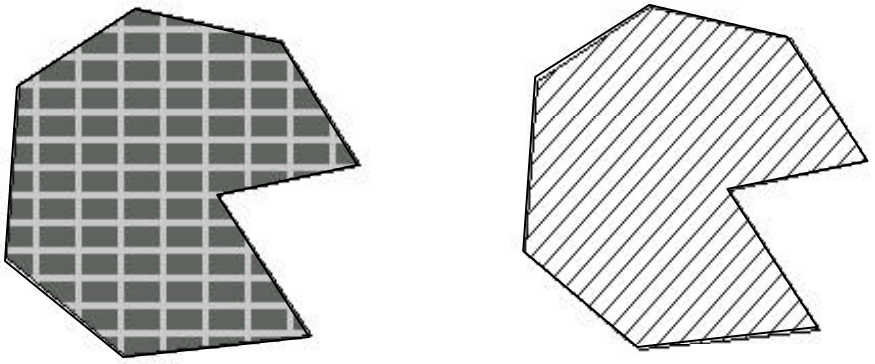
$$i = x \ \% \ N; \tag{1.20}$$

$$j = y \ \% \ M; \tag{1.21}$$

onde % é o operador aritimético de C que retorna o resto da divisão de dois inteiros.

A Figura 1.17b mostra o preenchimento de uma mesma área através de dois padrões diferentes, utilizando o algoritmo da Figura 1.13. Além da definição da matriz *padrão*(M, N), a única modificação feita no algoritmo foi a inclusão, na função *horizontal*, das três linhas de código acima.

(a) padrões de preenchimento



(b) efeito produzido

Figura 1.17 - Exemplos de padrões de preenchimento.

1.5 ESPESSURA DE SEGMENTOS DE RETA

A conversão de segmentos de reta com a consideração do atributo espessura pode ser implementada através de 3 métodos:

- método da replicação de pixels
- método da pena
- método de preenchimento de áreas entre contornos

No algoritmo de replicação, os pixels são replicados em colunas, para segmentos de reta com inclinações m tais que, $-1 \leq m \leq 1$, e em linhas para as demais inclinações. O método da pena consiste em posicionar o centro de uma pena imaginária de determinada seção transversal sobre os pixels convertidos (Figura 1.18). Pode-se por exemplo, utilizar penas de seção transversal retangular ou circular. A pena circular produz melhores resultados, uma vez que não depende da orientação da linha.

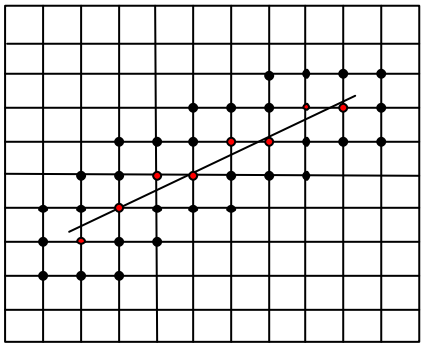


Figura 1.18 - Conversão de segmentos de reta com pena quadrada.

No método de preenchimento, os contornos externo e interno são traçados a uma distância t/2 do segmento, sendo t a espessura. O polígono resultante é então preenchido utilizando-se um algoritmo de preenchimento de polígonos (Figura 1.19).

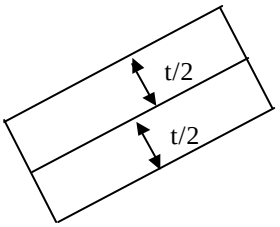


Figura 1.19 - Método de preenchimento.

1.6 RECORTE DE SEGMENTOS DE RETA

Um algoritmo bastante eficiente para o recorte (clipping) de segmentos de reta contra um retângulo de recorte (clip rectangle) é o algoritmo de Cohen-Sutherland [1,2]. O algoritmo testa, de início, os pontos extremos e verifica se o segmento se situa inteiramente em um dos 4 semi-planos externos ao retângulo (Figura 1.20) ou inteiro dentro do retângulo de recorte. Na primeira hipótese, o segmento é rejeitado e nenhuma conversão é efetuada. Na segunda hipótese, o segmento é convertido sem recortes. Caso nenhuma das hipóteses acima ocorra, a reta é dividida em dois segmentos, descartando-se o que se situa inteiramente em um dos 4 semi-planos (Figura 1.21). O novo segmento é então submetido ao teste inicial, procedendo-se desta maneira até que o segmento remanescente esteja situado inteiramente em um dos 4 semi-planos ou esteja inteiramente contido no retângulo de recorte. O algoritmo da Figura 1.22 descreve este procedimento.

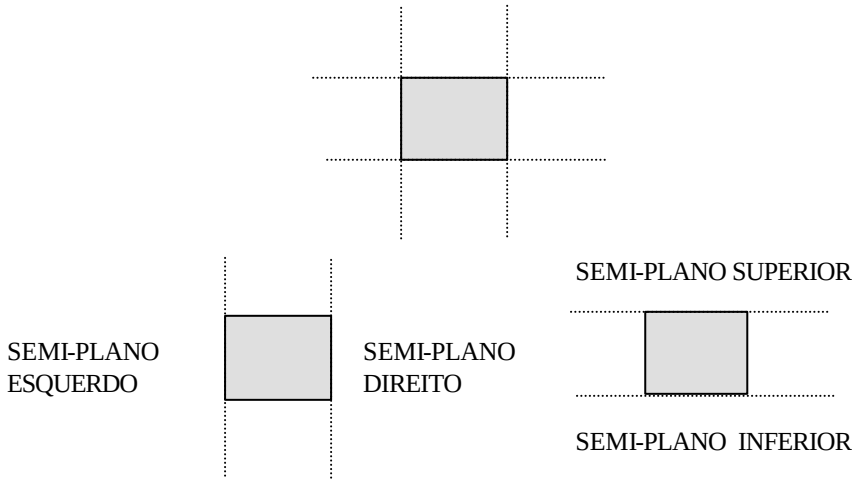


Figura 1.20 - Retângulo de recorte e os 4 semi-planos externos.

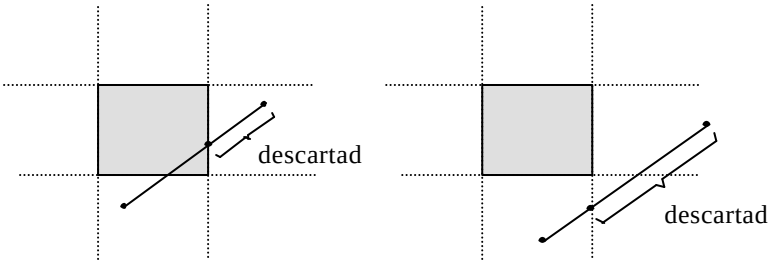


Figura 1.21 - Divisão dos segmentos.

```
1. Teste de aceitação:

        se (os dois pontos extremos se situam em um dos 4 semi-planos)
        {
                fim;
        }
        se (segmento inteiramente contido no retângulo)
        {
           converte segmento;
           fim;
        }

2. Divide o segmento em dois, tomando a interseção do segmento com uma
   das arestas (ou prolongamento) do retângulo;

3. Descarta o segmento situado inteiramente em um dos 4 semi-planos;

4. Retorna ao passo 1;
```

Figura 1.22 - Teste de aceitação de segmento.

Para implementar o teste de aceitação deve-se considerar as nove regiões formadas pelos prolongamentos das arestas do retângulo de recorte (Figura 1.23). A cada região é atribuído um código de 4 bits, que determina a posição de cada região em relação aos 4 semi-planos externos ao retângulo. Considerando 1 = TRUE e 0 = FALSE, a Figura 1.24 indica o significado destes bits. Os bits são numerados da esquerda para a direita.

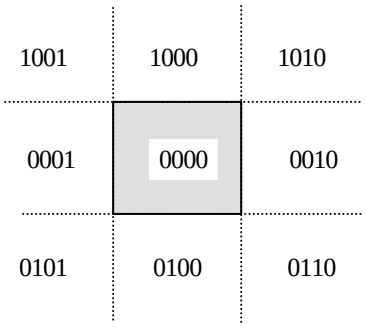


Figura 1.23 - Códigos das regiões.

BIT		
1	acima da aresta superior	$y > y_{max}$
2	abaixo da aresta inferior	$y < y_{min}$
3	à direita da aresta direita	$x > x_{max}$
4	à esquerda da aresta esquerda	$x < x_{min}$

Figura 1.24 - Significados dos bits.

A cada ponto extremo do segmento é atribuído um código, conforme a região à qual pertence. Se os dois pontos extremos possuem o código 0000, então o segmento está inteiramente contido no retângulo. Por outro lado, se o resultado da operação AND entre os dois códigos não for igual a zero, o segmento é rejeitado, ou seja, o segmento se situa inteiramente em um mesmo semi-plano (Figura 1.25). Se o segmento não passa no teste inicial, ou seja, não é convertido e nem rejeitado, deve-se então efetuar a subdivisão do segmento. Para isto utiliza-se uma aresta que intercepta o segmento, descartando-se o segmento que se situa inteiramente no semi-plano correspondente a esta aresta (Figura 1.26).

P_0	P_1	P_0 AND P_1	
0000	0000	0000	segmento no retângulo (convertido)
		$\neq 0$	segmento em um semi-plano (rejeitado)
		0000	segmento deve ser subdividido

Figura 1.25- Operação AND entre os dois pontos extremos do segmento.

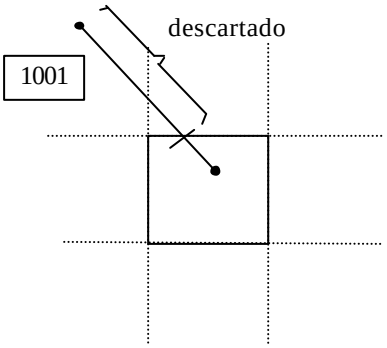
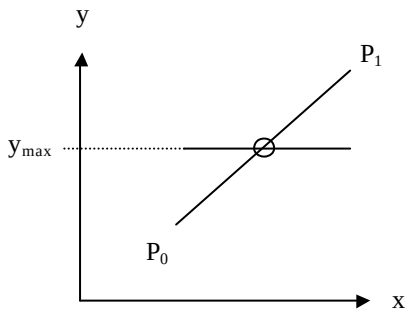


Figura 1.26- Divisão do segmento.

O loop nas arestas para a determinação da interseção com o segmento é feito na ordem superior-inferior, direita-esquerda, observando-se que os bits diferentes de zero dos pontos extremos indicam quais arestas interceptam o segmento (Figura 1.26). O algoritmo seleciona então um dos pontos extremos que se situa fora do retângulo (código diferente de zero), e toma o primeiro bit diferente de zero. A posição deste bit indica a aresta que intercepta o segmento, dividindo-o em dois, e o trecho que se situa inteiramente no semi-plano correspondente a esta aresta é descartado. As coordenadas dos pontos de interseção são calculadas de acordo com a aresta interceptada (Figura 1.27).

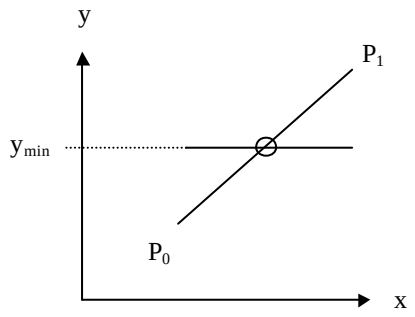
(a)



$$y = y_{max}$$

$$x = x_0 + \frac{(x_1 - x_0)}{(y_1 - y_0)}.(y_{max} - y_0)$$

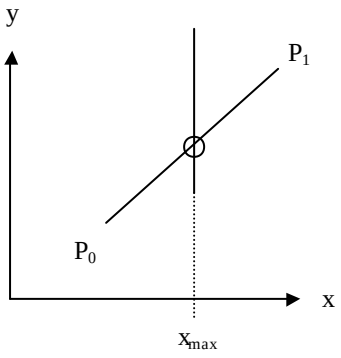
(b)



$$y = y_{min}$$

$$x = x_0 + \frac{(x_1 - x_0)}{(y_1 - y_0)}.(y_{min} - y_0)$$

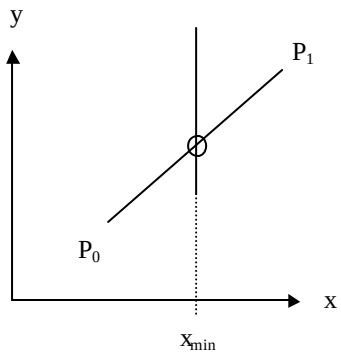
(c)



$$x = x_{max}$$

$$y = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}.(x_{max} - x_0)$$

(d)



$$x = x_{min}$$

$$y = y_0 + \frac{(y_1 - y_0)}{(x_1 - x_0)}.(x_{min} - x_0)$$

Figura 1.27- Coordenadas do ponto de interseção do segmento com a aresta superior (a), com a aresta inferior (b), com a aresta direita (c) e com a aresta esquerda (d).

A Figura 1.28 apresenta uma implementação em C do algoritmo acima descrito. A função principal *LineClip* tem como argumentos os ponteiros para as extremidades do segmento (x_0 , y_0), (x_1 , y_1) e os cantos inferior esquerdo e superior direito do retângulo de recorte (x_{min} , y_{min}) e (x_{max} , y_{max}). Ponteiros são utilizados como argumentos porque os valores das variáveis que armazenam as coordenadas das extremidades do segmento são alterados, se o segmento é recortado. Neste caso, as extremidades (x_0, y_0) e (x_1, y_1) mantêm a orientação original. A função *LineClip*, declarada como sendo do tipo *char*, pode retornar três valores, que são 1, no caso de segmentos situados total ou parcialmente no interior do retângulo de recorte, 0 no caso de segmentos localizados totalmente fora do retângulo, e -1 em casos excepcionais de erro. Este erro ocorre quando quatro passadas no laço principal da função não são suficientes para recortar o segmento.

Duas funções são chamadas pela função principal. A função *Ptcode* atribui um código de quatro bits a um ponto, como foi explicado anteriormente, e a função *Divide* divide o segmento em dois, descartando o que se situa inteiramente em um mesmo semi-plano.

```
/*      Declaração de funções :                 */

        int round(float x);
        char Ptcode(float x,float y,float xmin,float ymin,float xmax,float ymax);
        void Divide(float *x0,float *y0,float *x1,float *y1,char code0,char code1,
                                        float xmin,float ymin,float xmax,float ymax);

/* Recorte de linhas pelo método de Cohen-Sutherland.          */

char LineClip(float *x0,float *y0,float *x1,float *y1,float xmin,
                                        float ymin,float xmax, float ymax)
{
        char code0,code1,i;

        for (i = 0; i < 4; i++) {
                code0 = Ptcode(*x0,*y0,xmin,ymin,xmax,ymax);
                code1 = Ptcode(*x1,*y1,xmin,ymin,xmax,ymax);
                if (code0 == 0 && code1 == 0)       {
                /*      Segmento total ou parcialmente contido no retangulo !           */
                        return(1);
                }
                if ((code0 & code1)!= 0) {
                /*      Segmento totalmente fora do retangulo !       */
                        return(0);
                }
        /*      Divide o segmento:          */
                Divide(x0,y0,x1,y1,code0,code1,xmin,ymin,xmax,ymax);
        }
/*      Erro:                   */
        return(-1);
}
```

Figura 1.28 - Implementação em C do algoritmo de Cohen-Sutherland.

```
/*   Divide um segmento em dois, descartando o que situa em um mesmo semi-plano:  */

void Divide(float *x0,float *y0,float *x1,float *y1,char code0,char code1,
                                        float xmin,float ymin,float xmax,float ymax)
{
        char code,mask=8,bit=0,j=0;
        float x,y,xa,ya,xb,yb,dx,dy;

        xa = *x0;
        ya = *y0;
        xb = *x1;
        yb = *y1;
        code = code1;
        if (code == 0) {
                code = code0;
                x  = xb;
                y  = yb;
                xb = xa;
                yb = ya;
                xa = x;
                ya = y;   }
        while (bit == 0 && j < 4)     {
                bit =(mask & code) ? 1 : 0;
                mask >>= 1;
                j++;        }
        dx = xb-xa;
        dy = yb-ya;
        switch(j)  {
                case 1:
                        if (dy != 0.0)
                                xb = xa+dx*(ymax-ya)/dy;
                        yb = ymax;
                        break;
                case 2:
                        if (dy != 0.0)
                                xb = xa+dx*(ymin-ya)/dy;
                        yb = ymin;
                        break;
                case 3:
                        if (dx != 0.0)
                                yb = ya+dy*(xmax-xa)/dx;
                        xb = xmax;
                        break;
                case 4:
                        if (dx != 0.0)
                                yb = ya+dy*(xmin-xa)/dx;
                        xb = xmin;
        }
        if (code1 == 0) {
                *x0 = xb;
                *y0 = yb;   }
        else {
                *x1 = xb;
                *y1 = yb;   }
}
```

Figura 1.28 (Continuação).

```
/*      Determina o código de um ponto de acordo com a sua posicao em relação
        a um  retângulo.  */

char Ptcode(float x,float y,float xmin,float ymin,float xmax,float ymax)
{
        char code = 0,sup = 8,inf = 4,right = 2,left = 1;

        if (y > ymax) code = sup   | code;
        if (y < ymin) code = inf   | code;
        if (x > xmax) code = right | code;
        if (x < xmin) code = left  | code;
        return(code);
}
```

Figura 1.28 (Continuação).

1.7 RECORTE DE POLÍGONOS

O algoritmo de Sutherland-Hodgman [1,2,6], descrito a seguir, é aplicável ao recorte de polígonos convexos ou côncavos contra um polígono convexo de recorte. A idéia básica do algoritmo é efetuar o recorte do polígono considerando uma única aresta por vez. Desta forma, são obtidos polígonos recortados intermediários, como mostra a Figura 1.29.

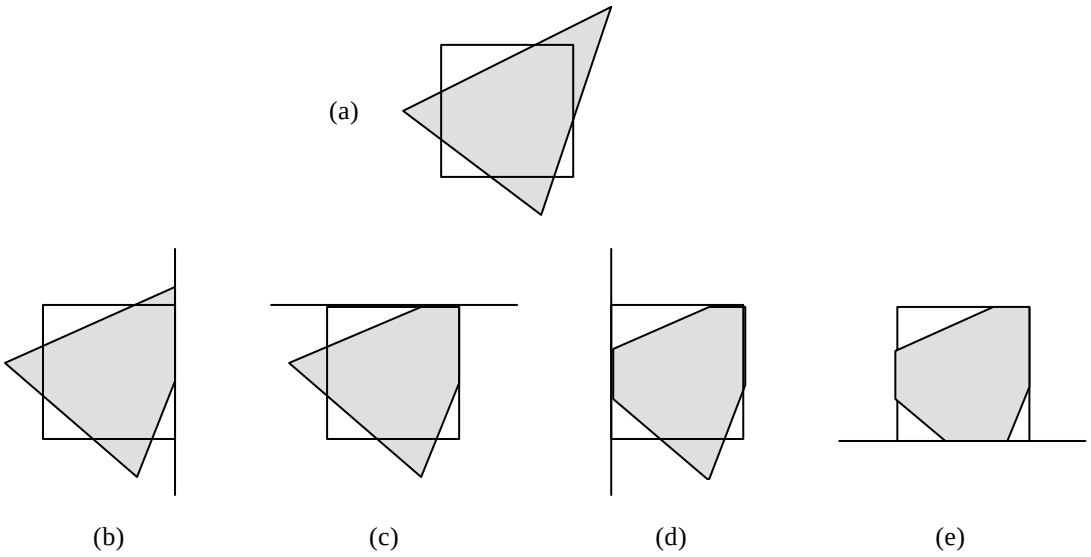


Figura 1.29 - Recorte de um polígono contra um retângulo, aresta por aresta: (a) polígono antes do recorte, (b) recortado contra a aresta direita, (c) recortado contra a aresta superior, (d) recortado contra a aresta esquerda e (e) recortado contra a aresta inferior.

A Figura 1.35 mostra uma implementação deste algoritmo. A função principal é a função *SHPolygonClip*, cujo valor de retorno é o número de vértices do polígono após o recorte. Os argumentos desta função tem o seguinte significado:

v[ ] - estrutura que armazena em sequência (horária ou anti-horária) os vértices do polígono.

n - número de vértices do polígono.

edge[2] - estrutura que contém as extremidades da aresta de recorte.

vclip[ ] - estrutura para receber os vértices do novo polígono recortado.

A aresta de recorte, de extremidades P_0 e P_1 , deve ser orientada de tal forma que o interior do polígono esteja à sua esquerda, considerando-se o sentido $P_0 \rightarrow P_1$ (Figura 1.30).

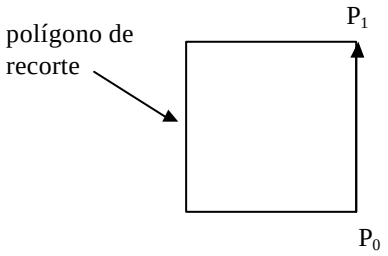


Figura 1.30 - Orientação das arestas do polígono de recorte.

O algoritmo percorre em sequência os vértices do polígono a ser recortado, começando pelo último, seguindo pelo primeiro, até retornar ao último novamente (Figura 1.31).

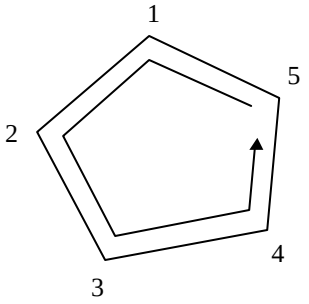


Figura 1.31 - Sequência em que o algoritmo percorre os vértices do polígono.

Para cada vértice S percorrido, o algoritmo verifica a posição da aresta formada por S e pelo vértice subsequente P em relação à aresta de recorte. A Figura 1.32 ilustra as 4 possibilidades que podem ocorrer. Dependendo do caso, dois, um ou zero vértices são armazenados na estrutura *vclip*[ ]. Na situação (a), o vértice P é adicionado ao novo polígono, em (b) a interseção das duas arestas é adicionada como novo vértice, e em (c) a interseção e P são armazenadas em *vclip*[ ]. Na hipótese(d) nenhum vértice é adicionado.

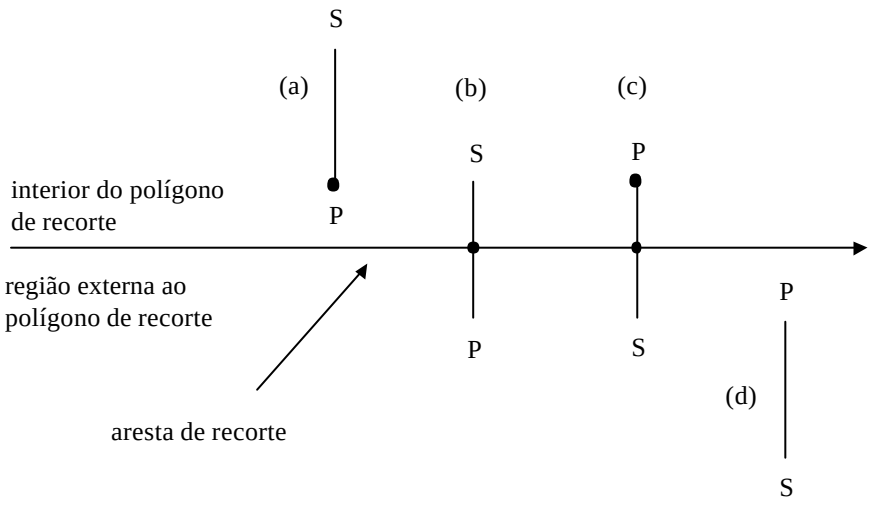


Figura 1.32 - Vértices adicionados ao polígono recortado.

A função *SHPolygonClip* chama três outras funções, que são a função *inside*, para verificar se um ponto se encontra dentro ou fora do polígono de recorte, a função *AddVertex*, que adiciona um vértice à estrutura *vclip*[ ], e a função *intersection*, que calcula a interseção da aresta de recorte com a aresta do polígono.

Verificar se um ponto P(x, y) se situa no interior do polígono equivale a saber se o ponto está à esquerda da aresta de recorte $P_0(x_0 , y_0) \rightarrow P_1(x_1 , y_1)$. Isto pode ser feito calculando-se o produto vetorial $PP_0 \times PP_1$ (Figura 1.33). A componente z deste produto é positiva, no caso em que o ponto P está à esquerda da aresta, negativa quando P está à direita, e zero quando P está localizado sobre a aresta. Considerando a aresta como sendo externa ao polígono, a condição para que o ponto P esteja contido no interior do polígono é então:

$$\left(PP_0 \times PP_1\right)_z > 0 \qquad (1.22)$$

ou simplesmente,

$$(x_1 - x) . (y_0 - y) < (x_0 - x) . (y_1 - y) \qquad (1.23)$$

Portanto, caso a expressão acima se verifique, o ponto P se situa no interior do polígono de recorte.

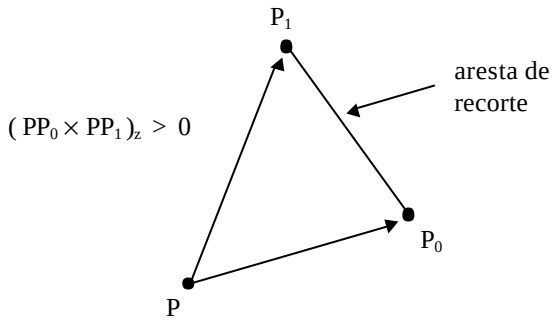


Figura 1.33 - Ponto P à esquerda da aresta de recorte.

Para recortes contra um polígono convexo qualquer, a função *SHPolygonClip* deve ser chamada seguidamente, uma vez para cada aresta do polígono de recorte.

No recorte de polígonos côncavos, algumas dificuldades ainda persitem com este algoritmo. Por exemplo, quando do recorte resultam múltiplos polígonos, estes permanecem ligados por arestas localizadas sobre o contorno do polígono de recorte, como mostra a Figura 1.34. É necessário um pós-processamento para corrigir este problema.

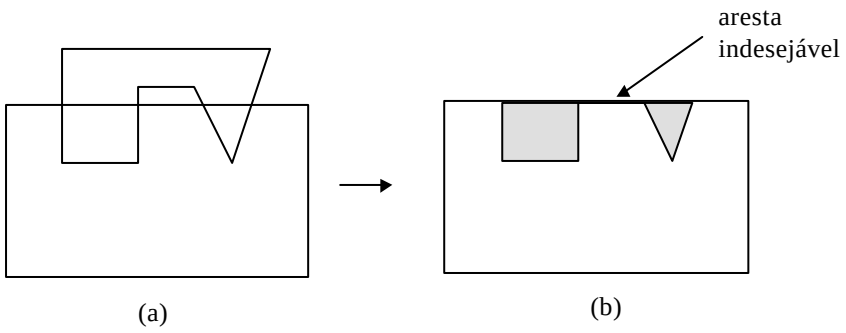


Figura 1.34 - (a) Polígono antes do recorte e (b) depois do recorte.

```
#define MAX 100
#define TRUE  1
#define FALSE 0

typedef char BOOL;
typedef struct {
        float x;
        float y;
} POINT2D;

POINT2D v[MAX],vclip[MAX],edge[2];

/*  Declaração de funções :                                                 */

        POINT2D intersection(POINT2D s,POINT2D p,POINT2D edge[]);
        int addVertex(POINT2D i,int nclip,POINT2D vclip[]);
        BOOL inside(POINT2D p,POINT2D edge[]);

/*  Recorte de um polígono contra uma aresta infinita pelo método de Sutherland-Hodgman :  */

int SHPolygonClip(POINT2D v[],int n,POINT2D edge[],POINT2D vclip[])
{
        int j,nclip;
        POINT2D s,p,i;

        nclip = 0;
        s = v[n-1];
        for (j = 0; j < n; j++) {
                p = v[j];
                if (inside(p,edge))
                        if (inside(s,edge))
                                nclip = addVertex(p,nclip,vclip);
                        else {
                                i = intersection(s,p,edge);
                                nclip = addVertex(i,nclip,vclip);
                                nclip = addVertex(p,nclip,vclip);
                        }
                else if (inside(s,edge)) {
                                i = intersection(s,p,edge);
                                nclip = addVertex(i,nclip,vclip);
                }
                s = p;
        }
        return nclip;
}
```

Figura 1.35 - Recorte de polígonos contra uma reta infinita.

```
/*   Determina se um ponto está localizado dentro ou fora do polígono de recorte :   */

BOOL inside(POINT2D p,POINT2D edge[])
{
        if (((edge[1].x-p.x)*(edge[0].y-p.y)) < ((edge[0].x-p.x)*(edge[1].y-p.y)))
                return TRUE;
        return FALSE;
}

/* Calcula a interseção de duas retas não paralelas entre si :      */

POINT2D intersection(POINT2D s,POINT2D p,POINT2D edge[])
{
        float dx,dy,m,c,a,b,t;
        POINT2D i;

        dx = edge[1].x-edge[0].x;
        if (dx != 0.0) {
                dy = edge[1].y-edge[0].y;
                m  = dy/dx;
                c = edge[0].y - m*edge[0].x;
                a = p.x - s.x;
                b = p.y - s.y;
                t = (s.x*dy + (c-s.y)*dx)/(b*dx-a*dy);
                i.x = s.x + a*t;
                i.y = s.y + b*t;   }
        else {
                dx = p.x-s.x;
                dy = p.y-s.y;
                m  = dy/dx;
                c = s.y - m*s.x;
                i.x = edge[0].x;
                i.y = m*edge[0].x + c;
        }
        return i;
}

/* Adiciona um vértice ao polígono recortado  :    */

int addVertex(POINT2D i,int nclip,POINT2D vclip[])
{
        vclip[nclip].x = i.x;
        vclip[nclip].y = i.y;
        return (nclip+1);
}
```

Figura 1.35 (Continuação)

2 TRANSFORMAÇÕES GEOMÉTRICAS

2.1 TRANSFORMAÇÕES 2D

2.1.1 Translação

Se um ponto P(x, y) sofre uma translação T(dx, dy), como mostra a Figura 2.1, sua nova posição P'(x', y') será dada por:

$$x' = x + dx \tag{2.1a}$$

$$y' = y + dy \tag{2.1b}$$

Matricialmente pode-se escrever:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix} \tag{2.2}$$

ou, de forma compacta:

$$\boxed{P' = P + T} \tag{2.3}$$

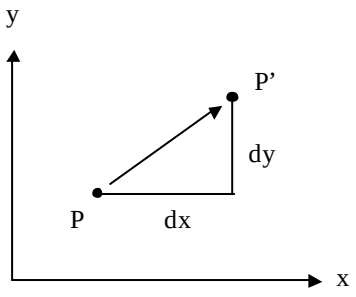


Figura 2.1 - Translação de um ponto.

2.1.2 Mudança de escala

Uma mudança de escala aplicada a um ponto isolado produz o efeito de alongar ou encurtar as componentes do vetor posição do ponto. Considerando uma mudança de escala $S(s_x, s_y)$, sendo s_x e s_y os fatores de escala nas direções x e y respectivamente, as novas coordenadas de um ponto P(x, y) são dadas por:

$$x' = s_x \cdot x \tag{2.4a}$$

$$y' = s_y \cdot y \tag{2.4b}$$

Matricialmente se tem que:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.5}$$

e de forma compacta,

$$\boxed{P' = S \cdot P} \tag{2.6}$$

2.1.3 Rotação

Aplicar uma transformação de rotação $R(\theta)$ a um ponto P(x, y) significa rotacionar de um ângulo θ o vetor posição do ponto em relação à origem (Figura 2.2). As coordenadas do ponto após a rotação são iguais a:

$$x' = x \cdot \cos\theta - y \cdot \sin\theta \tag{2.7a}$$

$$y' = x \cdot \sin\theta + y \cdot \cos\theta \tag{2.7b}$$

Matricialmente, a transformação é representada por:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.8}$$

De forma compacta,

$$P' = R \cdot P \tag{2.9}$$

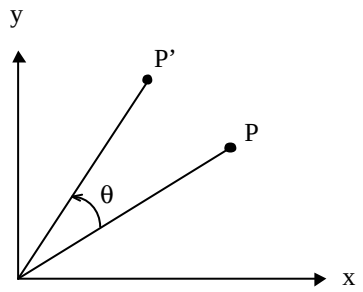


Figura 2.2 - Rotação de um ponto.

2.1.4 Coordenadas homogêneas

Em coordenadas homogêneas [7, 8], cada ponto é representado pelas coordenadas (x, y, w). Dois pontos, P = (x, y, w) e P' = (x', y', w') representam o mesmo ponto se e somente se as coordenadas de um são proporcionais às coordenadas do outro:

$$P \equiv P' \Rightarrow \begin{bmatrix} x \\ y \\ w \end{bmatrix} = t \cdot \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \tag{2.10}$$

Pelo menos uma das coordenadas homogêneas deve ser diferente de zero. O ponto (0, 0, 0) não faz parte do sistema de coordenadas homogêneas. Se $w \neq 0$, então:

$$(x, y, w) \equiv \left(\frac{x}{w}, \frac{y}{w}, 1\right) \tag{2.11}$$

Para $w \neq 0$, x/w e y/w representam as coordenadas cartesianas de (x, y, w). Os pontos para os quais w = 0 representam pontos no infinito. Os pontos (tx, ty, tw), para $t \neq 0$, representam uma reta em 3D. Consequentemente, cada ponto do sistema de coordenadas homogêneas representa uma reta em 3D cuja direção passa pela origem (Figura 2.3).

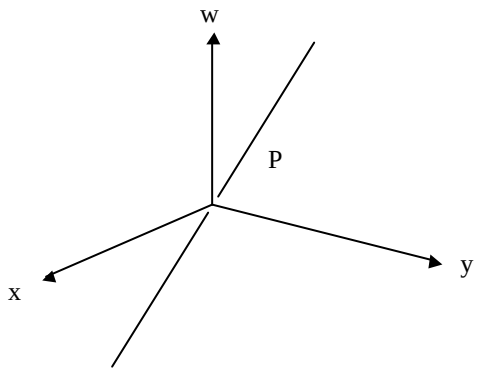


Figura 2.3 - Representação de um ponto em coordenadas homogêneas.

Os pontos em coordenadas homogêneas são normalizados dividindo-se suas coordenadas por w, obtendo-se pontos na forma (x, y, 1). O conjunto de todos os pontos normalizados forma o plano definido por w = 1, no espaço x-y-w (Figura 2.4).

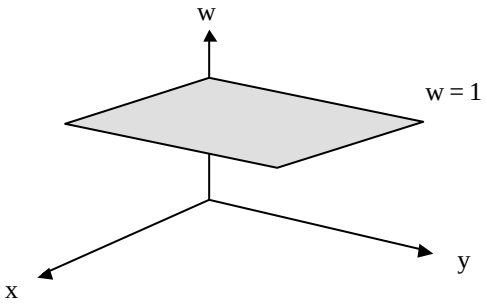


Figura 2.4 - Representação geométrica da normalização de pontos.

2.1.5 Translação usando coordenadas homogêneas

A transformação de translação, em coordenadas cartesianas, é expressa por uma operação de adição de vetores. Entretanto, através do uso de coordenadas homogêneas normalizadas é possível representar a translação por um produto de matrizes, da mesma forma que as transformações de mudança de escala e rotação:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.12}$$

De forma compacta,

$$P' = T(dx, dy) \cdot P \tag{2.13}$$

Se duas translações, $T(dx_1, dy_1)$ e $T(dx_2, dy_2)$, são aplicadas nesta ordem a um ponto P(x, y), pode-se escrever que:

$$P' = T(dx_1, dy_1) \cdot P \tag{2.14}$$

$$P'' = T(dx_2, dy_2) \cdot P' \tag{2.15}$$

Então,

$$P'' = T(dx_2, dy_2) \cdot (T(dx_1, dy_1) \cdot P) =$$

$$= T(dx_2, dy_2) \cdot T(dx_1, dy_1) \cdot P \tag{2.16}$$

$$T(dx_2, dy_2) \cdot T(dx_1, dy_1) = \begin{bmatrix} 1 & 0 & dx_2 \\ 0 & 1 & dy_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & dx_1 \\ 0 & 1 & dy_1 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & dx_1 + dx_2 \\ 0 & 1 & dy_1 + dy_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Consequentemente,

$$T(dx_2, dy_2) \cdot T(dx_1, dy_1) = T(dx_1+dx_2, dy_1+dy_2) \tag{2.18}$$

Ou seja, a aplicação de uma sequência de translações resulta em uma translação cujos deslocamentos são a soma dos deslocamentos de cada translação aplicada. A matriz resultante $T(dx_1+dx_2, dy_1+dy_2)$ denomina-se composição de $T(dx_1, dy_1)$ e $T(dx_2, dy_2)$. A ordem de aplicação das translações não altera o resultado.

2.1.6 Mudança de escala usando coordenadas homogêneas

Mudanças de escala em coordenadas homogêneas são representadas matricialmente por:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.19}$$

ou de forma compacta,

$$\boxed{P' = S(s_x, s_y) \cdot P} \tag{2.20}$$

Se duas mudanças de escala $S(s_{x1}, s_{y1})$ e $S(s_{x2}, s_{y2})$ são aplicadas consecutivamente a um ponto P(x, y) obtém-se as expressões:

$$P' = S(s_{x1}, s_{y1}) \cdot P \tag{2.21}$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P' \tag{2.22}$$

Substituindo (2.21) em (2.22) se tem que:

$$P'' = S(s_{x2}, s_{y2}) \cdot (S(s_{x1}, s_{y1}).P) = S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) \cdot P \tag{2.23}$$

$$S(s_{x2}, s_{y2}).S(s_{x1}, s_{y1}) = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_{x1}.s_{x2} & 0 & 0 \\ 0 & s_{y1}.s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

e portanto,

$$\boxed{S(s_{x2}, s_{y2}). S(s_{x1}, s_{y1}) = S(s_{x1}.s_{x2}, s_{y1}.s_{y2})} \tag{2.25}$$

O resultado acima mostra que uma composição de mudanças de escala produz uma transformação cujos fatores de escala são os produtos dos fatores de cada transformação envolvida na composição. Assim como na translação, a transformação resultante independe da ordem de aplicação das mudanças de escala.

2.1.7 Rotação usando coordenadas homogêneas

A rotação de um ponto P(x, y) é representada matricialmente por:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.26}$$

ou,

$$\boxed{P' = R(\theta) \,.\, P} \tag{2.27}$$

Aplicando uma sequência de rotações $R(\theta_1)$ e $R(\theta_2)$ a um ponto P(x, y) obtém-se:

$$P' = R(\theta_1) \,.\, P \tag{2.28}$$

$$P'' = R(\theta_2) \,.\, P' \tag{2.29}$$

Introduzindo (2.28) em (2.29):

$$P'' = R(\theta_2) \,.\, (R(\theta_1) \,.\, P) = R(\theta_2) \,.\, R(\theta_1) \,.\, P \tag{2.30}$$

$$R(\theta_2).R(\theta_1) = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} (\cos\theta_2.\cos\theta_1 - \sin\theta_2.\sin\theta_1) & (-\cos\theta_2.\sin\theta_1 - \sin\theta_2.\cos\theta_1) & 0 \\ (\sin\theta_2.\cos\theta_1 + \cos\theta_2.\sin\theta_1) & (-\sin\theta_2.\sin\theta_1 + \cos\theta_2.\cos\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

Da expressão acima conclui-se que:

$$\boxed{R(\theta_2) \,.\, R(\theta_1) \quad = \quad R(\theta_1 + \theta_2)} \tag{2.32}$$

Ou seja, a transformação resultante de uma composição de rotações pode ser obtida somando-se os ângulos das rotações envolvidas. Mais uma vez, a ordem de aplicação das rotações não altera o resultado final.

2.1.8 Transformações de corpo rígido

Uma matriz de transformação do tipo,

$$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.33}$$

onde a submatriz r_{ij} é uma matriz ortogonal e t_x e t_y são reais arbitrários, preserva ângulos e comprimentos. Transformações representadas por matrizes deste tipo denominam-se *transformações de corpo rígido*.

A submatriz de R(θ) em (2.26),

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.34}$$

é ortogonal e, portanto, a matriz de rotação representa uma transformação de corpo rígido. Uma matriz M é ortogonal se as seguintes relações se verificam:

$$M^{-1} = M^t \tag{2.35}$$

$$M^t.M \quad = \quad M\,.M^t \quad = \quad I \tag{2.36}$$

sendo I a matriz identidade.

Uma sequência arbitrária de rotações e translações tem a forma da matriz em (2.33), e portanto dá origem a uma transformação de corpo rígido.

2.1.9 Transformações afins

Sequências de rotações, translações e mudanças de escala dão origem a *transformações afins*, e têm a propriedade de preservar paralelismo de retas. No entanto, não preservam ângulos e comprimentos (Figura 2.5).

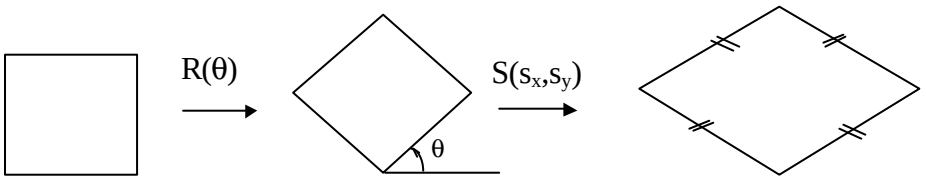


Figura 2.5 - Transformação *afim*.

2.1.10 Composição de transformações

A composição de transformações tem por objetivo ganhar eficiência com a aplicação de uma única matriz composta, ao invés de uma série de transformações. Considere, por exemplo, o problema de rotacionar o objeto da Figura 2.6 em torno de um ponto $P_1(x_1, y_1)$. O efeito desejado pode ser obtido através da seguinte sequência de transformações:

1. Translação do objeto tal que $P_1 \equiv 0$ (Figura 2.7a).
2. Rotação (Figura 2.7b).
3. Translação do objeto tal que P_1 retorne à posição original (Figura 2.7c).

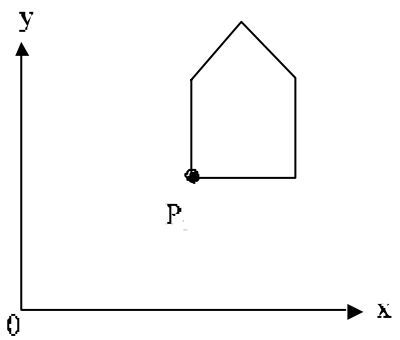
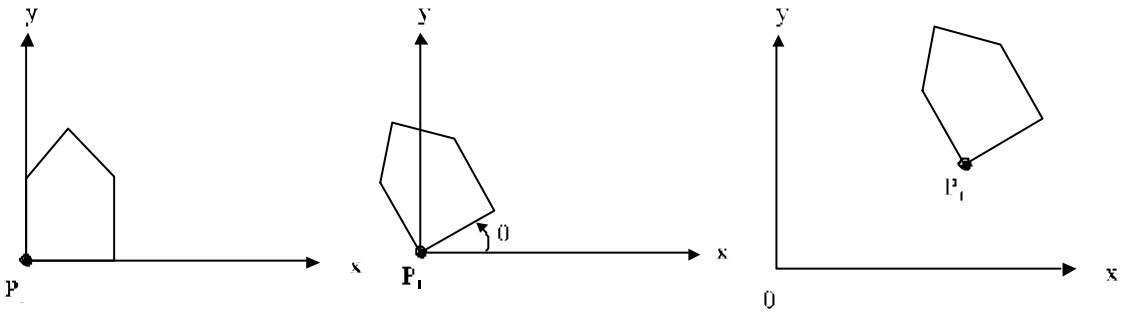


Figura 2.6

A transformação composta correspondente à sequência de transformações é dada por:

$$T(x_1,y_1)\,.\,R(\theta)\,.T(-x_1,-y_1) = \begin{bmatrix}1 & 0 & x_1\\ 0 & 1 & y_1\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & -x_1\\ 0 & 1 & -y_1\\ 0 & 0 & 1\end{bmatrix} =$$

$$= \begin{bmatrix}\cos\theta & -\sin\theta & (x_1(1-\cos\theta)+y_1\sin\theta)\\ \sin\theta & \cos\theta & (y_1(1-\cos\theta)-x_1\sin\theta)\\ 0 & 0 & 1\end{bmatrix} \qquad (2.37)$$



(a) (b) (c)

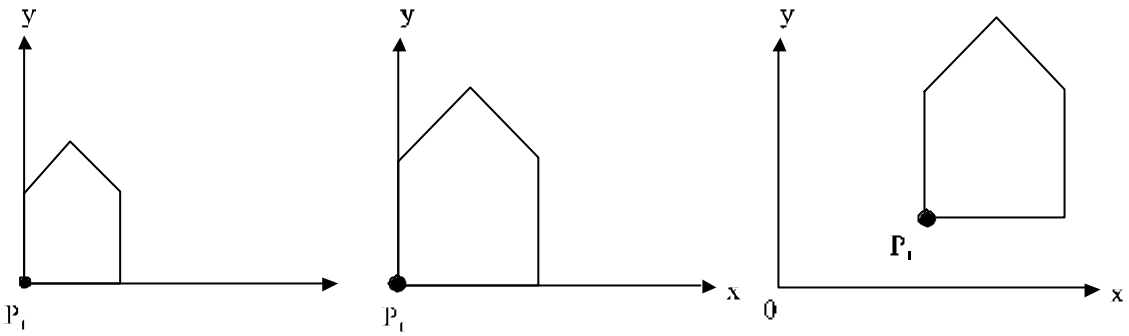
Figura 2.7 - Sequência de transformações para rotacionar o objeto em torno de um ponto.

Da mesma forma, a seguinte sequência de transformaçòes pode ser utilizada para mudar a escala do mesmo objeto em relação a um ponto $P_1(x_1,y_1)$:

1. Translação tal que $P_1 \equiv 0$ (Figura 2.8a).
2. Mudança de escala (Figura 2.8b).
3. Translação tal que P_1 retorne à posição original (Figura 2.8c).

Neste caso, a transformação resultante é dada por:

$$T(x_1,y_1)\,.S(s_x,s_y)\,.T(-x_1,-y_1) = \begin{bmatrix}s_x & 0 & x_1.(1-s_x)\\ 0 & s_y & y_1.(1-s_y)\\ 0 & 0 & 1\end{bmatrix} \qquad (2.38)$$



(a) (b) (c)

Figura 2.8 - Sequência de transformações para mudança de escala do objeto em relação a um ponto.

Outro exemplo de composição de transformações é o problema da mudança de escala e rotação de um objeto em relação a um ponto $P_1(x_1,y_1)$ e reposicionamento do objeto através da translação $P_1 \rightarrow P_2(x_2, y_2)$ (Figura 2.9). Neste caso, a seguinte sequência pode ser empregada:

1. Translação $P_1 \rightarrow 0$ (Figura 2.10a).
2. Mudança de escala (Figura 2.10b).
3. Rotação (Figura 2.10c).
4. Translação $P_1 \rightarrow P_2$ (Figura 2.10d).

A transformação composta é:

$$T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) \qquad (2.39)$$

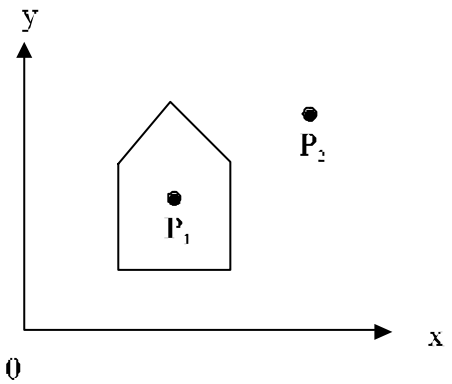


Figura 2.9 - Mudança de escala e rotação em relação a P_1 e translação $P_1 \rightarrow P_2$.

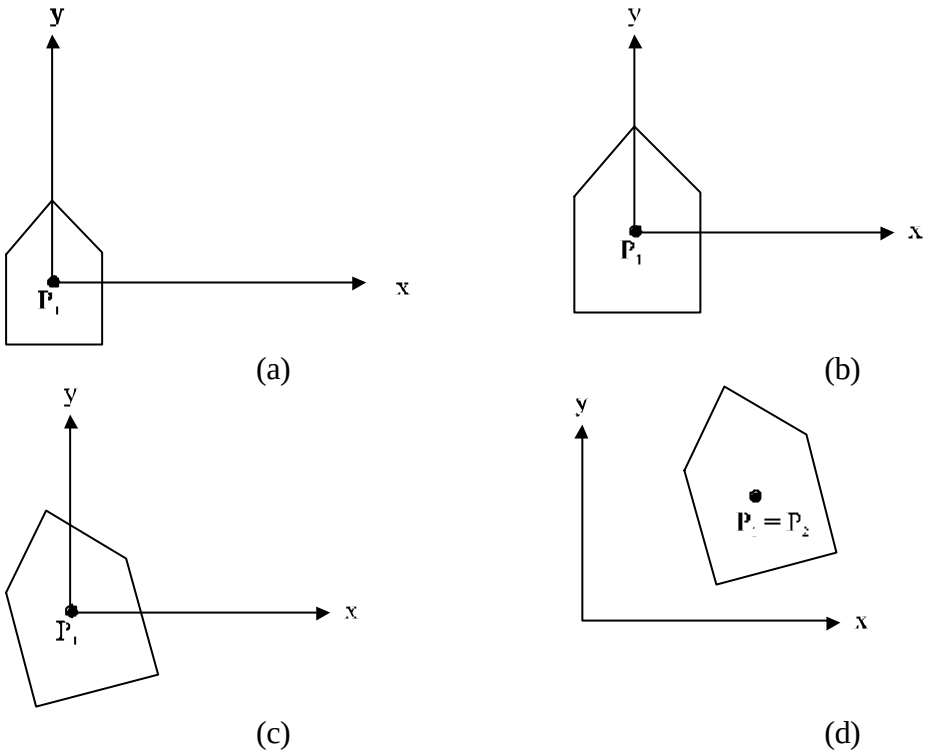


Figura 2.10 - Sequência de transformações para mudança de escala e rotação em relação a um ponto e reposicionamento do objeto.

Uma transformação composta incluindo transformações de rotação, mudança de escala e translação produz uma matriz M na forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.40}$$

A submatriz r_{ij} é uma composição de rotações e mudanças de escala, enquanto t_x e t_y são translações compostas.

O produto da matriz M por um ponto P(x, y) implica em nove multiplicações e seis adições:

$$x' = r_{11} . x + r_{12} . y + t_x .1 \tag{2.41a}$$

$$y' = r_{21} . x + r_{22} . y + t_y .1 \tag{2.41b}$$

$$1 = 0 . x + 0 . y + 1 . 1 \tag{2.41c}$$

A estrutura de M permite, no entanto, a seguinte simplificação,

$$x' = r_{11} . x + r_{12} . y + t_x \quad (2.42a)$$

$$y' = r_{21} . x + r_{22} . y + t_y \quad (2.42b)$$

reduzindo o número de operações para quatro multiplicações e quatro adições, o que representa uma significativa redução, considerando-se que milhares de pontos podem ser transformados.

2.1.11 Coordenadas de mundo e coordenadas de dispositivo

Coordenadas de mundo são aquelas com as quais os objetos são inicialmente representados. O mapeamento de uma janela do sistema de coordenadas de mundo (Figura 2.11) em uma região do dispositivo de saída gráfica (Figura 2.12) pode ser obtido através da seguinte sequência de transformacões:

1. Translação da janela para a origem do sistema de coordenadas de mundo (Figura 2.13a).

2. Mudança de escala para enquadrar a janela de WCS na região de visualização do dispositivo (Figura 2.13b).

3. Translação $T(u_{min} ,v_{min})$ no sistema de coordenadas de dispositivo (Figura 2.13c).

A transformação resultante é:

$$M_{wv} = T(u_{min} ,v_{min}) . S(s_x , s_y) .T(-x_{min} , -y_{min}) \quad (2.43)$$

sendo s_x e s_y dados por:

$$s_x = \frac{u_{max} - u_{min}}{x_{max} - x_{min}} \quad (2.44a)$$

$$s_y = \frac{v_{max} - v_{min}}{y_{max} - y_{min}} \quad (2.44b)$$

Efetuando os produtos matriciais em (2.43) obtém-se:

$$M_{wv} = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} s_x & 0 & -x_{min}.s_x + u_{min} \\ 0 & s_y & -y_{min}.s_y + v_{min} \\ 0 & 0 & 1 \end{bmatrix} \qquad (2.45)$$

As coordenadas (u, v) no dispositivo de saída de um ponto P(x, y) são então dadas por:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{wv} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} (x - x_{min}).s_x + u_{min} \\ (y - y_{min}).s_y + v_{min} \\ 1 \end{bmatrix} \qquad (2.46)$$

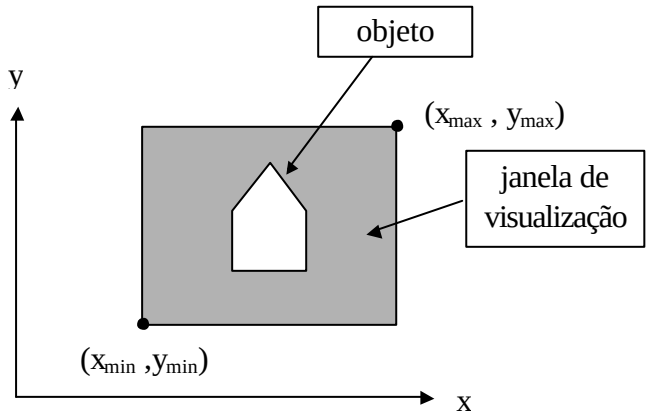


Figura 2.11 - representação de um objeto em coordenadas de mundo (world coordinates system, WCS).

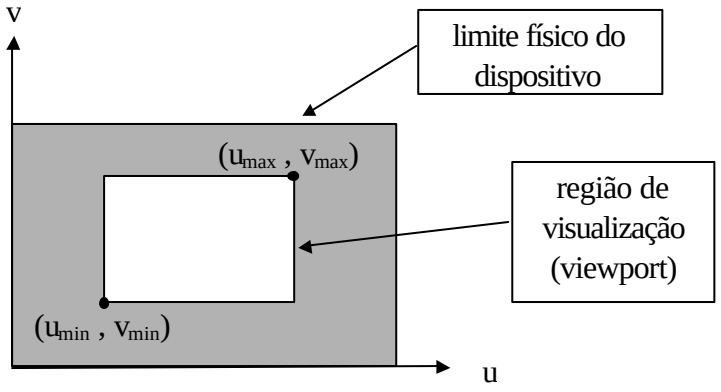


Figura 2.12 - Sistema de coordenadas de dispositivo (device coordinates system, DCS).

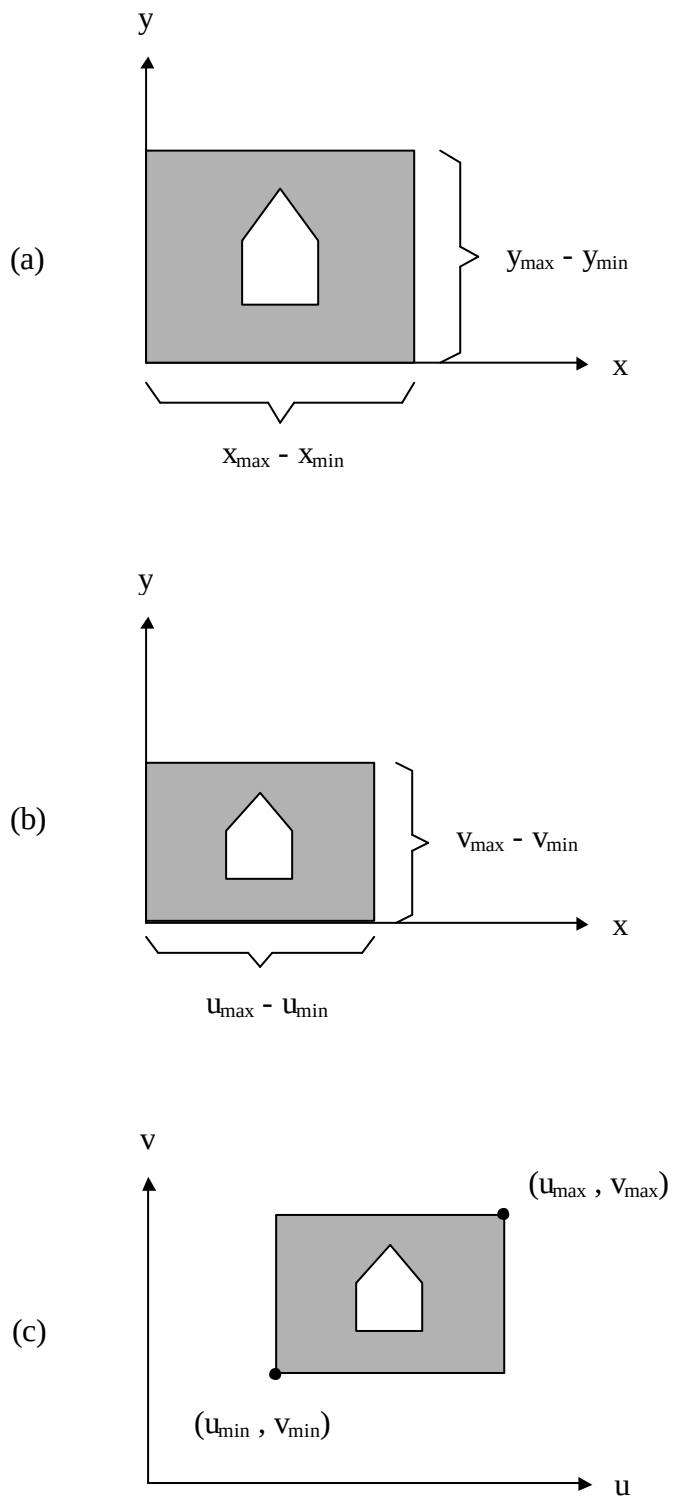


Figura 2.13 - Transformações necessárias para representação de um objeto no dispositivo de saída.

2.2 TRANSFORMAÇÕES 3D

Assim como as transformações 2D podem ser representadas por matrizes 3x3 usando coordenadas homogêneas, as transformações 3D podem ser representadas por matrizes 4x4. As coordenadas homogêneas de pontos do espaço são dadas por:

$$(x, y, z, w) \tag{2.47}$$

Assim como em duas dimensões, os pontos (x, y, z, w), para $w \neq 0$, são representados na forma:

$$(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}, 1) \tag{2.48}$$

Cada ponto do espaço é representado por uma reta em 4D cuja direção passa pela origem, e a normalização de todos os pontos em coordenadas homogêneas dá origem a um subespaço 3D do espaço 4D definido por w = 1.

2.2.1 Transformacões básicas

Em três dimensões, as transformações de translação T(dx,dy,dz) e mudança de escala $S(s_x, s_y, s_z)$ são representadas pelas matrizes:

$$T(dx, dy, dz) = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.49}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.50}$$

Considerando os sentidos da Figura 2.14, as rotações R_x, R_y e R_z , em torno respectivamente dos eixos x, y, z, são dadas pelas seguintes transformações:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.51}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.52}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.53}$$

Como pode-se verificar, as matrizes R_x, R_y e R_z são ortogonais.

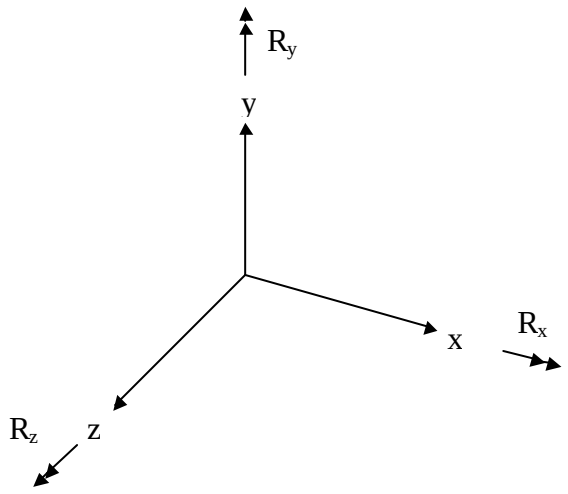


Figura 2.14 - Sentidos positivos das rotações em torno dos eixos x, y, z.

2.2.2 Operações inversas

Se um ponto sofre uma determinada transformação $\mathcal{L}$, a operação inversa $\mathcal{L}^{-1}$ desta transformação retorna o ponto à sua posição original, ou seja:

$$\mathcal{L} \cdot \mathcal{L}^{-1} = I \tag{2.54}$$

As operações inversas das transformações de translação, mudanca de escala e rotação são dadas pelas expressões:

$$T^{-1}(dx, dy, dz) = \begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.55}$$

$$S^{-1} = S(s_x, s_y, s_z) = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.56}$$

$$R_x^{-1}(\theta) = R_x^t(\theta) = R_x(-\theta) \tag{2.57a}$$

$$R_y^{-1}(\theta) = R_y^t(\theta) = R_y(-\theta) \tag{2.57b}$$

$$R_z^{-1}(\theta) = R_z^t(\theta) = R_z(-\theta) \tag{2.57c}$$

2.2.3 Composição de transformações

A matriz resultante de uma sequência arbitrária de transformações de translação, mudança de escala e rotação tem a forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.58}$$

onde a submatriz r_{ij} refere-se à composição de rotações e mudanças de escala, e o vetor coluna $(t_x, t_y, t_z, 1)^t$ representa uma composição de translações.

Como exemplo de composição de transformações, seja o problema de transformar os segmentos de reta P_1P_2 e P_1P_3 de sua posição original (Figura 2.15a) para a posição da Figura 2.15b. Para obter o efeito desejado, as seguintes transformações devem ser empregadas:

1. Translação $P_1 \rightarrow 0$.
2. Rotação em torno do eixo y, tal que o segmento P_1P_2 que fique contido no plano yz.
3. Rotação em torno do eixo x, tal que o segmento P_1P_2 fique contido no eixo z.
4. Rotação em torno do eixo z, tal que o segmento P_1P_3 fique contido no plano yz.

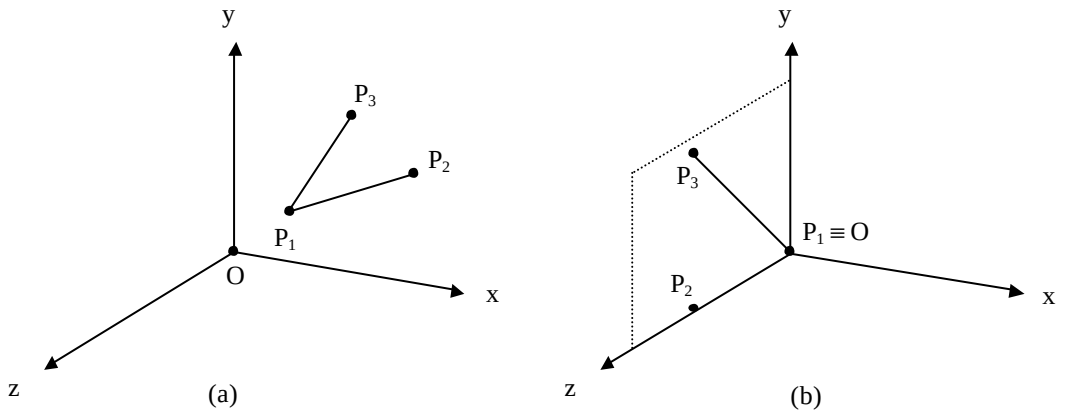


Figura 2.15 - (a) Posição original. (b) Posição final.

Aplicando a translação $T(-x_1, -y_1, -z_1)$ o ponto P_1 translada-se para a origem (Figura 2.16), obtendo-se as novas coordenadas:

$$P_1' = T(-x_1,-y_1,-z_1).P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.59}$$

$$P_2' = T(-x_1,-y_1,-z_1).P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} \tag{2.60}$$

$$P_3' = T(-x_1,-y_1,-z_1).P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} \tag{2.61}$$

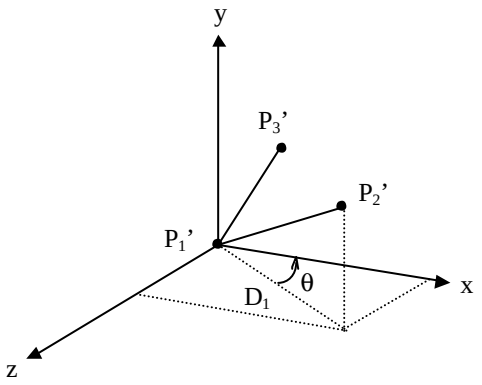


Figura 2.16 - Posição dos segmentos após translação de P_1 para a origem.

Observando a Figura 2.16 verifica-se que, para o segmento $P_1'P_2'$ ficar contido no plano yz, uma rotação $R_y(-(90-\theta))$ deve ser aplicada. Tendo em conta que:

$$\cos(-(90-\theta)) = \cos(\theta - 90) = \sin\theta = \frac{z_2'}{D_1} = \frac{z_2 - z_1}{D_1} \tag{2.62}$$

$$\sin(\theta - 90) = -\cos\theta = \frac{-x_2'}{D_1} = -\frac{x_2 - x_1}{D_1} \tag{2.63}$$

$$D_1 = \sqrt{x_2'^2 + z_2'^2} = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2} \tag{2.64}$$

as coordenadas dos pontos transformados P_1'', P_2'' e P_3'' são iguais a:

$$P_1'' = R_y(\theta\text{-}90) \cdot P_1' \tag{2.65}$$

$$P_2'' = R_y(\theta\text{-}90) \cdot P_2' \tag{2.66}$$

$$P_3'' = R_y(\theta\text{-}90) \cdot P_3' \tag{2.67}$$

sendo a matriz R_y dada por:

$$R_y(\theta - 90) = \begin{bmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_2 - x_1}{D_1} & 0 & \frac{z_2 - z_1}{D_1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.68}$$

Efetuando os produtos em (2.65), (2.66) e (2.67) obtém-se:

$$P_1'' = \begin{bmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_2 - x_1}{D_1} & 0 & \frac{z_2 - z_1}{D_1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = P_1' \tag{2.69}$$

$$P_2'' = \begin{bmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_2 - x_1}{D_1} & 0 & \frac{z_2 - z_1}{D_1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ y_2 - y_1 \\ D_1 \\ 1 \end{bmatrix} \tag{2.70}$$

$$P_3'' = \begin{bmatrix} \frac{z_2 - z_1}{D_1} & 0 & -\frac{x_2 - x_1}{D_1} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{x_2 - x_1}{D_1} & 0 & \frac{z_2 - z_1}{D_1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_3'' \\ y_3'' \\ z_3'' \\ 1 \end{bmatrix} \tag{2.71}$$

Para tornar o segmento $P_1''P_2''$ coincidente com o eixo z, deve-se aplicar uma rotação $R_x(\phi)$ em torno do eixo x (Figura 2.17). Considerando as relações:

$$\cos\phi = \frac{z_2''}{D_2} = \frac{D_1}{D_2} \tag{2.72}$$

$$\sin\phi = \frac{y_2''}{D_2} = \frac{y_2 - y_1}{D_2} \tag{2.73}$$

$$D_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.74}$$

a matriz $R_x(\phi)$ será dada por:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{D_1}{D_2} & \frac{y_1 - y_2}{D_2} & 0 \\ 0 & \frac{y_2 - y_1}{D_2} & \frac{D_1}{D_2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.75}$$

Aplicando esta rotação aos pontos P_1'', P_2'' e P_3'' obtém-se:

$$P_1''' = R_x(\phi).P_1'' = \begin{bmatrix} x_1''' \\ y_1''' \\ z_1''' \\ 1 \end{bmatrix} \tag{2.76}$$

$$P_2''' = R_x(\phi).P_2'' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{D_1}{D_2} & \frac{y_1 - y_2}{D_2} & 0 \\ 0 & \frac{y_2 - y_1}{D_2} & \frac{D_1}{D_2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ y_2 - y_1 \\ D_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ D_2 \\ 1 \end{bmatrix} \tag{2.77}$$

$$P_3''' = R_x(\phi).P_3'' = \begin{bmatrix} x_3''' \\ y_3''' \\ z_3''' \\ 1 \end{bmatrix} \tag{2.78}$$

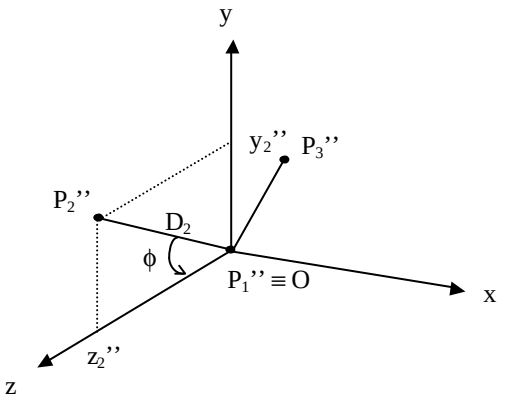


Figura 2.17 - Posição dos segmentos após rotação R_y.

Finalmente, para posicionar o segmento $P_1'''P_3'''$ no plano yz, deve-se aplicar uma rotação $R_z(\alpha)$ em torno de z (Figura 2.18). Os elementos desta matriz são dados por:

$$\cos\alpha = \frac{y_3'''}{D_3} \tag{2.79}$$

$$\sin\alpha = \frac{x_3'''}{D_3} \tag{2.80}$$

$$D_3 = \sqrt{x_3'''^2 + y_3'''^2} \tag{2.81}$$

A transformação composta resultante será então igual a:

$$M = R_z(\alpha) \,.\, R_x(\phi) \,.\, R_y(\theta\text{-}90) \,.\, T(-x_1, -y_1, -z_1) \tag{2.82}$$

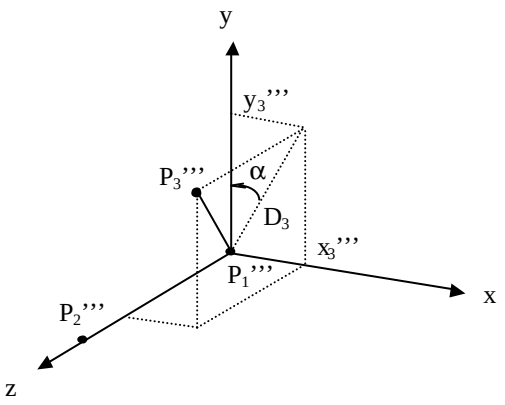


Figura 2.18 - Posição dos segmentos após rotação R_x.

2.2.4 Mudança de sistema de coordenadas

Seja $M_{j\leftarrow i}$ a transformação que converte a representação de um ponto em um sistema de coordenadas i em sua representação no sistema j:

$$P^{(j)} = M_{j\leftarrow i} \cdot P^{(i)} \tag{2.83}$$

A matriz que transforma as coordenadas de um sistema i em coordenadas de um sistema j (Figura 2.19) é dada pela seguinte expressão:

$$\begin{bmatrix} x^{(j)} \\ y^{(j)} \\ z^{(j)} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos x^{(j)}x^{(i)} & \cos x^{(j)}y^{(i)} & \cos x^{(j)}z^{(i)} & 0 \\ \cos y^{(j)}x^{(i)} & \cos y^{(j)}y^{(i)} & \cos y^{(j)}z^{(i)} & 0 \\ \cos z^{(j)}x^{(i)} & \cos z^{(j)}y^{(i)} & \cos z^{(j)}z^{(i)} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{(i)} \\ y^{(i)} \\ z^{(i)} \\ 1 \end{bmatrix} \tag{2.84}$$

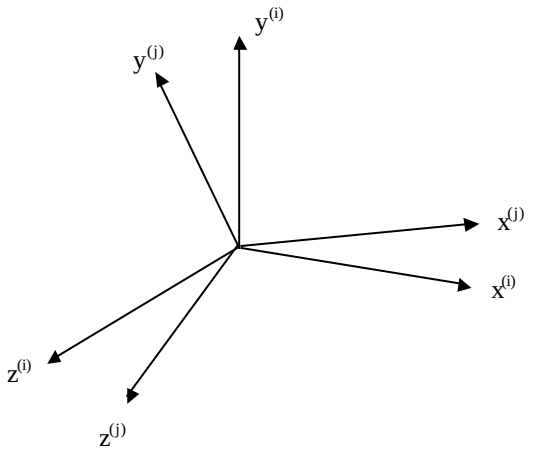


Figura 2.19 - Mudança de sistema de eixos.

A transformação de um ponto em um sistema de coordenadas é a matriz inversa da transformação correspondente a uma mudança de sistema de coordenadas.

Exemplo 1

Em duas dimensões, a matriz que transforma um sistema de coordenadas i em outro sistema j rotacionado de um ângulo θ (Figura 2.20) é dada por:

$$R_{j\leftarrow i}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.85}$$

Então, a transformação correspondente a uma rotação θ de um ponto (Figura 2.21) é dada por:

$$R(\theta) = R^{-1}_{j\leftarrow i}(\theta) = R^{t}_{j\leftarrow i}(\theta) = R_{j\leftarrow i}(-\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.86}$$

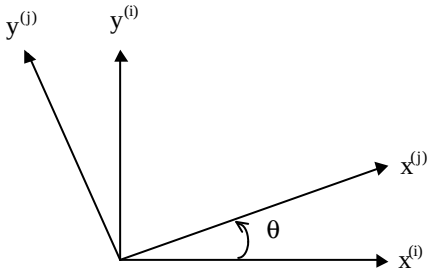


Figura 2.20 - Rotação θ do sistema de eixos.

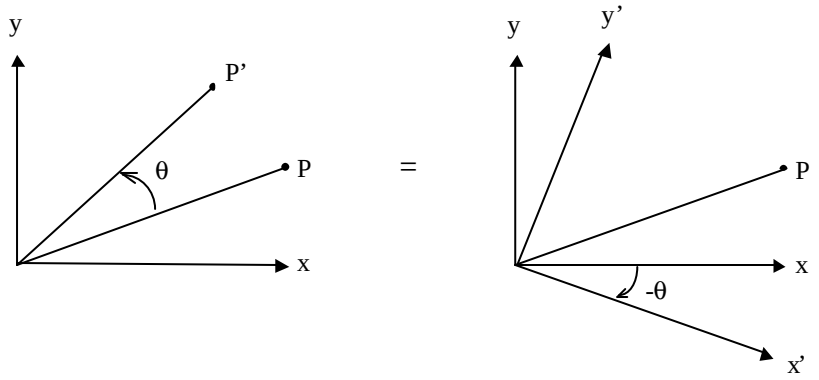


Figura 2.21 - Rotação θ de um ponto.

Exemplo 2

O exemplo de transformação composta do item 2.2.3 pode ser resolvido mais facilmente através de duas mudanças de sistema, uma translação e uma rotação, como mostram as Figuras 2.22a e b.

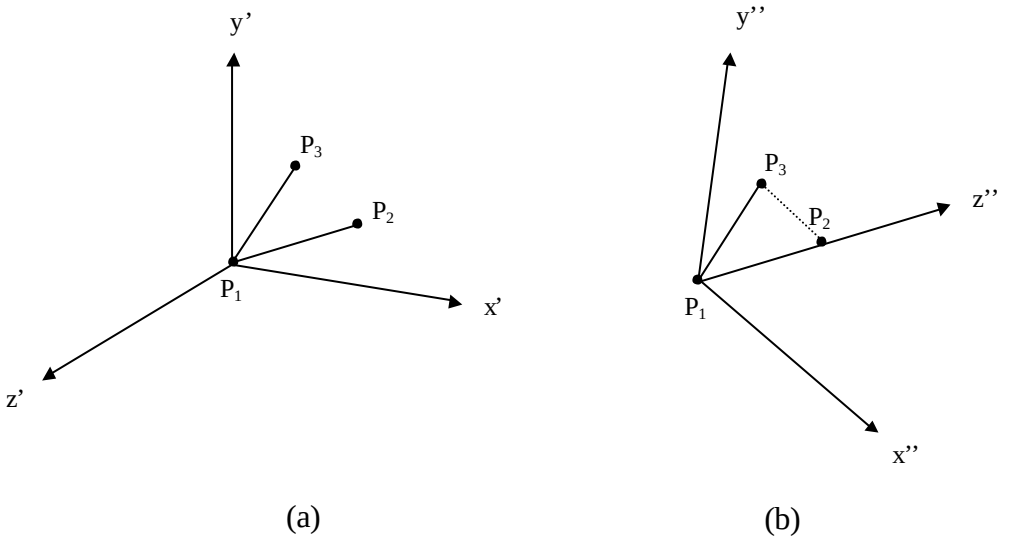


Figura 2.22 - (a) Translação e (b) rotação do sistema de eixos.

A primeira mudança de eixos é uma translação, posicionando o ponto P_1 na origem:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T(-x_1,-y_1,-z_1).\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.87}$$

A outra mudança é uma rotação de eixos, dada pela relação:

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos x''x' & \cos x''y' & \cos x''z' & 0 \\ \cos y''x' & \cos y''y' & \cos y''z' & 0 \\ \cos z''x' & \cos z''y' & \cos z''z' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \tag{2.88}$$

Da Figura 2.22 conclui-se que os cossenos diretores do eixo z'' são:

$$\cos z''x' = \frac{x'_2}{|P_1P_2|} \tag{2.89a}$$

$$\cos z''y' = \frac{y'_2}{|P_1P_2|} \tag{2.89b}$$

$$\cos z''z' = \frac{z'_2}{|P_1P_2|} \tag{2.89c}$$

e portanto, o unitário na direção z'' é:

$$w = \begin{bmatrix} \cos z''x' \\ \cos z''y' \\ \cos z''z' \end{bmatrix} \tag{2.90}$$

Como o eixo x'' é perpendicular ao plano formado pelos dois segmentos, o vetor unitário na direção x'' é obtido de acordo com a expressão:

$$u = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|} = \begin{bmatrix} u_{x'} \\ u_{y'} \\ u_{z'} \end{bmatrix} \tag{2.91}$$

E, finalmente, o vetor unitário na direção y'' pode ser obtido em função de x'' e z'':

$$v = \frac{w \times u}{|w \times u|} = \begin{bmatrix} v_{x'} \\ v_{y'} \\ v_{z'} \end{bmatrix} \tag{2.92}$$

Substituindo (2.90), (2.91) e (2.92) em (2.88) obtém-se os eixos rotacionados:

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} = \begin{bmatrix} u_{x'} & u_{y'} & u_{z'} & 0 \\ v_{x'} & v_{y'} & v_{z'} & 0 \\ \cos z''x' & \cos z''y' & \cos z''z' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \tag{2.93}$$

3 VISUALIZAÇÃO 3D

O primeiro passo para a visualização de um objeto tridimensional em um dispositivo gráfico é a definição de um modelo de visualização. O modelo de visualização deve ser construído de tal forma que a imagem bidimensional do objeto, representada no dispositivo de saída, reproduza o mais fielmente possível suas verdadeiras formas.

O modelo de visualização descrito no item 3.3 é utilizado pela maioria dos pacotes gráficos comerciais, e segue o padrão PHIGS. Em linhas gerais, este modelo consiste, em primeiro lugar, no estabelecimento de um sistema de coordenadas (coordenadas de mundo) utilizado para descrever o objeto espacialmente. Em seguida, deve-se especificar um ponto de observação (centro de projeção), limitando o campo de visão do observador a uma região finita do espaço, denominada volume de visualização. O volume de visualização é definido por um plano de projeção, pelos planos anterior e posterior de recorte, e pelo tipo de projeção escolhido (perspectiva ou paralela), como será visto mais adiante. As partes do objeto que se situam no interior do volume de visualização são projetadas no plano de projeção e daí mapeadas para uma região do dispositivo de saída.

A definição do modelo de visualização resulta, portanto, na caracterização de uma série de transformações geométricas, necessárias para transformar um objeto tridimensional em uma imagem bidimensional descrita pelo sistema de coordenadas do dispositivo. Pelo fato de envolver apenas transformações geométricas, o modelo de visualização é aplicável a pontos, e independe de como o objeto é modelado geometricamente. Em outras palavras, o modelo de visualização não depende dos elementos ou primitivas, tais como segmentos de reta e faces planas, utilizadas para compor o objeto.

As demais etapas do processo de visualização são as seguintes:

- Identificação das primitivas, ou partes destas, que se situam no interior do volume de visualização.
- Determinação da visibilidade das primitivas, sob o ponto de vista do observador.
- Conversão das primitivas.

Efeitos de uma cena real, tais como iluminação e transparência, também podem ser incorporados ao processo de visualização [1, 2].

3.1 PROJEÇÕES

Em geral, projeções transformam pontos de um sistema de coordenadas n-dimensional em pontos representados em um sistema de dimensão inferior. Neste item, serão estudadas as projeções de um sistema 3D em um sistema 2D.

As projeções podem ser classificadas em dois grupos principais, perspectiva e paralela, como mostra a Figura 3.1. Nas projeções perspectivas, as linhas de projeção convergem para um ponto, denominado centro de projeção, enquanto nas projeções paralelas, as linhas de projeção são paralelas entre si.

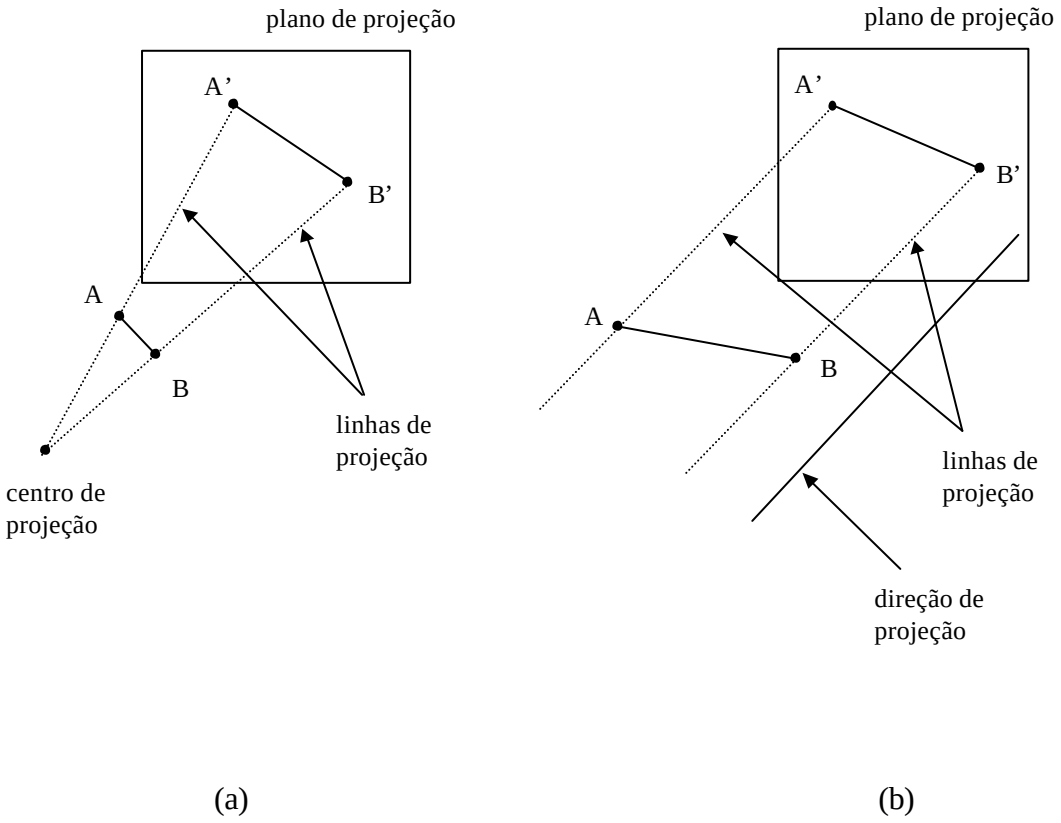
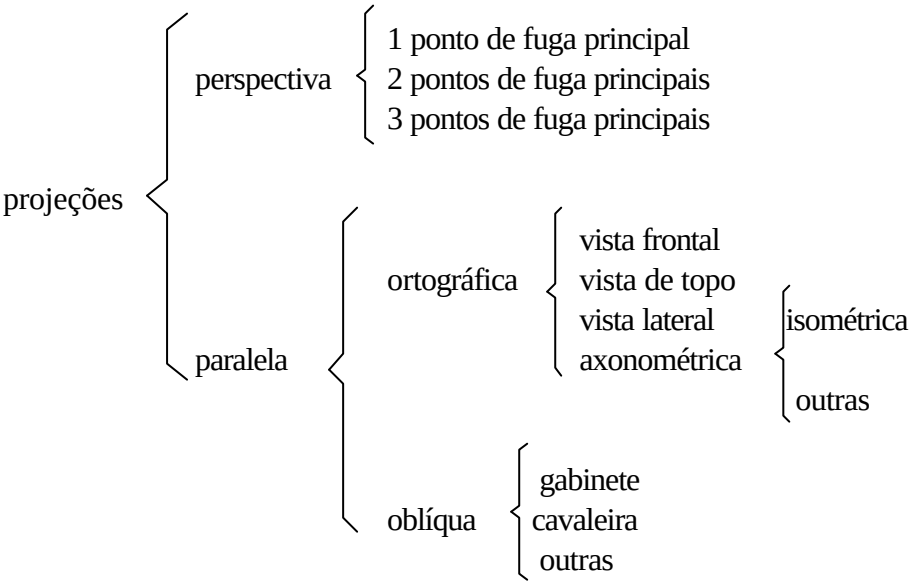


Figura 3.1 - (a) Projeção perspectiva. (b) Projeção paralela.

Os principais tipos de projeções perspectiva e paralela são os seguintes:



3.1.1 Projeção perspectiva

Em uma projeção perspectiva, um conjunto de retas paralelas não paralelas ao plano de projeção converge para um ponto chamado *ponto de fuga*. Como retas paralelas só se encontram no infinito, um ponto de fuga pode ser interpretado como a projeção de um ponto do infinito (Figura 3.2).

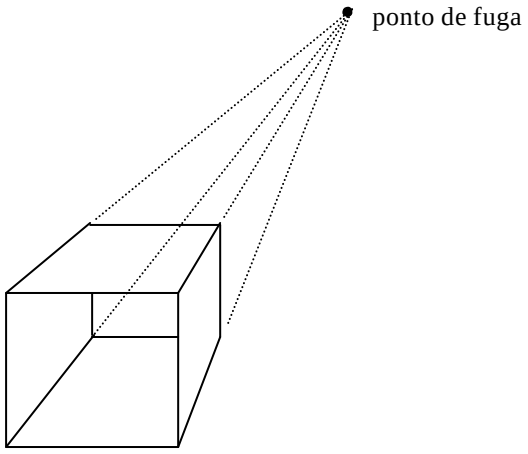


Figura 3.2 - Ponto de fuga.

Ângulos são preservados somente nas faces do objeto paralelas ao plano de projeção. Se um conjunto de retas é paralelo a um dos eixos principais do sistema, o ponto de fuga é dito *ponto de fuga principal* (Figura 3.3).

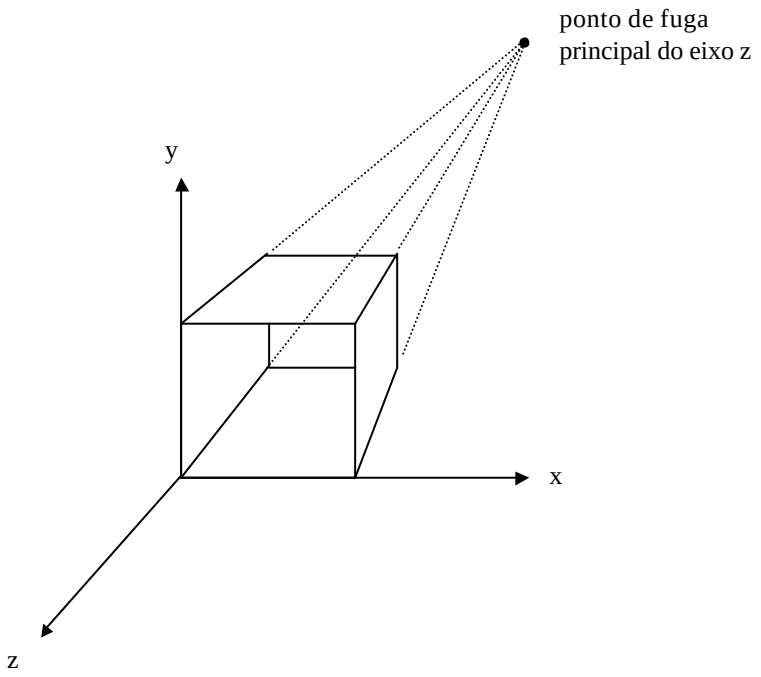


Figura 3.3 - Ponto de fuga principal.

Existem no máximo 3 pontos de fuga principais em uma projeção perspectiva, correspondendo ao número de eixos principais interceptados pelo plano de projeção. Por exemplo, se o plano de projeção intercepta apenas o eixo z (paralelo ao plano xy), somente o eixo z conterá um ponto de fuga principal, porque as linhas paralelas ao eixo x ou y são também paralelas ao plano de projeção. As projeções perspectiva, portanto, classificam-se de acordo com o número de pontos de fuga principais: um, dois ou três pontos de fuga principais.

3.1.2 Projeção paralela

Assim como as perspectivas, as projeções paralelas preservam ângulos somente nas faces do objeto paralelas ao plano de projeção. As projeções paralelas dividem-se em dois grupos, dependendo da relação entre a direção de projeção e a normal ao plano de projeção. Nas projeções paralelas *ortográficas*, a normal ao plano de projeção e a direção de projeção são paralelas. Nas projeções paralelas *oblíquas*, a normal ao plano de projeção é oblíqua à direção de projeção.

3.1.2.1 Projeções paralelas ortográficas

As projeções ortográficas caracterizam-se pelo fato de a normal n ao plano de projeção ser paralela à direção de projeção, ou seja, as linhas de projeção são perpendiculares ao plano de projeção (Figura 3.4).

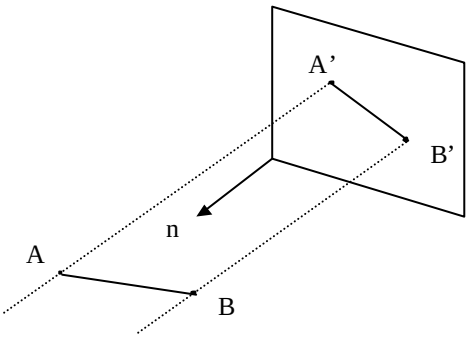


Figura 3.4 - Projeção paralela ortográfica.

As principais projeções ortográficas são a *vista frontal*, *vista de topo* e *vista lateral*. Estas três vistas representam projeções paralelas ortográficas nas quais o plano de projeção é perpendicular a um dos três eixos do sistema de coordenadas (Figura 3.5).

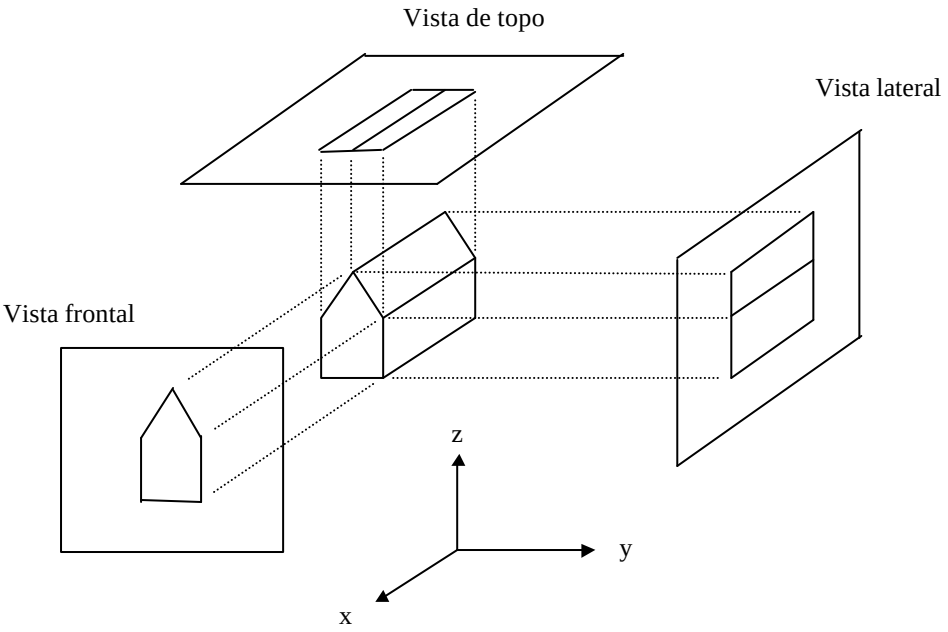


Figura 3.5 - Projeções ortográficas: vistas frontal, lateral e de topo.

As projeções paralelas ortográficas *axonométricas* usam planos de projeção não perpendiculares aos eixos principais. A projeção ortográfica *isométrica* é um caso particular da axonométrica, na qual a normal ao plano de projeção forma ângulos iguais com os três eixos principais do sistema de coordenadas.

3.1.2.2 Projeções paralelas oblíquas

As projeções paralelas oblíquas diferem das ortográficas pelo fato de a normal ao plano de projeção não ser paralela à direção de projeção. Os principais tipos de projeções oblíquas são as projeções *cavaleira* e de *gabinete.*

Nas projeções cavaleiras, a direção de projeção forma um ângulo de 45º com a normal ao plano de projeção, de modo que as projeções segmentos de reta perpendiculares ao plano de projeção mantenham seus comprimentos verdadeiros (Figura 3.6).

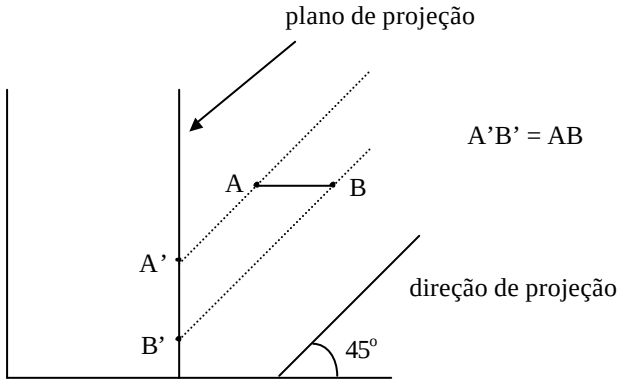


Figura 3.6 - Projeção cavaleira.

Projeções de gabinete são aquelas em que as projeções de segmentos de reta perpendiculares ao plano de projeção apresentam um comprimento igual à metade do comprimento verdadeiro (Figura 3.7).

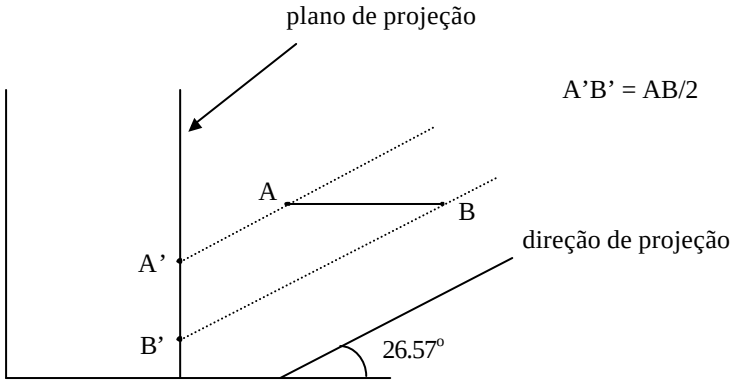


Figura 3.7 - Projeção de gabinete.

3.2 MATEMÁTICA DAS PROJEÇÕES

3.2.1 Projeção perspectiva

Seja a projeção perspectiva em um plano de projeção que contém o ponto $R_0(x_0, y_0, z_0)$ e cuja normal é dada pelo vetor unitário n, de cossenos diretores (n_x, n_y, n_z). O centro de projeção está localizado na origem do sistema de coordenadas (Figura 3.8). Em coordenadas homogêneas, a projeção P'(x', y', z', w') de um ponto P(x, y, z, 1) é dada por:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & 0 \\ n_x & n_y & n_z & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1a}$$

ou,

$$P' = M^0_{per} . P \tag{3.1b}$$

sendo,

$$d_0 = x_0 . n_x + y_0 . n_y + z_0 . n_z \tag{3.2}$$

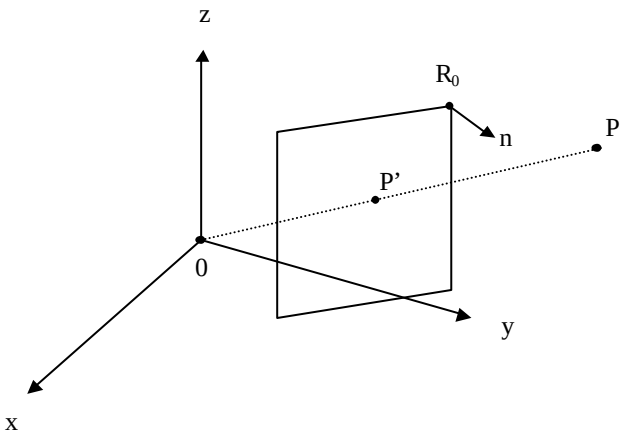


Figura 3.8 - Projeção perspectiva com centro de projeção na origem.

Demonstração:

Como os vetores $P'R_0$ e n são perpendiculares entre si, a equação do plano de projeção é dada por:

$$P'R_0 \bullet n = 0 \tag{3.3}$$

Efetuando o produto obtém-se:

$$(x_0 - x').n_x + (y_0 - y').n_y + (z_0 - z').n_z = 0 \tag{3.4}$$

Introduzindo (3.2) na equação acima:

$$x'. n_x + y'. n_y + z'. n_z = d_0 \tag{3.5}$$

Como P e P' pertencem a uma reta que passa pela origem pode-se escrever:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \alpha \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.6}$$

e substituindo este resultado em (3.5):

$$\alpha . x . n_x + \alpha . y . n_y + \alpha . z . n_z = d_0 \tag{3.7}$$

O valor de α é então dado por:

$$\alpha = \frac{d_0}{x.n_x + y.n_y + z.n_z} \tag{3.8}$$

Em consequência, em coordenadas homogêneas pode-se escrever:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & 0 \\ n_x & n_y & n_z & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.9}$$

Seja agora o mesmo plano de projeção, mas com um centro de projeção C(a, b, c), como mostra a Figura 3.9. A projeção P'(x', y', z', w') de um ponto P(x, y, z, 1) é dada por:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} d + an_x & an_y & an_z & -ad_0 \\ bn_x & d + bn_y & bn_z & -bd_0 \\ cn_x & cn_y & d + cn_z & -cd_0 \\ n_x & n_y & n_z & -d_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.10a}$$

ou,

$$P' = M_{per} \cdot P \tag{3.10b}$$

sendo,

$$d_0 = x_0 \cdot n_x + y_0 \cdot n_y + z_0 \cdot n_z \tag{3.11}$$

$$d_1 = a \cdot n_x + b \cdot n_y + c \cdot n_z \tag{3.12}$$

$$d = d_0 - d_1 \tag{3.13}$$

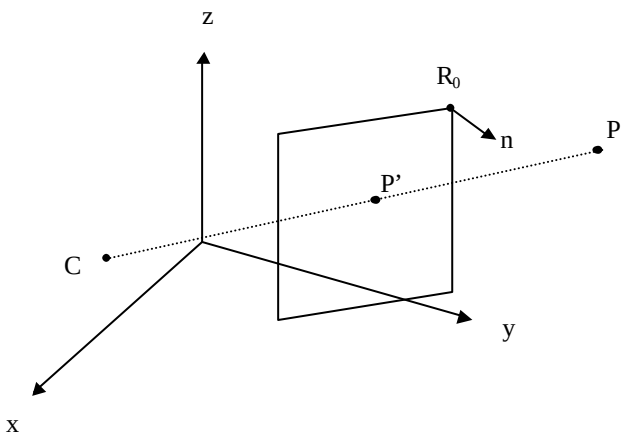


Figura 3.9 - Projeção perspectiva com centro de projeção C(a, b, c).

Demonstração:

A transformação desejada pode ser obtida a partir da seguinte sequência de transformações:

1. Translação T(-a, -b, -c), trazendo o centro de projeção para a origem.
2. Projeção do ponto usando a matriz de projeção em (3.1), para centros de projeção na origem.
3. Translação T(a, b, c), retornando o centro de projeção para a posição original.

Após a translação T(-a, -b, -c), as coordenadas do ponto R_0, pertencente ao plano de projeção, passam a ser (x_0-a, y_0-b, z_0-c). Aplicando a matriz de projeção para centros de projeção na origem obtém-se:

$$M_{per}^{0} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_x & n_y & n_z & 0 \end{bmatrix} \tag{3.14}$$

sendo d igual a,

$$d = (x_0 - a).n_x + (y_0 - b).n_y + (z_0 - c).n_z \tag{3.15}$$

Reagrupando os termos da expressão acima, pode-se escrever:

$$d = (x_0 . n_x + y_0 . n_y + z_0 . n_z) - (a . n_x + b . n_y + c . n_z) = d_0 - d_1 \tag{3.16}$$

Finalmente, aplicando a translação T(a, b, c) obtém-se a transformação desejada:

$$P' = T(a,b,c).M_{per}^{0}.T(-a,-b,-c).P \tag{3.17}$$

ou,

$$P' = M_{per}.P \tag{3.18}$$

sendo a matriz M_{per} igual a:

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ n_x & n_y & n_z & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d + an_x & an_y & an_z & -ad_0 \\ bn_x & d + bn_y & bn_z & -bd_0 \\ cn_x & cn_y & d + cn_z & -cd_0 \\ n_x & n_y & n_z & -d_1 \end{bmatrix} \qquad (3.19)$$

3.2.2 Projeção paralela

Considere um plano de projeção que contém o ponto $R_0(x_0, y_0, z_0)$ de normal $n(n_x, n_y, n_z)$, e uma direção de projeção dada pela direção do vetor v(a, b, c), como mostra a Figura 3.10. A projeção paralela P'(x', y', z', w') de um ponto P(x,y,z,1) é dada pela expressão:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} d_1 - an_x & -an_y & -an_z & ad_0 \\ -bn_x & d_1 - bn_y & -bn_z & bd_0 \\ -cn_x & -cn_y & d_1 - cn_z & cd_0 \\ 0 & 0 & 0 & d_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad (3.20a)$$

ou,

$$P' = M_{par} . P \qquad (3.20b)$$

sendo,

$$d_0 = x_0.n_x + y_0.n_y + z_0.n_z \qquad (3.21)$$

$$d_1 = a.n_x + b.n_y + c.n_z \qquad (3.22)$$

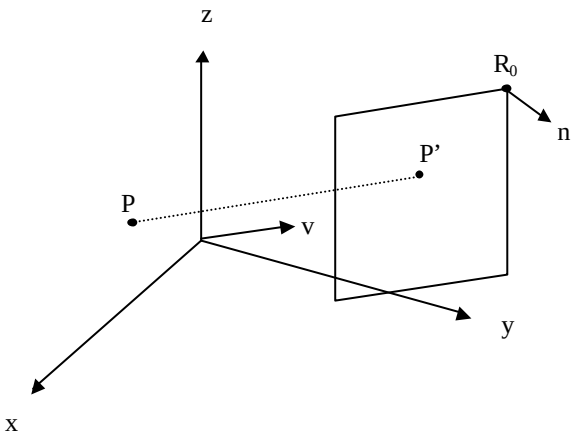


Figura 3.10 - Projeção paralela.

Demonstração:

Equação do plano de projeção:

$$(x_0\text{-}x') \cdot n_x + (y_0\text{-}y') \cdot n_y + (z_0\text{-}z') \cdot n_z = 0 \qquad (3.23)$$

$$x'. n_x + y'. n_y + z'. n_z = d_0 \qquad (3.24)$$

Como P e P' formam uma reta de cossenos diretores (a, b, c), são válidas as relações:

$$P' = P + \begin{bmatrix} a \\ b \\ c \end{bmatrix} . t \qquad (3.25a)$$

ou,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix} . t \qquad (3.25b)$$

onde t é um parâmetro escalar. O ponto de interseção desta reta com o plano de projeção pode ser obtido substituindo a equação acima em (3.24):

$$(x + a.t) \cdot n_x + (y + b.t) \cdot n_y + (z + c.t) \cdot n_z = d_0 \qquad (3.26)$$

e obtendo-se então o valor de t para a interseção:

$$t = \frac{d_0 - (x.n_x + y.n_y + z.n_z)}{d_1} \qquad (3.27)$$

Com o valor de t acima, a projeção P'pode ser expressa por:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix} . \left(\frac{d_0 - (x.n_x + y.n_y + z.n_z)}{d_1} \right) \tag{3.28}$$

ou em coordenadas homogêneas:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} d_1 - an_x & -an_y & -an_z & ad_0 \\ -bn_x & d_1 - bn_y & -bn_z & bd_0 \\ -cn_x & -cn_y & d_1 - cn_z & cd_0 \\ 0 & 0 & 0 & d_1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.29}$$

3.3 MODELO DE VISUALIZAÇÃO 3D

Em termos conceituais, o modelo de visualização é especificado através de uma projeção (perspectiva ou paralela) em um plano de projeção, um volume de visualização (view volume), e uma região de visualização (viewport) do dispositivo de saída, como será visto a seguir.

3.3.1 Volume de visualização

Para a especificação do volume de visualização é necessário definir, além do sistema de coordenadas de mundo (world coordinates - WC), um outro sistema de coordenadas, o sistema de referência de visualização (viewing reference coordinates - VRC). A origem deste sistema é o ponto de referência VRP (view reference point), fornecido em coordenadas de mundo (Figura 3.11).

Um dos eixos do sistema de referência é o eixo n, definido pelo vetor VPN (view plane normal) normal ao plano de projeção. O eixo v é determinado pela projeção do vetor VUP (view up vector), paralela a VPN, em um plano paralelo ao plano de projeção. O outro eixo, o eixo u, é então determinado de modo que o sistema u-v-n forme um triedro direto (Figura 3.11).

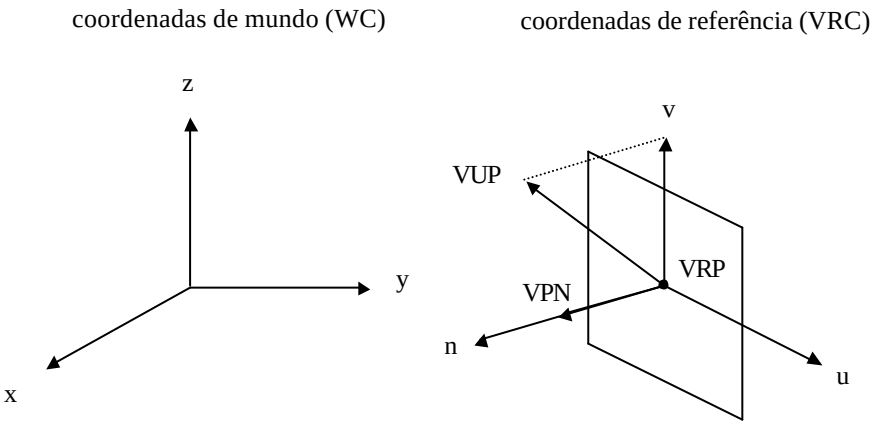


Figura 3.11 - Sistema de coordenadas de referência de visualização.

O plano de projeção (view plane), por definição paralelo ao plano u-v, é determinado por sua distância VPD (view plane distance) à origem de VRC, medida em coordenadas n (Figura 3.12). Uma janela de visualização (view window) do plano de projeção deve ser especificada em termos de coordenadas u e v, como mostra a Figura 3.13.

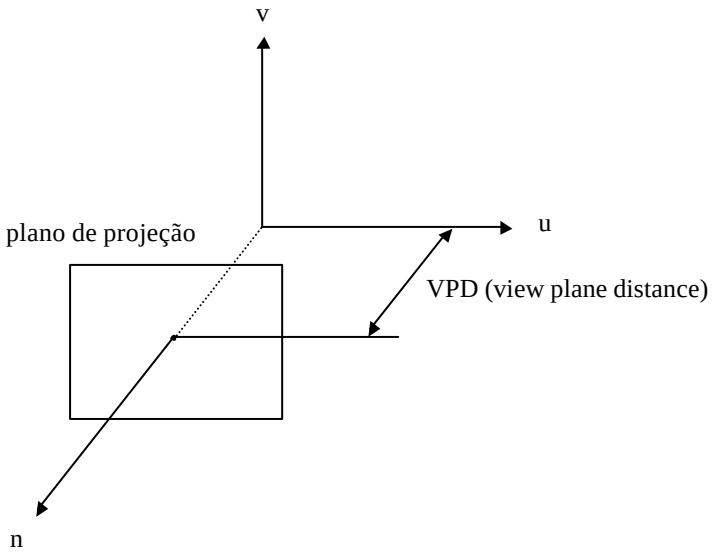


Figura 3.12 - Plano de projeção.

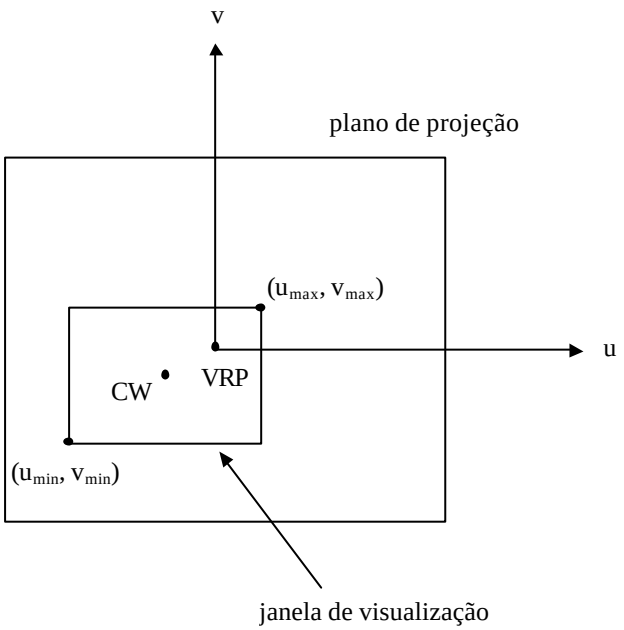


Figura 3.13 - Janela de visualização.

O centro de projeção e a direção de projeção DOP (direction of projection) são definidos pelo ponto de referência de projeção PRP (projection reference point) e um indicador do tipo de projeção. Se a projeção é perspectiva, PRP é o próprio centro de projeção. Se a projeção é paralela, a direção de projeção DOP é definida pela direção da reta PRP-CW, sendo CW o centro da janela de visualização. O ponto PRP é especificado no sistema VRC, e consequentemente sua posição em relação a VRP independe da posição e orientação do sistema VRC em relação ao sistema WC.

Na projeção perspectiva, o volume de visualização é a pirâmide semi-infinita com vértice em PRP e faces passando pelos limites da janela de visualização (Figura 3.14). Nas projeções paralelas, o volume de visualização é o paralelepípedo infinito com faces paralelas à direção de projeção, passando pelos limites da janela de visualização (Figura 3.15).

O volume de visualização deve ser tornado finito através da consideração de dois planos paralelos ao plano de projeção, o plano anterior de recorte (front plane) e o plano posterior de recorte (back plane). Assim como o plano de projeção, estes dois planos são especificados através de suas distâncias F e B à origem de VRC, dadas em coordenadas n (Figura 3.16).

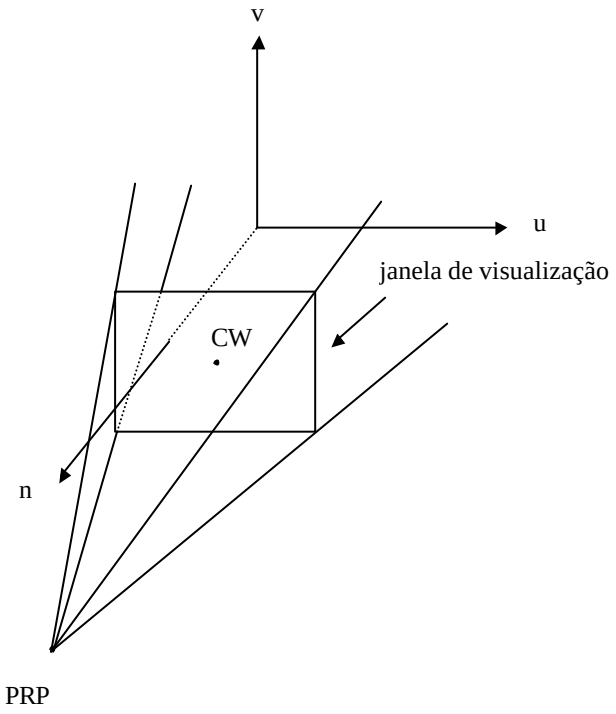


Figura 3.14 - Volume de visualização na projeção perspectiva.

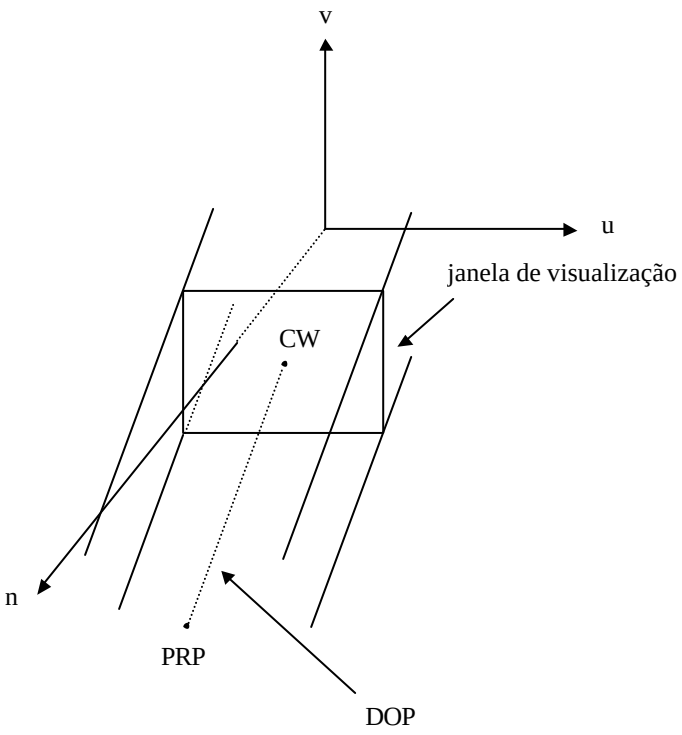


Figura 3.15 - Volume de visualização na projeção paralela.

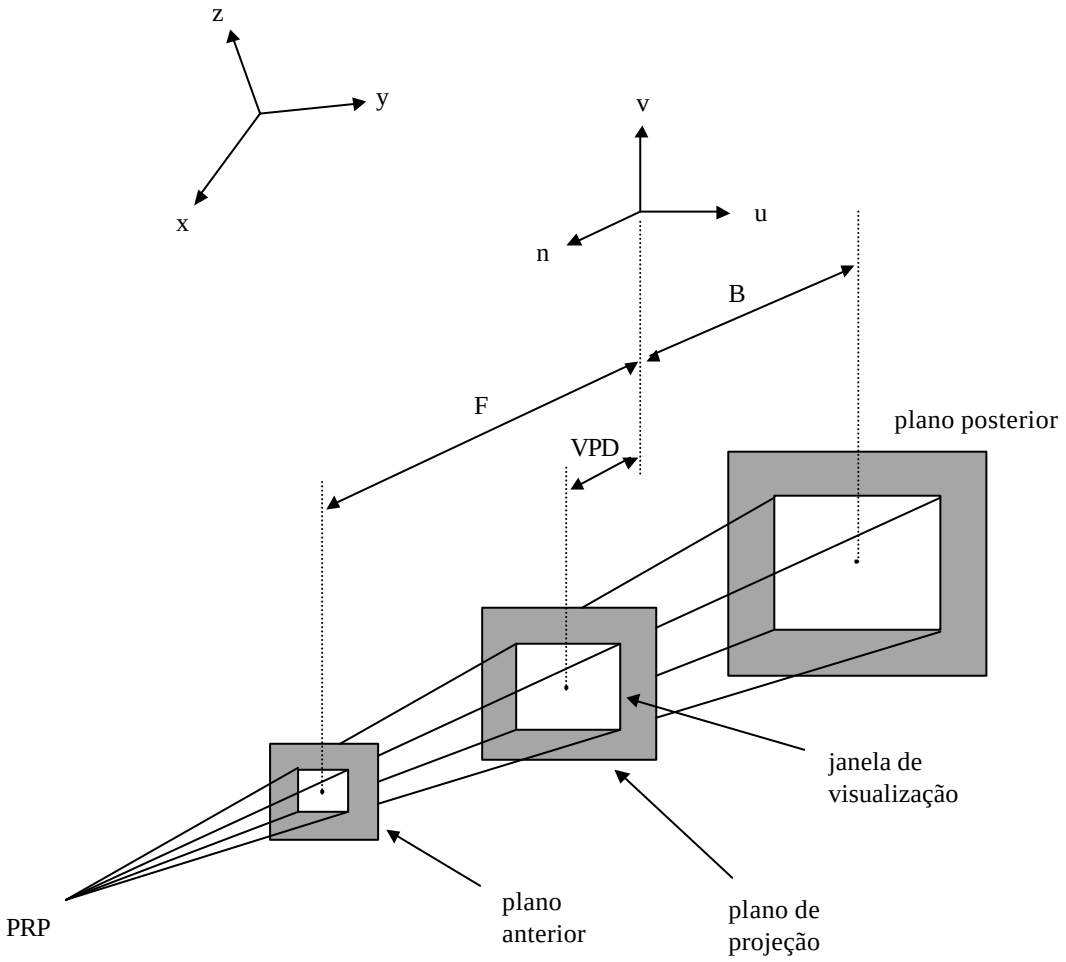


Figura 3.16 - Volume de visualização finito na projeção perspectiva.

3.3.2 Coordenadas normalizadas

Antes de ser mapeado para a região de visualização do dispositivo de saída, o volume de visualização deve ser transformado em um volume de visualização normalizado, definido em um sistema r-s-t de coordenadas normalizadas (NPC - normalized projection coordinates).

Os volumes de visualização normalizados são convenientes, em especial porque simplificam as operações de recorte. As Figuras 3.17 e 3.18 ilustram os dois volumes normalizados adotados no padrão PHIGS, utilizados respectivamente para as projeções perspectiva e paralela.

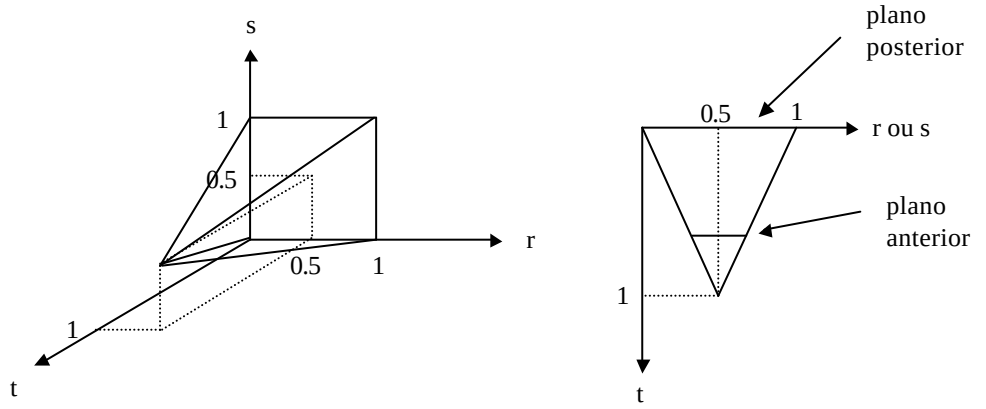


Figura 3.17 - Volume de visualização normalizado - projeção perspectiva.

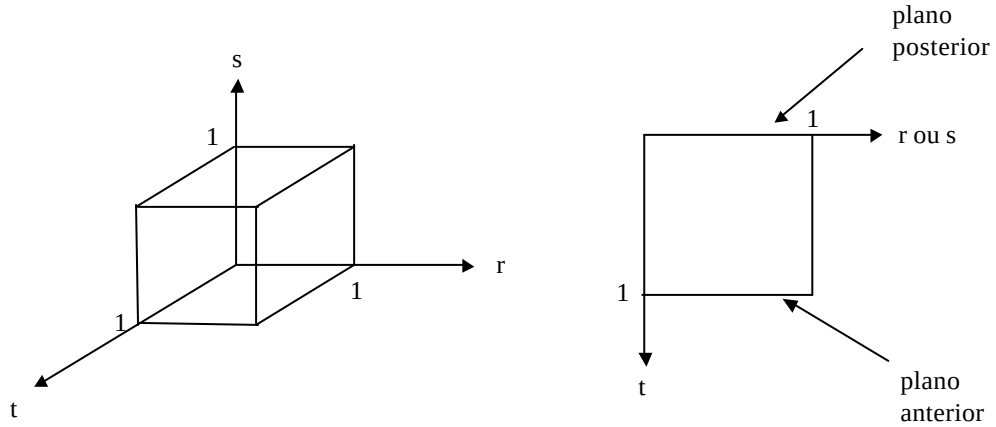


Figura 3.18 - Volume de visualização normalizado - projeção paralela.

3.3.3 Transformações de visualização

O procedimento de visualização 3D de um objeto consiste nas seguintes etapas:

1. Definição do objeto em coordenadas de mundo.

2. Especificação do sistema de coordenadas de referência de visualização VRC, através do ponto de referência VRP e dos vetores VUP e VPN.

3. Especificação do tipo de projeção (perspectiva ou paralela), do ponto de referência de projeção PRP, do plano de projeção, da janela de visualização, e dos planos anterior e posterior de recorte, definido assim o volume de visualização.
4. Especificação da região de visualização no dispositivo de saída gráfica.

5. Transformação das coordenadas do objeto de coordenadas de mundo para coordenadas de referência (WC $\rightarrow$ VRC).

6. Transformação de coordenadas do sistema VRC para o sistema de coordenadas normalizadas NPC, de tal modo que o volume de visualização definido em VRC se transforme no volume normalizado definido em NPC.

7. Operações de recorte 3D efetuadas no volume de visualização normalizado.

8. Determinação da visibilidade das primitivas que descrevem o objeto.

9. Projeção do objeto no plano de projeção.

10. Transformação de coordenadas de tal modo que a janela de visualização definida no plano de projeção se enquadre na região de visualização do dispositivo de saída.

11. Conversão das primitivas.

Dependendo do algoritmo utilizado, o problema da determinação da visibilidade das primitivas pode ser resolvido durante a fase de rasterização. As transformações envolvidas nas etapas 5, 6 e 10 são descritas a seguir, e as operações de recorte são discutidas no item 3.7.

3.3.3.1 Transformação coordenadas de mundo-coordenadas de referência

Dados os pontos VRP(VRP_x,VRP_y,VRP_z) e os vetores VPN e VUP (Figura 3.19), os vetores unitários R_u, R_v e R_n nas direções u, v e n são calculados através das seguintes expressões:

$$R_u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \frac{VUP \times VPN}{|VUP \times VPN|} \tag{3.30}$$

$$R_n = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \frac{VPN}{|VPN|} \tag{3.31}$$

$$R_v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = R_n \times R_u \tag{3.32}$$

e a transformação WC $\rightarrow$ VRC é dada por:

$$\begin{bmatrix} u \\ v \\ n \\ 1 \end{bmatrix} = R \,.\, T(-VRP_x, -VRP_y, -VRP_z) \,.\, \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.33}$$

sendo R a matriz de rotação:

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.34}$$

e T a translação:

$$T = \begin{bmatrix} 1 & 0 & 0 & -VRP_x \\ 0 & 1 & 0 & -VRP_y \\ 0 & 0 & 1 & -VRP_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

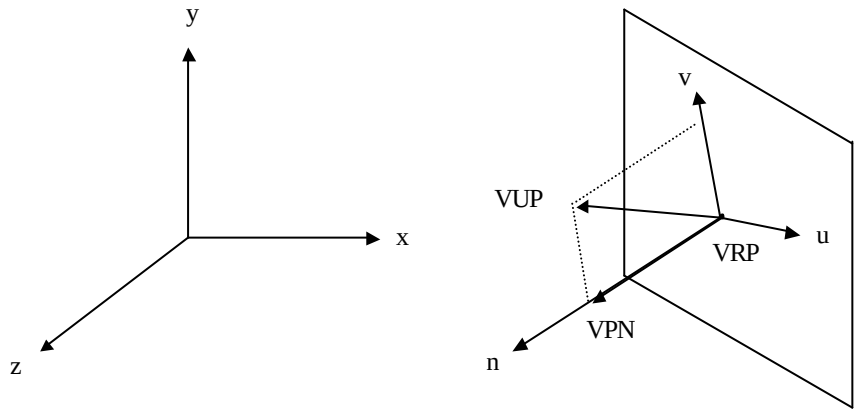


Figura 3.19 - Definição do sistema de coordenadas de referência de visualização.

No padrão PHIGS, o produto R.T corresponde à matriz denominada *view orientation matrix* (VOM):

$$VOM = R.T = \begin{bmatrix} u_x & u_y & u_z & -(u_x.VRP_x + u_y.VRP_y + u_z.VRP_z) \\ v_x & v_y & v_z & -(v_x.VRP_x + v_y.VRP_y + v_z.VRP_z) \\ n_x & n_y & n_z & -(n_x.VRP_x + n_y.VRP_y + n_z.VRP_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.36)$$

3.3.3.2 Transformação coordenadas de referência-coordenadas normalizadas

A transformação do sistema de coordenadas de referência de visualização (VRC) para o sistema de coordenadas normalizadas deve ser tal que o volume de visualização definido em VRC se transforme em um volume normalizado pré-definido. Para isto, são necessárias transformações de translação, mudança de escala e distorção linear (shear transformation).

Em duas dimensões, uma matriz de distorção linear na direção x tem a seguinte forma:

$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.37)$$

onde a é o coeficiente de distorção linear na direção x. A Figura 3.20 mostra o efeito produzido por uma transformação deste tipo. Da mesma forma, uma matriz de distorção linear na direção y tem a forma:

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

onde b é o coeficiente de distorção na direção y (Figura 3.21).

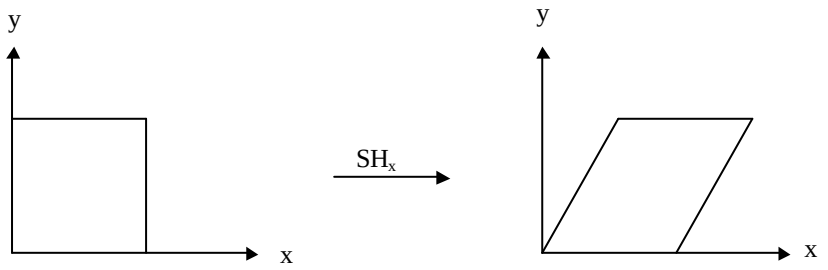


Figura 3.20 - Distorção linear na direção x, em duas dimensões.

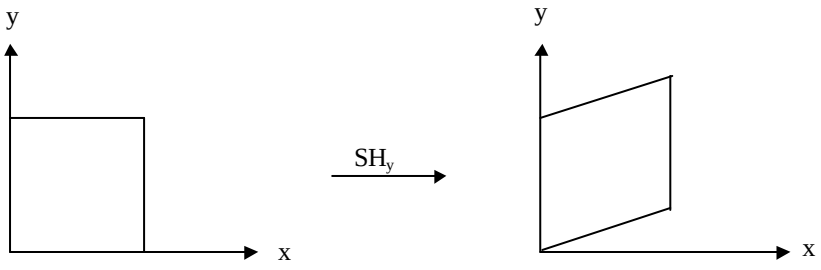


Figura 3.21 - Distorção linear na direção y.

3.3.3.2.1 Projeção paralela

Na projeção paralela, a transformação VRC → NPC deve ser tal que o volume de visualização da Figura 3.22 seja transformado no volume normalizado da Figura 3.18. Considerando que os eixos r-s-t do sistema NPC sejam inicialmente coincidentes com os eixos u-v-n de VRC, a transformação desejada pode ser obtida através das seguintes etapas:

1. Translação T(0,0,-B) tal que o plano posterior fique contido no plano r-s.

2. Transformação de distorção linear em u e v, tal que a direção de projeção seja paralela ao eixo t (Figura 3.23).

3. Translação T(-dr, -ds, 0).

4. Mudança de escala de forma a normalizar o paralelepípedo da Figura 3.24.

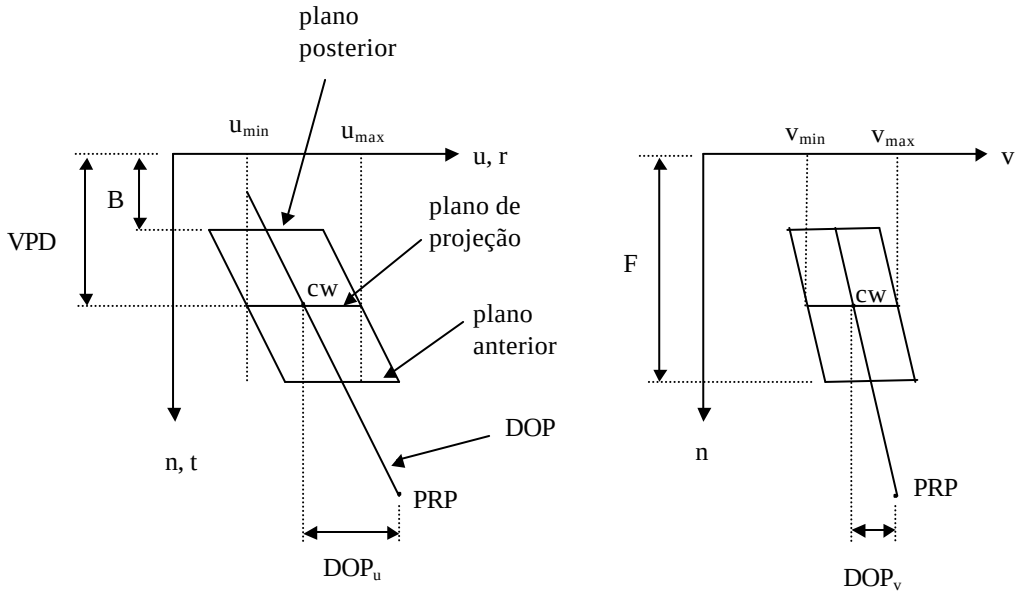


Figura 3.22 - Volume de visualização em VRC - projeção paralela.

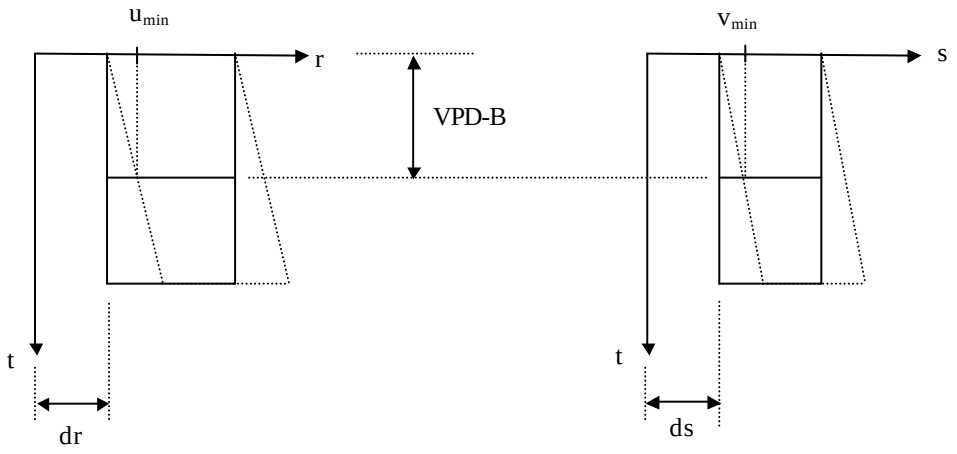


Figura 3.23 - Volume de visualização após translação T(0, 0, -B) e distorção SH_{uv}.

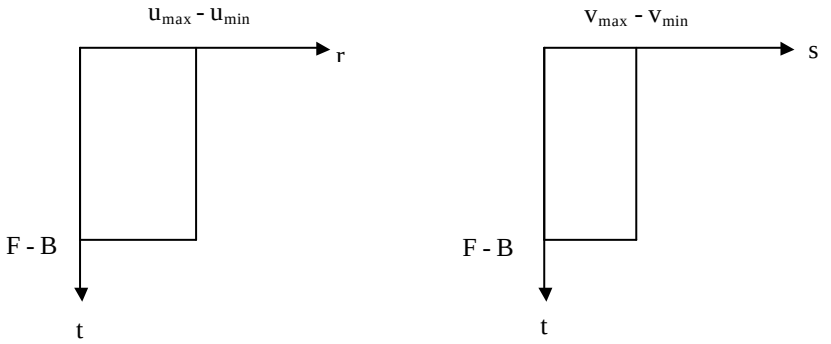


Figura 3.24 - Volume de visualização após translação T(-dr, -ds, 0).

A matriz de distorção linear tem a forma:

$$SH_{uv} = \begin{bmatrix} 1 & 0 & sh_u & 0 \\ 0 & 1 & sh_v & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.39)$$

onde os coeficientes sh_u e sh_v são iguais a:

$$sh_u = -\frac{DOP_u}{DOP_n} \tag{3.40a}$$

$$sh_v = -\frac{DOP_v}{DOP_n} \tag{3.40b}$$

sendo as componentes da direção de projeção dadas por:

$$DOP = PRP - CW = \begin{bmatrix} DOP_u \\ DOP_v \\ DOP_n \end{bmatrix} = \begin{bmatrix} PRP_u \\ PRP_v \\ PRP_n \end{bmatrix} - \begin{bmatrix} (u_{max} + u_{min})/2 \\ (v_{max} + v_{min})/2 \\ VPD \end{bmatrix} \tag{3.41}$$

As translações dr e ds são obtidas a partir das expressões:

$$dr = u_{min} + (VPD\text{-}B) \cdot sh_u \tag{3.42}$$

$$ds = v_{min} + (VPD\text{-}B) \cdot sh_v \tag{3.43}$$

Finalmente, comparando as Figuras 3.24 e 3.18 conclui-se que os fatores da mudança de escala $S(s_u , s_v , s_n)$ devem ser iguais a:

$$s_u = \frac{1}{u_{max} - u_{min}} \tag{3.44}$$

$$s_v = \frac{1}{v_{max} - v_{min}} \tag{3.45}$$

$$s_n = \frac{1}{F - B} \tag{3.46}$$

Com estes resultados, a transformação composta resultante pode ser escrita na forma:

$$N_{par} = S(s_u , s_v , s_n) \,.\, T(-dr, -ds, 0) \,.\, SH_{uv}(sh_u , sh_v) \,.\, T(0, 0, -B) \qquad (3.47)$$

Efetuando o produto acima, obtém-se:

$$N_{par} = \begin{bmatrix} s_u & 0 & s_u.sh_u & -s_u.(B.sh_u + dr) \\ 0 & s_v & s_v.sh_v & -s_v.(B.sh_v + ds) \\ 0 & 0 & s_n & -s_n.B \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.48)$$

Cada ponto do sistema de coordenadas VRC é transformado para o sistema NPC através da relação:

$$\begin{bmatrix} r \\ s \\ t \\ 1 \end{bmatrix} = N_{par} \cdot \begin{bmatrix} u \\ v \\ n \\ 1 \end{bmatrix} \qquad (3.49)$$

No padrão PHIGS, a matriz N_{par} corresponde à matriz denominada *view mapping matrix*, responsável pela transformação do sistema VRC para o sistema NPC.

3.3.3.2.2 Projeção perspectiva

Na projeção perspectiva, o volume de visualização da Figura 3.25 deve ser transformado no volume normalizado da Figura 3.17. A transformação VRC $\rightarrow$ NPC, neste caso, pode ser obtida seguindo-se as etapas:

1. Translação T(0, 0, -B), trazendo o plano posterior para o plano rs.

2. Transformação de distorção linear em u e v, de modo que a direção PRP-CW fique paralela ao eixo n (Figura 3.26).

3. Translação T(-dr, -ds, 0).

4. Mudança de escala tal que o volume em VRC (Figura 3.16) seja mapeado para o volume normalizado (Figura 3.17).

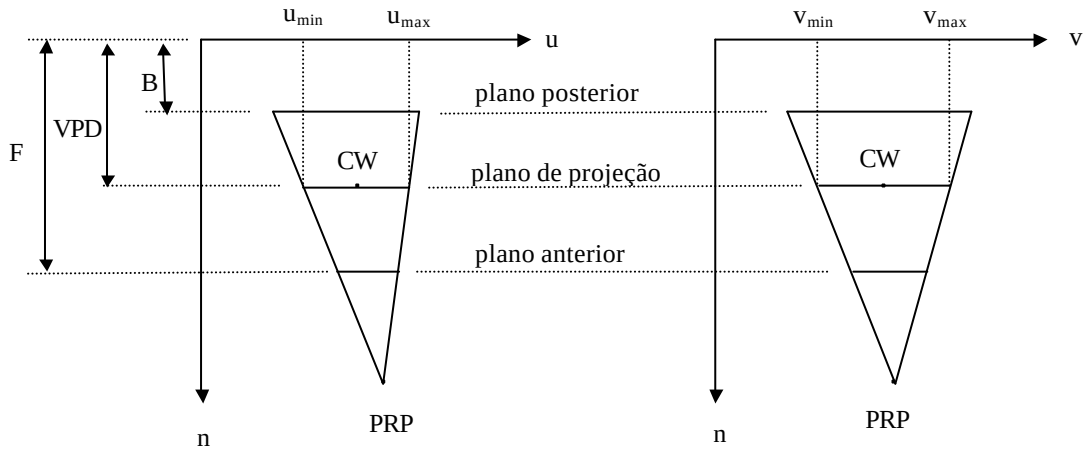


Figura 3.25 - Volume de visualização no sistema VRC.

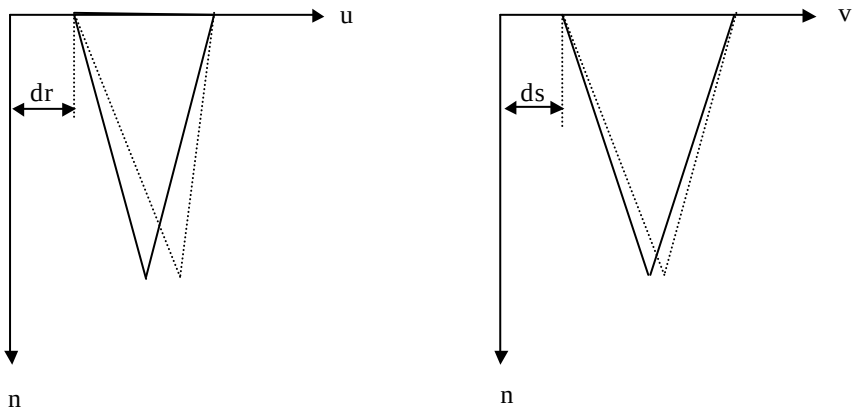


Figura 3.26 - Volume de visualização após translação e distorção linear.

As duas primeiras transformações são idênticas às suas correpondentes na projeção paralela, enquanto que os deslocamentos da translação T(-dr, -ds, 0) são dados por:

$$dr = \frac{(B - PRP_n).u_{min} + PRP_u.(VPD - B)}{VPD - PRP_n} \tag{3.50}$$

$$ds = \frac{(B - PRP_n).v_{min} + PRP_v.(VPD - B)}{VPD - PRP_n} \tag{3.51}$$

Após estas três transformações, o volume de visualização tem a forma de uma pirâmide de base retangular de lados h_u , h_v, e de altura h_n (Figura 3.27):

$$h_u = \frac{(u_{max} - u_{min}).(PRP_n - B)}{PRP_n - VPD} \tag{3.52}$$

$$h_v = \frac{(v_{max} - v_{min}).(PRP_n - B)}{PRP_n - VPD} \tag{3.53}$$

$$h_n = PRP_n - B \tag{3.54}$$

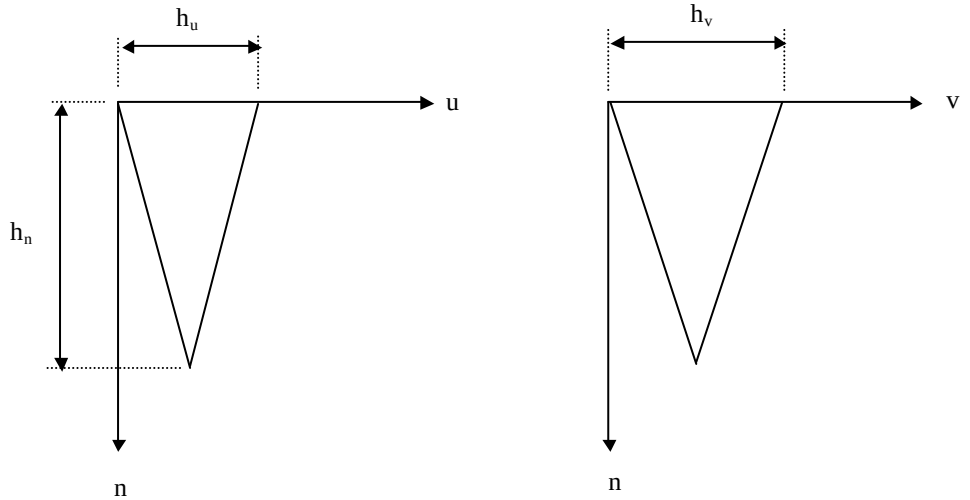


Figura 3.27 - Volume de visualização após translação T(-dr, -ds, 0).

Finalmente, a mudança de escala $S(s_u, s_v, s_n)$ deve ser tal que transforme o volume da Figura 3.27 no volume normalizado. Portanto, os fatores de escala são:

$$s_u = \frac{1}{h_u} \tag{3.55}$$

$$s_v = \frac{1}{h_v} \tag{3.56}$$

$$s_n = \frac{1}{h_n} \tag{3.57}$$

A transformação resultante é :

$$N_{per} = S(s_u , s_v , s_n) \,.\, T(-dr, -ds, 0) \,.\, SH_{uv}(sh_u , sh_v) \,.\, T(0, 0, -B) \tag{3.58}$$

e tem a mesma forma da matriz em (3.48). Cada ponto do sistema de coordenadas VRC é transformado para o sistema NPC através da relação:

$$\begin{bmatrix} r \\ s \\ t \\ 1 \end{bmatrix} = N_{per} \cdot \begin{bmatrix} u \\ v \\ n \\ 1 \end{bmatrix} \tag{3.59}$$

No padrão PHIGS, a matriz N_{per} corresponde à matriz denominada *view mapping matrix*, responsável pela transformação do sistema VRC para o sistema NPC para projeções perspectivas.

3.3.3.3 Transformação coordenadas normalizadas - coordenadas de dispositivo

Os objetos são visualizados no dispositivo de saída através do mapeamento da projeção do volume de visualização normalizado em uma região de visualização (viewport) do dispositivo. No caso da projeção paralela, a projeção é imediata, bastando considerar as coordenadas normalizadas r e s (Figura 3.28). Na projeção perspectiva (Figura 3.29), a projeção das primitivas no plano de projeção é dada pela matriz:

$$M_{per} = \begin{bmatrix} dt-1 & 0 & 0.5 & -dt/2 \\ 0 & dt-1 & 0.5 & -dt/2 \\ 0 & 0 & dt & -dt \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.60}$$

sendo dt igual a:

$$dt = (VPD - B) / (PRP_n - B) \tag{3.61}$$

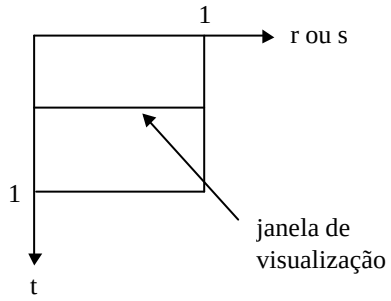


Figura 3.28 - Projeção paralela: janela de visualização em NPC.

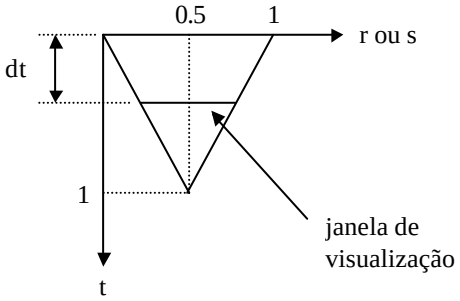


Figura 3.29 - Projeção perpspectiva: janela de visualização em NPC.

A projeção perspectiva é então obtida a partir de :

$$\begin{bmatrix} r' \\ s' \\ t' \\ w \end{bmatrix} = M_{per} \cdot \begin{bmatrix} r \\ s \\ t \\ 1 \end{bmatrix} \tag{3.62}$$

obtendo-se as coordenadas projetadas (r'/w, s'/w, t'/w, 1).

Efetuada a projeção, paralela ou perspectiva, a janela de visualização definida em coordenadas r e s (Figura 3.30) é então mapeada na região de visualização do dispositivo (Figura 3.31). Como foi estudado anteriormente, isto é feito através da matriz M_{wv} :

$$M_{wv} = \begin{bmatrix} s_r & 0 & -r_{min}.s_r + X_{min} \\ 0 & s_s & -s_{min}.s_s + Y_{min} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.63}$$

sendo,

$$s_r = \frac{X_{max} - X_{min}}{r_{max} - r_{min}} \tag{3.64}$$

$$s_s = \frac{Y_{max} - Y_{min}}{s_{max} - s_{min}} \tag{3.65}$$

Um ponto P(X,Y) da região de visualização tem então as seguintes coordenadas:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = M_{wv}.\begin{bmatrix} r' \\ s' \\ 1 \end{bmatrix} \tag{3.66}$$

sendo r' e s' as coordenadas projetadas.
Tendo em vista que na projeção paralela a janela de visualização é dada por:

$$r_{min} = s_{min} = 0 \tag{3.67}$$

$$r_{max} = s_{max} = 1 \tag{3.68}$$

Os fatores de escala se reduzem a:

$$s_r = X_{max} - X_{min} \tag{3.69}$$

$$s_s = Y_{max} - Y_{min} \tag{3.70}$$

Na projeção perspectiva tem-se que:

$$r_{min} = s_{min} = dt/2 \quad (3.71)$$

$$r_{max} = s_{max} = 1\text{-}dt/2 \quad (3.72)$$

e os fatores de escala são:

$$s_r = (X_{max} - X_{min}) / (1 - dt) \quad (3.73)$$

$$s_s = (Y_{max} - Y_{min}) / (1 - dt) \quad (3.74)$$

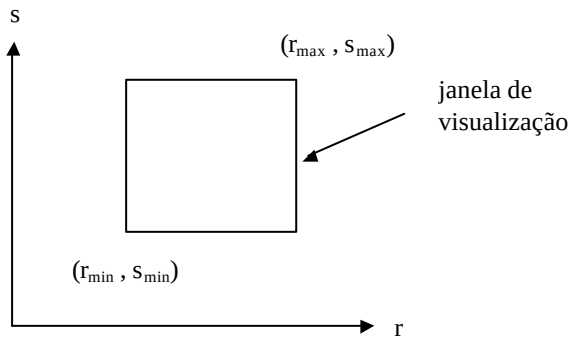


Figura 3.30 - Limites da janela de visualização em NPC.

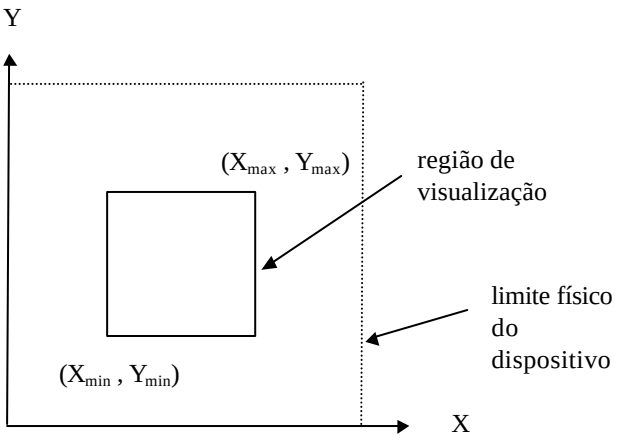


Figura 3.31 - Região de visualização.

3.4 RECORTE EM TRÊS DIMENSÕES

O algoritmo de Sutherland-Hodgman, descrito no capítulo 1, pode ser estendido, em três dimensões, para o recorte de polígonos convexos ou côncavos contra um poliedro convexo. Análogo ao caso bidimensional, o recorte é feito considerando cada face do poliedro isoladamente, uma por vez. A Figura 3.32 mostra o recorte de um retângulo contra um hexaedro.

O interior do volume de recorte é identificado pelas normais externas aos planos que definem o contorno do poliedro. Portanto, o problema agora é saber se um ponto se situa no lado interno ou externo de cada face do poliedro. Conhecendo-se um ponto $P_0(x_0, y_0, z_0)$ da face e o vetor unitário $n(n_x, n_y, n_z)$, que representa a normal externa, um ponto P(x, y, z) está situado no lado interno se:

$$(P_0P) \bullet n < 0 \tag{3.75}$$

Desenvolvendo o produto acima, obtém-se:

$$(x - x_0).n_x + (y - y_0).n_y + (z - z_0).n_z < 0 \tag{3.76}$$

$$x.n_x + y.n_y + z.n_z - (x_0.n_x + y_0.n_y + z_0.n_z) < 0 \tag{3.77}$$

Fazendo,

$$d_0 = x_0.n_x + y_0.n_y + z_0.n_z \tag{3.78}$$

chega-se à expressão:

$$x.n_x + y.n_y + z.n_z < d_0 \tag{3.79}$$

que deve ser satisfeita para pontos do lado interno da face. A Figura 3.33 ilustra esta situação.

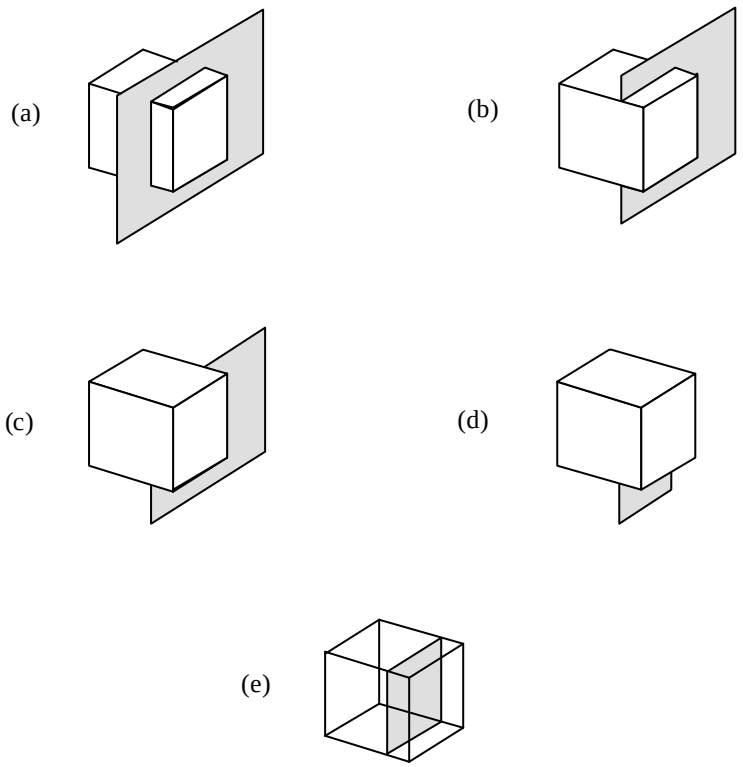


Figura 3.32 - Recorte de um retângulo contra um hexaedro: (a) retângulo antes do recorte, (b)-(d) polígonos intermediários, e (e) resultado final.

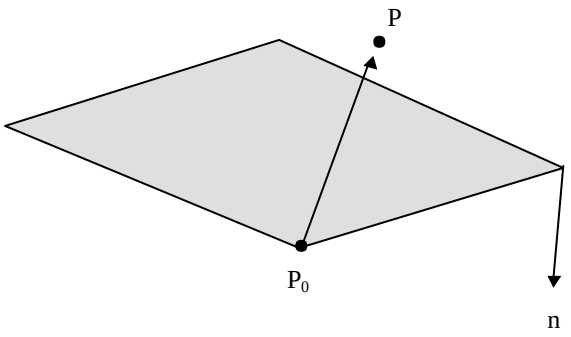


Figura 3.33 - Ponto P no lado interno da face: projeção do vetor P_0P negativa na direção da normal externa n.

No algoritmo 2D, a função *intersection* calculava a interseção entre duas retas. Aqui, as interseções a serem determinadas são entre uma aresta e um plano infinito. Seja, por exemplo, uma aresta de extremidades $P_0(x_0 , y_0 , z_0)$ e $P_1(x_1 , y_1 , z_1)$, e um plano que contém o ponto $P(x_p , y_p , z_p)$, de normal externa $n(n_x , n_y , n_z)$. As equações paramétricas da reta são:

$$x = x_0 + (x_1 - x_0).t \tag{3.80}$$

$$y = y_0 + (y_1 - y_0).t \qquad (0 \leq t \leq 1) \tag{3.81}$$

$$z = z_0 + (z_1 - z_0).t \tag{3.82}$$

e a equação do plano é dada por:

$$x.n_x + y.n_y + z.n_z = x_p.n_x + y_p.n_y + z_p.n_z \tag{3.83}$$

Substituindo (3.80)-(3.82) na equação acima obtém-se a expressão:

$$\begin{aligned} &[x_0 + (x_1 - x_0).t]n_x + [y_0 + (y_1 - x_0).t]n_y + [z_0 + (z_1 - z_0).t]n_z = \\ &= x_p.n_x + y_p.n_y + z_p.n_z \end{aligned} \tag{3.84}$$

de onde se tira o valor de t para o ponto de interseção da reta com o plano:

$$t = \frac{(x_p - x_0).n_x + (y_p - y_0).n_y + (z_p - z_0).n_z}{(x_1 - x_0).n_x + (y_1 - y_0).n_y + (z_1 - z_0).n_z} \tag{3.85}$$

Essencialmente, a versão tridimensional do algoritmo de Sutherland (Figura 3.34) difere do algoritmo bidimensional descrito no item 17 apenas no conteúdo das funções *inside* e *intersection*. O argumento de entrada *edge*, correspondente à aresta de recorte naquele algoritmo, agora é substituído por um plano de recorte, especificado através de um ponto e pelo vetor normal unitário, que são armazenados na estrutura *plane*. No lugar da estrutura POINT2D, que antes representava pontos em duas dimensões, utiliza-se a estrutura POINT3D. A função *SHPolygonClip3D* da Figura 3.34 deve ser chamada em sequência, uma vez para cada face do poliedro de recorte.

Para o caso particular de recorte contra os volumes de visualização normalizados descritos no item 3.3.2, é conveniente usar uma rotina que contenha funções *inside* e *intersection* específicas para cada uma das 6 faces do volume, reduzindo assim o número de operações de ponto flutuante. Por exemplo, o plano posterior de recorte, tanto na projeção perspectiva como na paralela, pode ser especificado através do ponto (0, 0, 0) e pela normal (0, 0, -1). Isto reduz as expressões (3.79) e (3.85) a:

$$z > 0 \tag{3.86}$$

$$t = \frac{z_0}{z_0 - z_1} \tag{3.87}$$

Como foi dito anteriormente, simplificar as operações de recorte é uma das vantagens do volume de visualização normalizado.

```
#define MAX 100
#define TRUE  1
#define FALSE 0

typedef char BOOL;
typedef struct {
        float x;
        float y;
        float z;
} POINT3D;

POINT3D v[MAX],vclip[MAX],plane[2];

/*  Declaração de funções :                                                 */

        POINT3D intersection(POINT3D s,POINT3D p,POINT3D plane[]);
        int addVertex(POINT3D i,int nclip,POINT3D vclip[]);
        BOOL inside(POINT3D p,POINT3D plane[]);

/*  Recorte de um polígono contra um plano infinito pelo método de Sutherland-Hodgman :  */

int SHPolygonClip3D(POINT3D v[],int n,POINT3D plane[],POINT3D vclip[])
{
        int j,nclip;
        POINT3D s,p,i;

        nclip = 0;
        s = v[n-1];
        for (j = 0; j < n; j++) {
                p = v[j];
                if (inside(p,plane))
                        if (inside(s,plane))
                                nclip = addVertex(p,nclip,vclip);
                        else {
                                i = intersection(s,p,plane);
                                nclip = addVertex(i,nclip,vclip);
                                nclip = addVertex(p,nclip,vclip);
                        }
                else if (inside(s,plane)) {
                                i = intersection(s,p,plane);
                                nclip = addVertex(i,nclip,vclip);
                }
                s = p;
        }
        return nclip;
```

Figura 3.34 - Recorte de polígonos contra um plano infinito

```
/*   Determina se um ponto está localizado no lado interno de um plano:   */

BOOL inside(POINT3D p,POINT3D plane[])
{
        if ((p.x*plane[1].x+p.y*plane[1].y+p.z*plane[1].z) <
            (plane[0].x*plane[1].x+plane[0].y*plane[1].y+plane[0].z*plane[1].z))
        return TRUE;
        return FALSE;
}

/* Calcula a interseção de uma aresta com um plano :      */

POINT3D intersection(POINT3D s,POINT3D p,POINT3D plane[])
{
        float a,b,c,t;
        POINT3D i;

        a = p.x - s.x;
        b = p.y - s.y;
        c = p.z - s.z;
        t =  ( (plane[0].x-s.x)*plane[1].x + (plane[0].y-s.y)*plane[1].y +
               (plane[0].z-s.z)*plane[1].z ) /
            ( a*plane[1].x + b*plane[1].y + c*plane[1].z );
        i.x = s.x + a*t;
        i.y = s.y + b*t;
        i.z = s.z + c*t;

        return i;
}

/* Adiciona um vértice ao polígono recortado  :    */

int addVertex(POINT3D i,int nclip,POINT3D vclip[])
{
        vclip[nclip].x = i.x;
        vclip[nclip].y = i.y;
        vclip[nclip].z = i.z;
        return (nclip+1);
}
```

Figura 3.34 (Continuação)

4 REPRESENTAÇÃO DE CURVAS E SUPERFÍCIES

Antes de serem visualizados, os objetos devem ser modelados geometricamente, ou discretizados por elementos ou primitivas. Estes elementos podem ser de uma, duas, ou três dimensões, conforme o número de parâmetros necessários para descrevê-los (Figura 4.1). Podem também ser lineares ou de grau superior, dependendo do grau p de suas equações paramétricas. Frequentemente utiliza-se o termo *wire-frame* para designar a representação de um objeto por meio de elementos unidimensionais, que são segmentos de reta (lineares) ou segmentos de curva ($p > 1$). Os elementos bidimensionais são superfícies curvas, quando o grau dos poliômios em s e t é maior do que 1, ou faces planas ou polígonos, quando as equações paramétricas são lineares. Como as operações de conversão são tipicamente bidimensionais, os objetos sólidos devem ser representados por pequenos volumes vazios, definidos por uma associação de elementos de superfície. Poliedros são um exemplo de elementos sólidos lineares.

A utilização de elementos lineares na visualização de um objeto facilita as operações de conversão, de recorte e de determinação da visibilidade dos elementos. No entanto, a representação paramétrica de elementos de ordem superior se constitui em uma ferramenta poderosa na geração de elementos lineares. Assim, uma forma geométrica pode ser inicialmente representada por equações paramétricas cúbicas, por exemplo, e posteriormente subdividida em vários elementos lineares, conforme a precisão desejada. Os conceitos básicos sobre a representação paramétrica de curvas e superfícies [13] são apresentados a seguir.

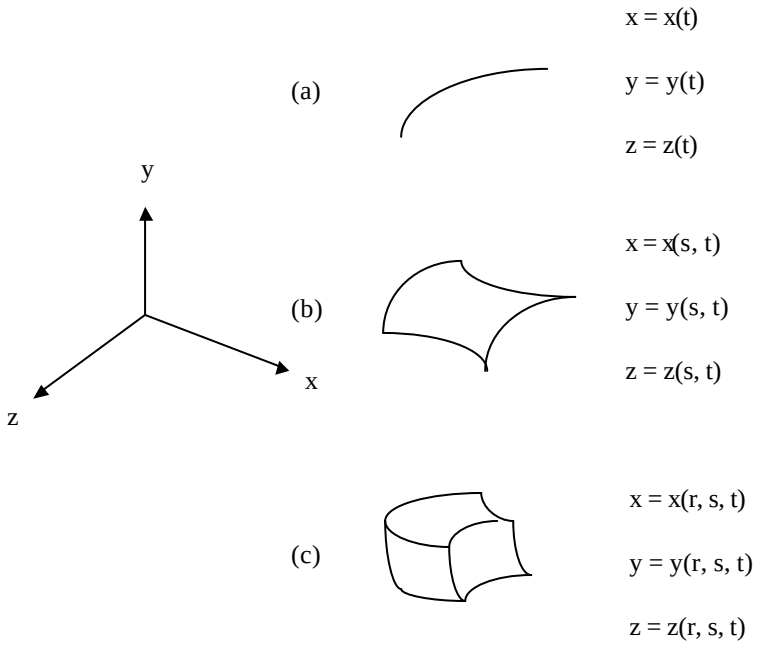


Figura 4.1 - Elementos unidimensionais (a), bidimensionais (b) e tridimensionais (c).

4.1 CURVAS PARAMÉTRICAS

A representação paramétrica de curvas,

$$x = x(t) \tag{4.1}$$

$$y = y(t) \tag{4.2}$$

$$z = z(t) \tag{4.3}$$

é útil em muitos casos. Uma curva genérica pode ser representada por segmentos, sendo cada segmento Q definido por polinômios de grau *p* no parâmetro t. Por exemplo, os polinômios cúbicos que definem um segmento,

$$Q(t) = [x(t) \quad y(t) \quad z(t)] \tag{4.4}$$

são dados por:

$$x(t) = a_x \cdot t^3 + b_x \cdot t^2 + c_x \cdot t + d_x \tag{4.5}$$

$$y(t) = a_y \cdot t^3 + b_y \cdot t^2 + c_y \cdot t + d_y \qquad (0 \leq t \leq 1) \tag{4.6}$$

$$z(t) = a_z \cdot t^3 + b_z \cdot t^2 + c_z \cdot t + d_z \tag{4.7}$$

Matricialmente, o segmento Q é representado pela expressão:

$$Q(t) = [\, x(t) \;\; y(t) \;\; z(t) \,] = T \cdot C \tag{4.8}$$

onde T e a matriz de coeficientes C são iguais a:

$$T = [\, t^3 \quad t^2 \quad t \quad 1 \,] \tag{4.9}$$

$$C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix} \tag{4.10}$$

A derivada de Q(t) representa o vetor tangente à curva:

$$\frac{d}{dt}Q(t) = Q'(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t) \quad \frac{d}{dt}z(t) \right] =$$

$$= \frac{dT}{dt}.C = \left[3t^2 \quad 2t \quad 1 \quad 0\right] C \tag{4.11}$$

Uma curva formada simplesmente pela conexão de dois segmentos tem continuidade geométrica G^0. Se as direções (mas não necessariamente as magnitudes) dos dois vetores tangentes são iguais no ponto de conexão, então a curva tem continuidade geométrica G^1. Se os dois vetores tangentes à curva no ponto de conexão são iguais em direção e magnitude, então a curva tem continuidade C^1 (continuidade paramétrica). Generalizando, se as direções e magnitudes das derivadas até a ordem n são iguais, então diz-se que a curva é de continuidade C^n. O vetor Q'(t) corresponde à velocidade de um ponto da curva em relação ao parâmetro t, e a segunda derivada Q''(t) representa a aceleração. Em geral continuidade C^1 implica em continuidade geométrica G^1, mas a recíproca não é verdadeira, isto é, uma curva pode ter continuidade G^1 e não ter continuidade C^1. A exceção ocorre quando os dois vetores tangentes à curva no ponto de conexão são iguais a (0, 0, 0). Neste caso, por definição, a curva apresenta continuidade C^1, mas as tangentes no ponto de conexão podem ser diferentes, como mostra a Figura 4.2.

Um polinômio cúbico tem quatro coeficientes e, portanto, são necessárias quatro condições para determiná-los. Por exemplo, nos polinômios cúbicos de Hermite, as condições impostas são os dois pontos extremos e seus dois vetores tangentes. Os polinômios cúbicos de Bézier, por outro lado, são definidos pelos dois pontos extremos e mais dois pontos não pertencentes à curva que definem as tangentes nas extremidades.

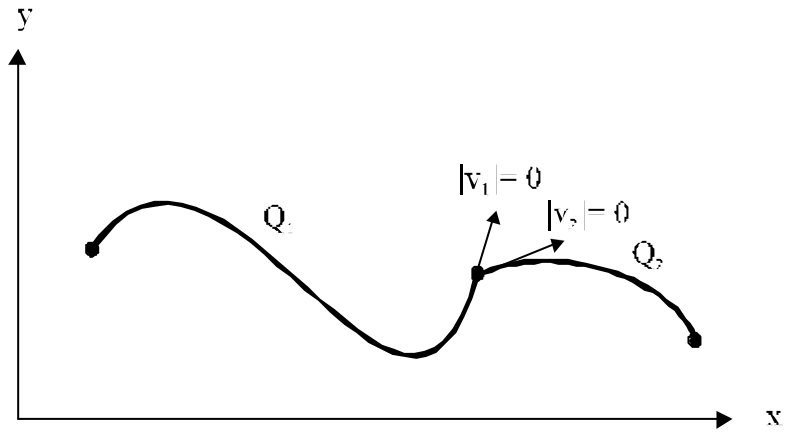


Figura 4.2 - Caso especial de continuidade paramétrica maior do que continuidade geométrica.

A matriz de coeficientes C de um polinômio cúbico pode ser escrita sob a forma de um produto de uma matriz base M, de ordem 4x4, por outra outra matriz G (matriz geométrica), que representa as condições geométricas impostas:

$$C = M \cdot G \tag{4.12}$$

sendo,

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \tag{4.13}$$

$$G = \begin{bmatrix} G_x & G_y & G_z \end{bmatrix} = \begin{bmatrix} g_{1x} & g_{1y} & g_{1z} \\ g_{2x} & g_{2y} & g_{2z} \\ g_{3x} & g_{3y} & g_{3z} \\ g_{4x} & g_{4y} & g_{4z} \end{bmatrix} \tag{4.14}$$

As colunas G_x, G_y e G_z da matriz G representam as condições geométricas que correspondem a x, y e z, respectivamente, e os coeficientes m_{ij} da matriz M dependem destas condições. Deste modo, um segmento de curva Q(t) pode ser representado por:

$$Q(t) = [\, x(t) \quad y(t) \quad z(t) \,] = T \cdot C = T \cdot M \cdot G \tag{4.15}$$

ou,

$$x(t) = T \cdot M \cdot G_x \tag{4.16}$$

$$y(t) = T \cdot M \cdot G_y \tag{4.17}$$

$$z(t) = T \cdot M \cdot G_z \tag{4.18}$$

4.1.1 Polinômios de Lagrange

Os polinômios cúbicos de Lagrange são definidos pelos pontos extremos e mais dois pontos pertencentes à curva (Figura 4.3). Matricialmente, são representados pela expressão:

$$Q(t) = T \cdot M_L \cdot G_L \qquad (0 \leq t \leq 1) \qquad (4.19)$$

onde M_L é a matriz base e G_L é a matriz geométrica de Lagrange:

$$G_L = \begin{bmatrix} G_{Lx} & G_{Ly} & G_{Lz} \end{bmatrix} = \begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{2x} & P_{2y} & P_{2z} \\ P_{3x} & P_{3y} & P_{3z} \\ P_{4x} & P_{4y} & P_{4z} \end{bmatrix} \qquad (4.20)$$

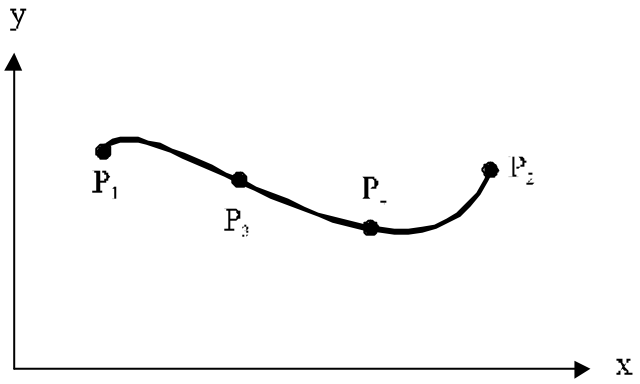


Figura 4.3 - Polinômios cúbicos de Lagrange.

Considerando a direção x, as condições geométricas são:

$$x(0) = P_{1x} = [\,0\ \ 0\ \ 0\ \ 1\,]\, M_L \cdot G_{Lx} \qquad (4.21)$$

$$x(1) = P_{2x} = [\,1\ \ 1\ \ 1\ \ 1\,]\, M_L \cdot G_{Lx} \qquad (4.22)$$

$$x(1/3) = P_{3x} = [\,1/27\ \ 1/9\ \ 1/3\ \ 1\,]\, M_L \cdot G_{Lx} \qquad (4.23)$$

$$x(2/3) = P_{4x} = [\,8/27\ \ 4/9\ \ 2/3\ \ 1\,]\, M_L \cdot G_{Lx} \qquad (4.24)$$

ou de forma mais compacta:

$$\begin{bmatrix} P_{1x} \\ P_{2x} \\ P_{3x} \\ P_{4x} \end{bmatrix} = G_{Lx} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1/27 & 1/9 & 1/3 & 1 \\ 8/27 & 4/9 & 2/3 & 1 \end{bmatrix} M_L \,.G_{Lx} \tag{4.25}$$

Para que a expressão acima seja válida, a matriz base deve ser igual a:

$$M_L = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1/27 & 1/9 & 1/3 & 1 \\ 8/27 & 4/9 & 2/3 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} -9/2 & 9/2 & 27/2 & -27/2 \\ 9 & -9/2 & -45/2 & 18 \\ -11/2 & 1 & 9 & -9/2 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.26}$$

Fazendo,

$$B_L = T.\, M_L \tag{4.27}$$

e introduzindo este resultado em (4.19), obtém-se:

$$Q(t) = B_L \,.\, G_L = [\; B_L.G_{Lx} \quad B_L.G_{Ly} \quad B_L.G_{Lz} \;] \tag{4.28}$$

onde B_L representa as funções de interpolação de Lagrange:

$$B_L = [\; L_1 \quad L_2 \quad L_3 \quad L_4 \;] \tag{4.29}$$

Estas funções, representadas graficamente na Figura 4.4, são iguais a:

$$L_1 = (-9/2)\, t^3 + 9\, t^2 - (11/2)\, t + 1 \tag{4.30}$$

$$L_2 = (9/2)\, t^3 - (9/2)\, t^2 + t \tag{4.31}$$

$$L_3 = (27/2)\, t^3 - (45/2)\, t^2 + 9\, t \tag{4.32}$$

$$L_4 = -(27/2)\, t^3 + 18\, t^2 - (9/2)\, t \tag{4.33}$$

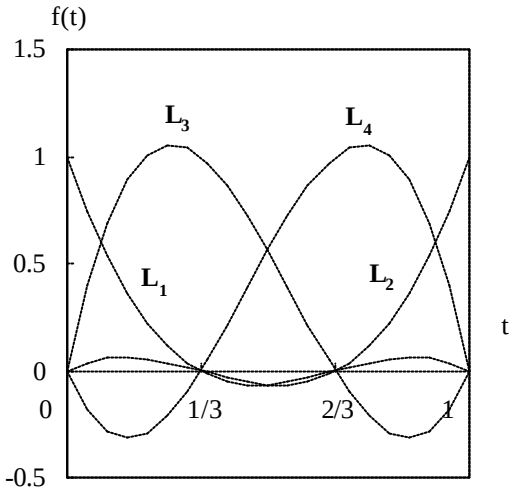


Figura 4.4 - Funções de interpolação de Lagrange.

Inspecionando as equações (4.30)-(4.33), conclui-se que as funções de interpolação satisfazem as relações:

$$L_1 + L_2 + L_3 + L_4 = 1 \qquad (0 \le t \le 1) \tag{4.34}$$

$$L_i = \begin{cases} 1, & t = t_i \\ 0, & t = t_j \quad (j \neq i) \end{cases} \tag{4.35}$$

onde t_i representa o valor de t nos pontos P_i (i = 1,..,4). A derivada Q'(t) é igual a:

$$Q'(t) = [3t^2 \quad 2t \quad 1 \quad 0].M_L.G_L = \left(\frac{d}{dt}B_L\right).G_L \tag{4.36}$$

e, portanto, a componente do vetor velocidade na direção x é dada por:

$$\frac{d}{dt}x(t) = \frac{d}{dt}L_1.P_{1x} + \frac{d}{dt}L_2.P_{2x} + \frac{d}{dt}L_3.P_{3x} + \frac{d}{dt}L_4.P_{4x} =$$

$$= \left(-\frac{27}{2}\,t^2 + 18t - \frac{11}{2}\right).P_{1x} + \left(\frac{27}{2}\,t^2 - 9t + 1\right).P_{2x} +$$

$$+ \left(\frac{81}{2}\,t^2 - 45t + 9\right).P_{3x} + \left(-\frac{81}{2}\,t^2 - 36t - \frac{9}{2}\right).P_{4x} \tag{4.37}$$

Reagrupando os termos da expressão acima, obtém-se:

$$\frac{d}{dt}x(t) = \left(-\frac{27}{2}P_{1x} + \frac{27}{2}P_{2x} + \frac{81}{2}P_{3x} - \frac{81}{2}P_{4x}\right)t^2 +$$

$$+\left(18P_{1x} - 9P_{2x} - 45P_{3x} - 36P_{4x}\right)t + \tag{4.38}$$

$$\left(-\frac{11}{2}P_{1x} + P_{2x} + 9P_{3x} - \frac{9}{2}P_{4x}\right)$$

Para que esta componente de velocidade seja constante, os termos em t e t^2 devem ser iguais a zero. Isto acontece se os pontos P_i estão igualmente espaçados em x (ver Figura 4.5), ou seja:

$$\Delta x = P_{2x} - P_{1x} \tag{4.39}$$

$$P_{3x} = P_{1x} + \frac{\Delta x}{3} \tag{4.40}$$

$$P_{4x} = P_{2x} - \frac{\Delta x}{3} \tag{4.41}$$

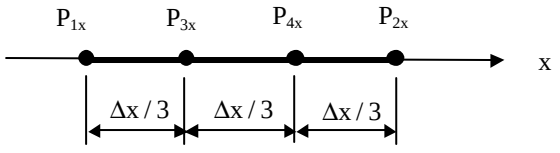


Figura 4.5 - Espaçamento uniforme em x.

De fato, para que a velocidade ao longo de uma curva seja constante, o espaçamento entre os pontos P_i deve ser uniforme ou, em outras palavras, a intervalos constantes Δt devem corresponder distâncias iguais ao longo da curva.

Para um polinômio genérico de grau p, as condições geométricas são dadas por p+1 pontos pertencentes à curva, e as funções de interpolação são obtidas através da expressão:

$$L_i^p = \frac{(t - t_1).....(t - t_{i-1})(t - t_{i+1}).....(t - t_{p+1})}{(t_i - t_1)....(t_i - t_{i-1})(t_i - t_{i+1}).....(t_i - t_{p+1})} \qquad (4.42)$$

4.1.2 Polinômios cúbicos de Hermite

Um polinômio cúbico de Hermite é determinado pelos pontos extremos P_1, P_2 e por seus vetores tangentes R_1 e R_2 (ver Figura 4.6), sendo representado pela expressão:

$$Q(t) = T \,.\, M_H \,.\, G_H \qquad (0 \leq t \leq 1) \qquad (4.43)$$

onde M_H é a matriz base de Hermite e G_H a matriz geométrica associada:

$$G_H = \begin{bmatrix} G_{Hx} & G_{Hy} & G_{Hz} \end{bmatrix} = \begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{2x} & P_{2y} & P_{2z} \\ R_{1x} & R_{1y} & R_{1z} \\ R_{2x} & R_{2y} & R_{2z} \end{bmatrix} \qquad (4.44)$$

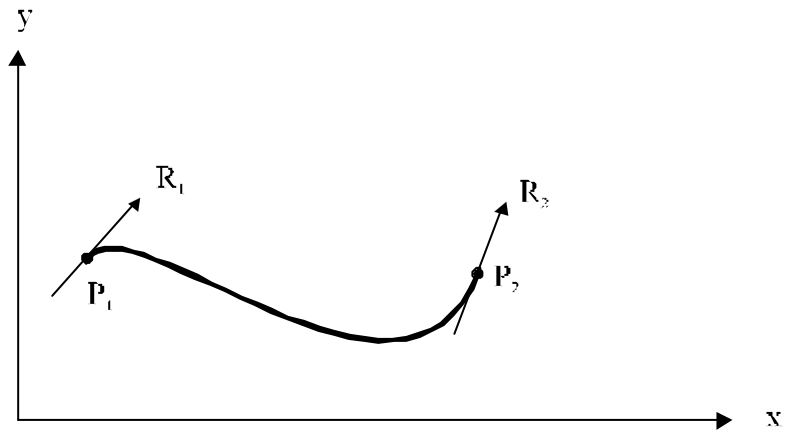


Figura 4.6 - Polinômios cúbicos de Hermite.

Considerando por simplicidade a direção x, as condições nos pontos extremos são dadas por:

$$x(0) = P_{1x} = [\,0\ 0\ 0\ 1\,]\, M_H \,.\, G_{Hx} \tag{4.45}$$

$$x(1) = P_{2x} = [\,1\ 1\ 1\ 1\,]\, M_H \,.\, G_{Hx} \tag{4.46}$$

$$x'(0) = R_{1x} = [\,0\ 0\ 1\ 0\,]\, M_H \,.\, G_{Hx} \tag{4.47}$$

$$x'(1) = R_{2x} = [\,3\ 2\ 1\ 0\,]\, M_H \,.\, G_{Hx} \tag{4.48}$$

ou de modo equivalente,

$$\begin{bmatrix} P_{1x} \\ P_{2x} \\ R_{1x} \\ R_{2x} \end{bmatrix} = G_{Hx} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_H \,.\, G_{Hx} \tag{4.49}$$

Para a expressão acima ser satisfeita, a matriz base M_H deve ser igual a:

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.50}$$

A matriz M_H é utilizada para a determinação de x(t) sob as condições impostas no vetor G_{Hx}. O mesmo é válido para as direções y e z. Fazendo

$$B_H = T.\, M_H \tag{4.51}$$

e introduzindo este resultado em (4.43), obtém-se:

$$Q(t) = B_H \,.\, G_H = [\ B_H.G_{Hx} \quad B_H.G_{Hy} \quad B_H.G_{Hz}\ ] \tag{4.52}$$

onde B_H representa as funções de interpolação de Hermite:

$$B_H = [\ H_1 \quad H_2 \quad H_3 \quad H_4\ ] \tag{4.53}$$

Efetuando o produto em (4.51) obtém-se estas funções:

$$H_1 = 2\,t^3 - 3\,t^2 + 1 \tag{4.54}$$

$$H_2 = -2\,t^3 + 3\,t^2 \tag{4.55}$$

$$H_3 = t^3 - 2\,t^2 + t \tag{4.56}$$

$$H_4 = t^3 - t^2 \tag{4.57}$$

e expandindo o polinômio x(t) obtém-se:

$$x(t) = B_H \,.G_{Hx} = H_1 \,.P_{1x} + H_2 \,.P_{2x} + H_3 \,.R_{1x} + H_4 \,.\, R_{2x} \tag{4.58}$$

A Figura 4.7 mostra as funções de Hermite H_1, H_2, H_3 e H_4.

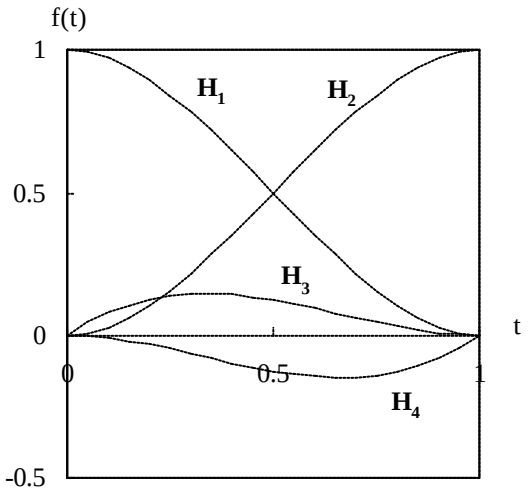


Figura 4.7 - Funções de interpolação cúbicas de Hermite.

Para que dois segmentos cúbicos de Hermite Q_1 e Q_2 partilhem um mesmo vértice com continuidade geométrica G^1 (Figura 4.8), é necessário que os vetores geométricos dos dois segmentos sejam iguais a:

$$G_H^1 = \begin{bmatrix} P_1 \\ P_2 \\ R_1 \\ R_2 \end{bmatrix} \tag{4.59}$$

$$G_H^2 = \begin{bmatrix} P_2 \\ P_3 \\ k \cdot R_2 \\ R_3 \end{bmatrix} \tag{4.60}$$

com $k > 0$. Se $k = 1$ tem-se continuidade C^1 .

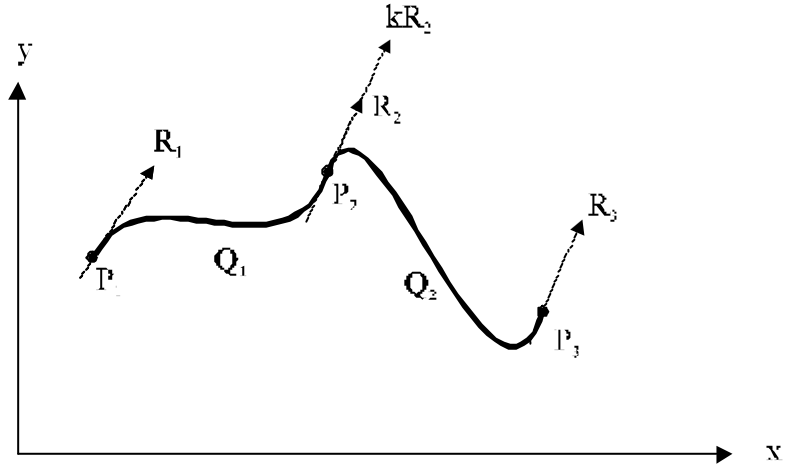


Figura 4.8 - Conexão de dois segmentos cúbicos de Hermite.

4.1.3 Curvas de Bézier

Os polinômios cúbicos de Bézier são especificados através dos pontos extremos e de dois pontos não pertencentes à curva que controlam as tangentes nas extremidades (Figura 4.9). Os vetores tangentes das extremidades são determinados pelos vetores P_1P_3 e P_4P_2 , que estão relacionados a R_1 e R_2 por:

$$R_1 = Q'(0) = 3\,(P_3 - P_1) \tag{4.61}$$

$$R_2 = Q'(1) = 3\,(P_2 - P_4) \tag{4.62}$$

A matriz geométrica de Bézier é dada por:

$$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} P_{1x} & P_{1y} & P_{1z} \\ P_{2x} & P_{2y} & P_{2z} \\ P_{3x} & P_{3y} & P_{3z} \\ P_{4x} & P_{4y} & P_{4z} \end{bmatrix} \tag{4.63}$$

e, em consequência, a matriz M_{HB} que transforma a matriz geométrica de Bézier na matriz geométrica de Hermite,

$$G_H = M_{HB} \,.\, G_B \tag{4.64}$$

é obtida da relação:

$$G_H = \begin{bmatrix} P_1 \\ P_2 \\ R_1 \\ R_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 \\ 0 & 3 & 0 & -3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} . G_B \tag{4.65}$$

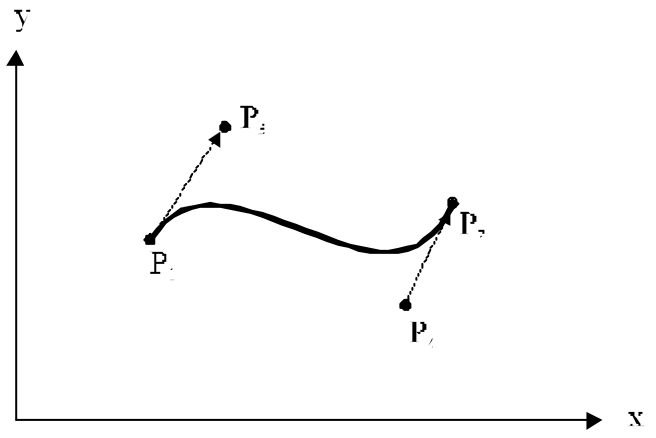


Figura 4.9 - Polinômio cúbico de Bézier.

Sendo assim, pode-se escrever:

$$Q(t) = T \cdot M_H \cdot G_H = T \cdot M_H \cdot M_{HB} \cdot G_B \tag{4.66}$$

e, portanto, a matriz base de Bézier é igual a:

$$M_B = M_H \cdot M_{HB} = \begin{bmatrix} -1 & 1 & 3 & -3 \\ 3 & 0 & -6 & 3 \\ -3 & 0 & 3 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.67}$$

A curva paramétrica de Bézier pode então ser escrita na forma:

$$Q(t) = T \cdot M_B \cdot G_B \tag{4.68}$$

ou

$$Q(t) = B_B \cdot G_B \tag{4.69}$$

onde B_B representa as funções de interpolação de Bézier:

$$B_B = T \cdot M_B = [\, B_1 \quad B_2 \quad B_3 \quad B_4 \,] \tag{4.70}$$

Estas funções são:

$$B_1 = (1- t)^3 \tag{4.71}$$

$$B_2 = t^3 \tag{4.72}$$

$$B_3 = 3t\,(1 - t)^2 \tag{4.73}$$

$$B_4 = 3t^2\,(1 - t) \tag{4.74}$$

Expandindo o produto $B_B \cdot G_B$, obtém-se:

$$Q(t) = B_1.P_1 + B_2.P_2 + B_3.P_3 + B_4.P_4 \tag{4.75}$$

As funções de interpolação em B_B são chamadas polinômios de Bernstein (Figura 4.10). Pode-se verificar que:

$$\begin{cases} B_i \geq 0 \\ \qquad\qquad (0 \leq t \leq 1) \\ B_1 + B_2 + B_3 + B_4 = 1 \end{cases} \tag{4.76}$$

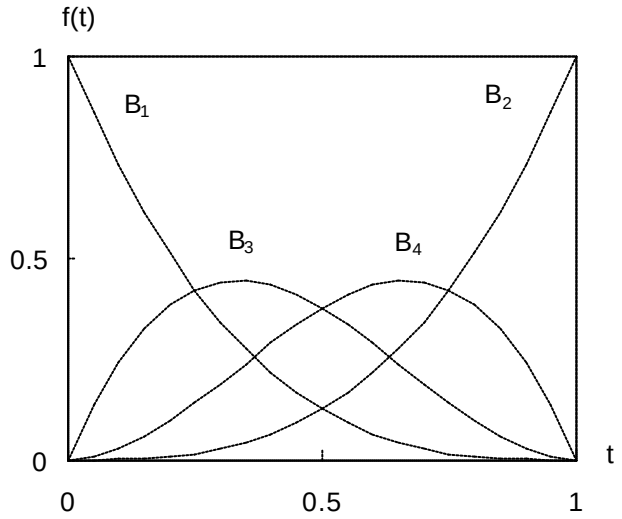


Figura 4.10 - Funções de interpolação de Bernstein.

Exercício

Seja um polinômio cúbico Q(t) em duas dimensões, com as condições P_1, P_2, $R_1 = \beta (P_3 - P_1)$ e $R_2 = \beta (P_2 - P_4)$. Sendo $P_1 = (0, 0)$, $P_2 = (3, 0)$, $P_3 = (1, 0)$ e $P_4 = (2, 0)$ (Figura 4.11), mostre que β deve ser igual a 3 para que a curva paramétrica Q(t) tenha velocidade constante.

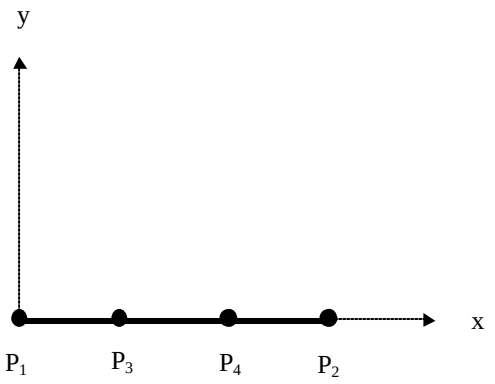


Figura 4.11 - Polinômio cúbico com os quatro pontos de controle sobre o eixo x.

Como os quatro pontos de controle se situam sobre o eixo x, as componentes y do polinômio e de sua derivada são nulas:

$$Q'(t) = [\ x'(t) \quad y'(t)\ ] = [\ x'(t) \quad 0\ ] \tag{4.77}$$

Escrevendo a função x'(t) na forma de Hermite obtém-se:

$$x'(t) = (6t^2 - 6t)\, P_{1x} + (-6t^2 + 6t)\, P_{2x} + (3t^2 - 4t + 1)\, R_{1x} + \\ + (3t^2 - 2t)\, R_{2x} \tag{4.78}$$

e substituindo nesta expressão as condições impostas,

$$P_{1x} = 0 \tag{4.79}$$

$$P_{2x} = 3 \tag{4.80}$$

$$R_{1x} = \beta\,(P_{3x} - P_{1x}) = \beta \tag{4.81}$$

$$R_{2x} = \beta\,(P_{2x} - P_{4x}) = \beta \tag{4.82}$$

chega-se ao resultado:

$$x'(t) = 3\,(-6t^2 + 6t) + \beta\,(3t^2 - 4t + 1) + \beta\,(3t^2 - 2t) = \\ = 3(-6t^2 + 6t) + \beta\,(6t^2 - 6t) + \beta = (6t^2-6t)\,(\beta-3) + \beta \tag{4.83}$$

Portanto, para que a expressão acima seja constante, β deve ser igual a 3.

4.1.4 B-splines

B-splines cúbicas aproximam uma série de m+3 pontos de controle, P_0, P_1, ..., P_{m+2}, por uma curva constituída de m segmentos polinomiais cúbicos, Q_0, Q_1,...,Q_{m-1} . Os pontos extremos de cada segmento Q_i (t), i = 0,..., m-1, são os pontos $Q_i(t_i)$ e $Q_i(t_{i+1})$, ou seja, o parâmetro t varia no intervalo $t_i \le t \le t_{i+1}$ em cada segmento (Figura 4.12).

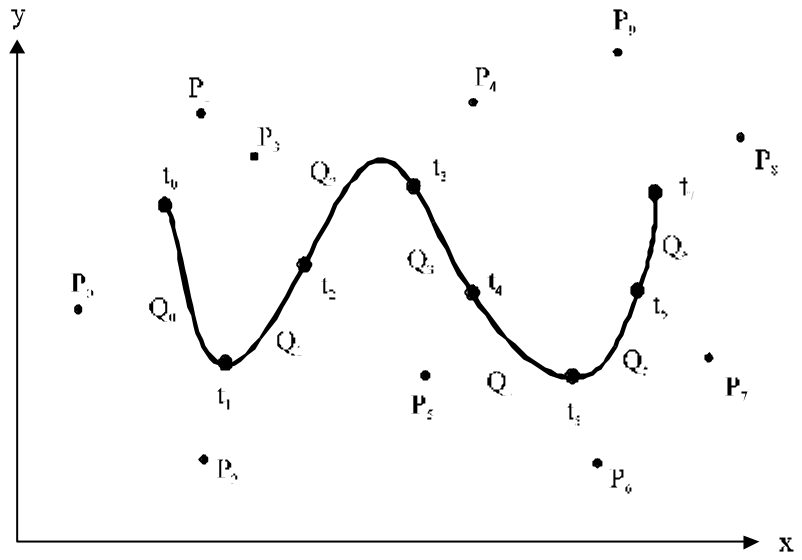


Figura 4.12 - B-spline cúbica com m = 7.

A letra B refere-se ao termo base, uma vez que a curva aproximada pode ser escrita na forma de uma combinação linear de funções de base. Se os intervalos $\Delta t_i = t_{i+1}-t_i$, são iguais para todos os segmentos, então a curva é dita uniforme. Pode-se assumir, por exemplo, $t_0 = 0$ e $\Delta t_i = 1$.

Cada segmento da curva é determinado por quatro pontos de controle. Em particular, o segmento Q_i é definido pelos pontos P_i, P_{i+1}, P_{i+2}, P_{i+3}. Consequentemente, o vetor geométrico de cada segmento Q_i é representado por:

$$G_{BS_i} = \begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix}, \; i = 0,\ldots,m-1 \tag{4.84}$$

A posição e os vetores tangentes das extremidades de cada segmento relacionam-se aos pontos de controle através das expressões:

$$Q_i(t_i) = (P_i + 4P_{i+1} + P_{i+2}) / 6 \tag{4.85}$$

$$Q_i(t_{i+1}) = (P_{i+1} + 4P_{i+2} + P_{i+3}) / 6 \tag{4.86}$$

$$Q_i'(t_i) = (P_{i+2} - P_i) / 2 \tag{4.87}$$

$$Q_i'(t_{i+1}) = (P_{i+3} - P_{i+1}) / 2 \tag{4.88}$$

Das expressões (4.85) e (4.86) pode-se concluir que, de modo geral, o segmento Q começa próximo ao ponto P_{i+1} e termina próximo ao ponto P_{i+2}. Assim como cada segmento é definido por quatro pontos de controle, cada ponto de controle (exceto P_0 e P_{m+2}) tem efeito sobre quatro segmentos de curva. O deslocamento de um ponto de controle em uma dada direção provoca movimento das quatro curvas sob a influência deste ponto na mesma direção. Os demais segmentos permanecem inalterados. Esta é uma característica que permite um controle local da curva. Definindo,

$$T_i = [\,(t - t_i)^3 \quad (t - t_i)^2 \quad (t - t_i) \quad 1\,] \tag{4.89}$$

cada segmento pode ser escrito na forma,

$$Q_i(t - t_i) = T_i \,.\, M_{BS} \,.\, G_{BS_i} \quad , \quad (t_i \le t \le t_{i+1}) \tag{4.90}$$

onde M_{BS} é a matriz base. Fazendo $t' = t - t_i$ e assumindo $\Delta t_i = 1$, a expressão acima pode ser reescrita na forma:

$$Q_i(t') = T' \,.\, M_{BS} \,.\, G_{BS_i} \quad , \quad (0 \le t' \le 1) \tag{4.91}$$

com o produto $T'.M_{BS}$ representando as funções de interpolação:

$$B_{BS} = T'.M_{BS} = [\,t'^3 \quad t'^2 \quad t' \quad 1\,]\, M_{BS} \tag{4.92}$$

Com a mudança de variável acima, a posição e os vetores tangentes das extremidades de cada segmento Q_i podem ser obtidos de acordo com:

$$Q_i(0) = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix} . M_{BS} \,.\, G_{BS_i} \tag{4.93}$$

$$Q_i(1) = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} . M_{BS} \,.\, G_{BS_i} \tag{4.94}$$

$$Q_i'(0) = \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix} . M_{BS} \,.\, G_{BS_i} \tag{4.95}$$

$$Q_i'(1) = \begin{bmatrix}3 & 2 & 1 & 0\end{bmatrix} . M_{BS} \,.\, G_{BS_i} \tag{4.96}$$

ou,

$$\begin{bmatrix} Q_i(0) \\ Q_i(1) \\ Q_i'(0) \\ Q_i'(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_{BS} . G_{BS_i} \tag{4.97}$$

Por outro lado, aplicando a mesma mudança de variável às equações (4.85)-(4.88), obtém-se:

$$\begin{bmatrix} Q_i(0) \\ Q_i(1) \\ Q_i'(0) \\ Q_i'(1) \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \end{bmatrix} G_{BS_i} \tag{4.98}$$

Comparando as equações (4.97) e (4.98), conclui-se que:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} M_{BS} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \end{bmatrix} \tag{4.99}$$

e obtém-se então a matriz base M_{BS} para uma B-spline uniforme:

$$M_{BS} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ -3 & 0 & 3 & 0 \\ 0 & -3 & 0 & 3 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{4.100}$$

Substituindo o resultado acima em (4.92), obtém-se as funções de base ou funções de interpolação:

$$B_{BS} = [\ BS_0 \quad BS_1 \quad BS_2 \quad BS_3\ ] \tag{4.101}$$

$$BS_0 = \frac{1}{6}(1-t')^3 \tag{4.102}$$

$$BS_1 = \frac{1}{6}(3t'^3 - 6t'^2 + 4) \tag{4.103}$$

$$BS_2 = \frac{1}{6}(-3t'^3 + 3t'^2 + 3t' + 1) \tag{4.104}$$

$$BS_3 = \frac{1}{6}t'^3 \tag{4.105}$$

Utilizando as expressões acima, cada segmento pode ser escrito na forma:

$$\begin{aligned} Q_i(t - t_i) &= B_{BS} . G_{BS_i} = \\ &= BS_0 . P_i + BS_1 . P_{i+1} + BS_2 . P_{i+2} + BS_3 . P_{i+3}, \quad (t_i \le t \le t_{i+1}) \end{aligned} \tag{4.106}$$

Observando as funções BS_i (Figura 4.13), pode-se verificar que:

$$\begin{cases} BS_i \ge 0 \\ \\ BS_0 + BS_1 + BS_2 + BS_3 = 1 \end{cases} \qquad (0 \le t' \le 1) \tag{4.107}$$

Isto significa que cada segmento $Q_i(t\text{-}t_i)$ deve estar contido no interior do polígono convexo formado pelos quatro pontos de controle (ver Figura 4.14). Se três pontos de controle são coincidentes, o polígono degenera-se em uma reta, e em consequência o segmento curvo terá que ser necessariamente também uma reta.

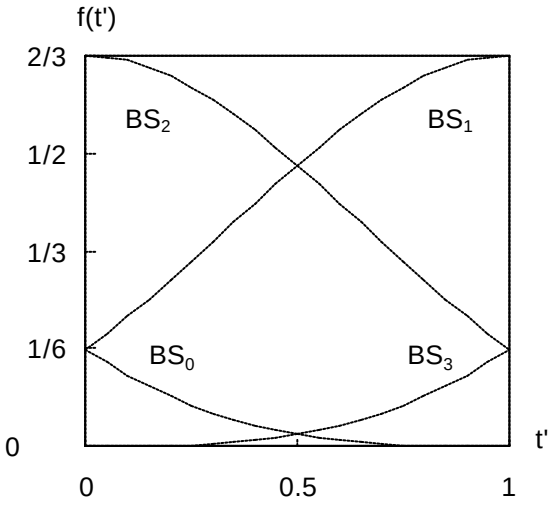


Figura 4.13 - Funções de base de uma B-spline uniforme cúbica.

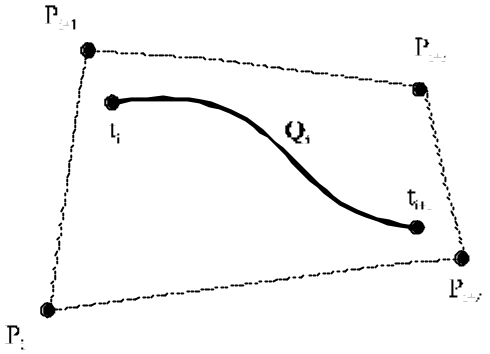


Figura 4.14 - Quadrilátero formado pelos quatro pontos de controle de uma B-spline cúbica.

Pode-se mostrar que os segmentos Q possuem continuidade C^2 nos pontos de conexão. Para isto, é suficiente mostrar que para dois segmentos adjacentes Q e Q_{i+1} (Figura 4.15) são válidas as relações (considerando-se, por simplicidade, somente a direção x):

$$x_i(t_{i+1}) = x_{i+1}(t_{i+1}) \tag{4.108}$$

$$\frac{d}{dt}x_i(t_{i+1}) = \frac{d}{dt}x_{i+1}(t_{i+1}) \tag{4.109}$$

$$\frac{d^2}{dt^2}x_i(t_{i+1}) = \frac{d^2}{dt^2}x_{i+1}(t_{i+1}) \tag{4.110}$$

ou equivalentemente (como as funções de interpolação são definidas no intervalo $0 \leq t' \leq 1$):

$$x_i\big|_{t'=1} = x_{i+1}\big|_{t'=0} \tag{4.111}$$

$$\frac{d}{dt}x_i\bigg|_{t'=1} = \frac{d}{dt}x_{i+1}\bigg|_{t'=0} \tag{4.112}$$

$$\frac{d^2}{dt^2}x_i\bigg|_{t'=1} = \frac{d^2}{dt^2}x_{i+1}\bigg|_{t'=0} \tag{4.113}$$

As derivadas das funções de interpolação são:

$$\frac{d}{dt}B_{BS} = \frac{1}{6}\left[-3(1-t')^2 \quad (9t'^2-12t') \quad (-9t'^2+6t'+3) \quad 3t'^2\right] \quad (4.114)$$

$$\frac{d^2}{dt^2}B_{BS} = \frac{1}{6}\left[6(1-t') \quad (18t'-12) \quad (-18t'+6) \quad 6t'\right] \quad (4.115)$$

e para t' = 0 e t' = 1 pode-se escrever:

$$B_{BS}\Big|_{t'=0} = \left[1/6 \quad 2/3 \quad 1/6 \quad 0\right] \quad (4.116)$$

$$B_{BS}\Big|_{t'=1} = \left[0 \quad 1/6 \quad 2/3 \quad 1/6\right] \quad (4.117)$$

$$\frac{d}{dt}B_{BS}\Big|_{t'=0} = \left[-1/2 \quad 0 \quad 1/2 \quad 0\right] \quad (4.118)$$

$$\frac{d}{dt}B_{BS}\Big|_{t'=1} = \left[0 \quad -1/2 \quad 0 \quad 1/2\right] \quad (4.119)$$

$$\frac{d^2}{dt^2}B_{BS}\Big|_{t'=0} = \left[1 \quad -2 \quad 1 \quad 0\right] \quad (4.120)$$

$$\frac{d^2}{dt^2}B_{BS}\Big|_{t'=1} = \left[0 \quad -1 \quad -2 \quad 0\right] \quad (4.121)$$

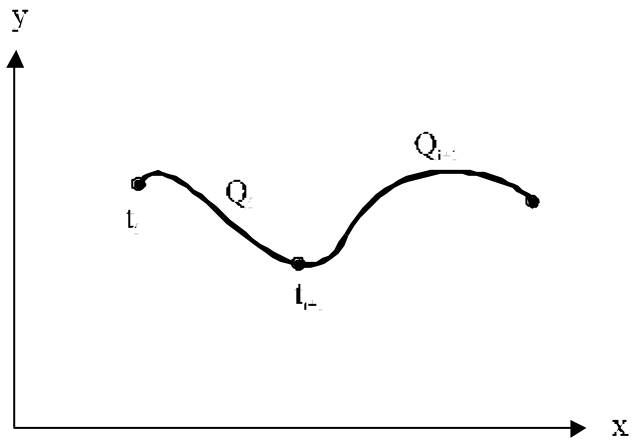


Figura 4.15 - Dois segmentos de curva adjacentes Q_i e Q_{i+1} .

Sendo x_i e x_{i+1} dados por:

$$x_i = B_{BS}.G_{BS_i} = B_{BS}\begin{bmatrix} P_{i,x} \\ P_{i+1,x} \\ P_{i+2,x} \\ P_{i+3,x} \end{bmatrix} \tag{4.122}$$

$$x_{i+1} = B_{BS}.G_{BS_{i+1}} = B_{BS}\begin{bmatrix} P_{i+1,x} \\ P_{i+2,x} \\ P_{i+3,x} \\ P_{i+4,x} \end{bmatrix} \tag{4.123}$$

pode-se escrever:

$$x_i\big|_{t'=1} = B_{BS}\big|_{t'=1}.G_{BS_i} = \frac{1}{6}P_{i+1,x} + \frac{2}{3}P_{i+2,x} + \frac{1}{6}P_{i+3,x} \tag{4.124}$$

$$x_{i+1}\big|_{t'=0} = B_{BS}\big|_{t'=0}.G_{BS_{i+1}} = \frac{1}{6}P_{i+1,x} + \frac{2}{3}P_{i+2,x} + \frac{1}{6}P_{i+3,x} \tag{4.125}$$

$$\frac{d}{dt}x_i\bigg|_{t'=1} = \frac{d}{dt}B_{BS}\bigg|_{t'=1}.G_{BS_i} = \frac{1}{2}(P_{i+3,x} - P_{i+1,x}) \tag{4.126}$$

$$\frac{d}{dt}x_{i+1}\bigg|_{t'=0} = \frac{d}{dt}B_{BS}\bigg|_{t'=0}.G_{BS_{i+1}} = \frac{1}{2}(P_{i+3,x} - P_{i+1,x}) \tag{4.127}$$

$$\frac{d^2}{dt^2}x_i\bigg|_{t'=1} = \frac{d^2}{dt^2}B_{BS}\bigg|_{t'=1}.G_{BS_i} = P_{i+1,x} - 2P_{i+2,x} + P_{i+3,x} \tag{4.128}$$

$$\frac{d^2}{dt^2}x_{i+1}\bigg|_{t'=0} = \frac{d^2}{dt^2}B_{BS}\bigg|_{t'=0}.G_{BS_{i+1}} = P_{i+1,x} - 2P_{i+2,x} + P_{i+3,x} \tag{4.129}$$

demonstrando-se a continuidade C^2 nos pontos de conexão dos segmentos.

4.2 SUPERFÍCIES PARAMÉTRICAS

Assim como as curvas, uma superfície Q pode também ser representada de forma paramétrica (Figura 4.16):

$$Q(s,t) = [\, x(s, t) \quad y(s, t) \quad z(s, t) \,] \tag{4.130}$$

Matricialmente, uma superfície cúbica é representada por:

$$x(s, t) = S \cdot C_x \cdot T^t \tag{4.131}$$

$$y(s, t) = S \cdot C_y \cdot T^t \tag{4.132}$$

$$z(s, t) = S \cdot C_z \cdot T^t \tag{4.133}$$

onde C_x , C_y e C_z são as matrizes 4x4 de coeficientes nas direções x, y e z, respectivamente, e S e T são iguais a:

$$S = [\, s^3 \quad s^2 \quad s \quad 1 \,] \tag{4.134}$$

$$T = [\, t^3 \quad t^2 \quad t \quad 1 \,] \tag{4.135}$$

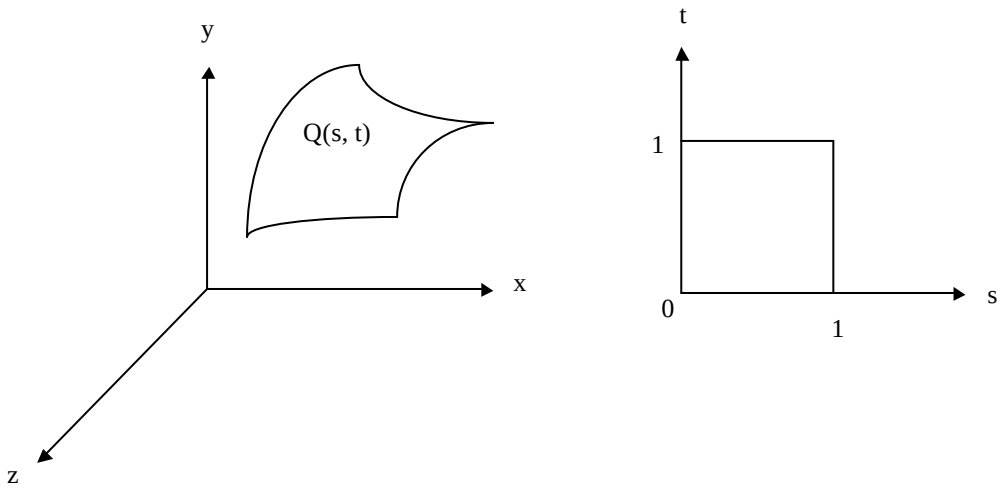


Figura 4.16 - Superfície paramétrica.

4.2.1 Superfícies de Lagrange

Uma superfície cúbica de Lagrange é definida por 16 pontos de controle pertencentes à superfície, como mostra a Figura 4.17, sendo descrita pelas expressões:

$$x(s, t) = (S.M_L).G_{Lx}.(T.M_L)^t \tag{4.136}$$

$$y(s, t) = (S.M_L).G_{Ly}.(T.M_L)^t \quad (0 \leq s \leq 1), (0 \leq t \leq 1) \tag{4.137}$$

$$z(s, t) = (S.M_L).G_{Lz}.(T.M_L)^t \tag{4.138}$$

onde M_L é a matriz base de Lagrange obtida em (4.26), e G_{Lx} , G_{Ly} , G_{Lz} são as matrizes geométricas nas direções x, y e z:

$$G_{Lx} = \begin{bmatrix} P_{11,x} & P_{12,x} & P_{13,x} & P_{14,x} \\ P_{21,x} & P_{22,x} & P_{23,x} & P_{24,x} \\ P_{31,x} & P_{32,x} & P_{33,x} & P_{34,x} \\ P_{41,x} & P_{42,x} & P_{43,x} & P_{44,x} \end{bmatrix} \tag{4.139}$$

$$G_{Ly} = \begin{bmatrix} P_{11,y} & P_{12,y} & P_{13,y} & P_{14,y} \\ P_{21,y} & P_{22,y} & P_{23,y} & P_{24,y} \\ P_{31,y} & P_{32,y} & P_{33,y} & P_{34,y} \\ P_{41,y} & P_{42,y} & P_{43,y} & P_{44,y} \end{bmatrix} \tag{4.140}$$

$$G_{Lz} = \begin{bmatrix} P_{11,z} & P_{12,z} & P_{13,z} & P_{14,z} \\ P_{21,z} & P_{22,z} & P_{23,z} & P_{24,z} \\ P_{31,z} & P_{32,z} & P_{33,z} & P_{34,z} \\ P_{41,z} & P_{42,z} & P_{43,z} & P_{44,z} \end{bmatrix} \tag{4.141}$$

Os produtos $(S.M_L)$ e $(T.M_L)$ das equações acima representam as funções unidimensionais cúbicas de Lagrange nas direções s e t, respectivamente:

$$S.M_L = B_L(s) = [\, L_1(s) \quad L_2(s) \quad L_3(s) \quad L_4(s) \,] \tag{4.142}$$

$$T.M_L = B_L(t) = [\, L_1(t) \quad L_2(t) \quad L_3(t) \quad L_4(t) \,] \tag{4.143}$$

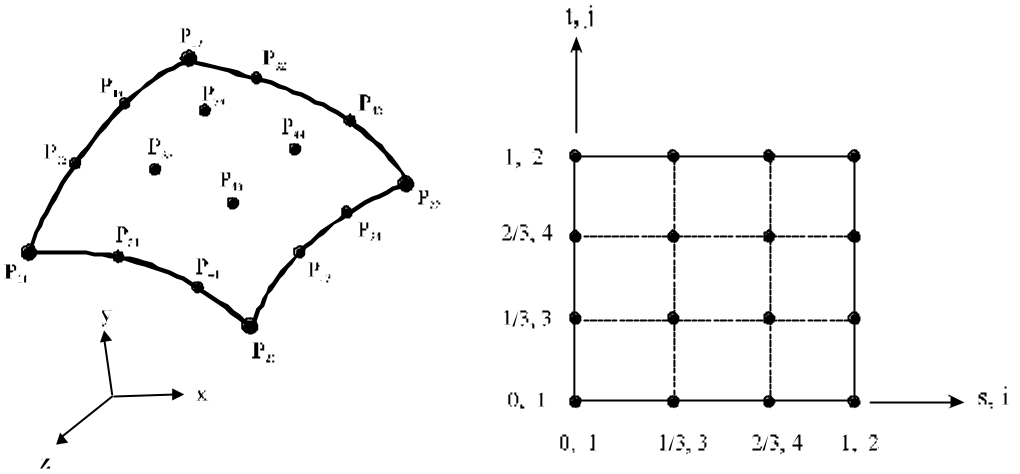


Figura 4.17 - Pontos de controle de uma superfície cúbica de Lagrange.

Da substituição destas expressões em (4.136)-(4.138) resultam:

$$x(t) = B_L(s).G_{Lx}.B_L{}^t(t) \tag{4.144}$$

$$y(t) = B_L(s).G_{Ly}.B_L{}^t(t) \tag{4.145}$$

$$z(t) = B_L(s).G_{Lz}.B_L{}^t(t) \tag{4.146}$$

e efetuando os produtos, obtém-se:

$$x(t) = \sum_{j=1}^{4}\sum_{i=1}^{4} L_i(s).L_j(t).P_{ij,x} \tag{4.147}$$

$$y(t) = \sum_{j=1}^{4}\sum_{i=1}^{4} L_i(s).L_j(t).P_{ij,y} \tag{4.148}$$

$$z(t) = \sum_{j=1}^{4}\sum_{i=1}^{4} L_i(s).L_j(t).P_{ij,z} \tag{4.149}$$

Portanto, pode-se concluir que as funções de interpolação bidimensionais de Lagrange $\varphi_{ij}(s,t)$, correspondentes aos pontos P_{ij}, são iguais aos produtos das funções unidimensionais nas direções s e t:

$$\varphi_{ij}(s,t) = L_i(s).L_j(t) \tag{4.150}$$

O mesmo raciocínio é válido para uma superfície de grau p, com $(p+1)^2$ pontos de controle.

4.2.2 Superfícies cúbicas de Hermite

Nas superfícies cúbicas de Hermite, as componentes x, y e z são representadas pelas expressões:

$$x(s,t) = S.M_H.G_{Hx}.M_H^t.T^t \tag{4.151}$$

$$y(s,t) = S.M_H.G_{Hy}.M_H^t.T^t \tag{4.152}$$

$$z(s,t) = S.M_H.G_{Hz}.M_H^t.T^t \tag{4.153}$$

onde M_H é a matriz base de Hermite da equação (4.50), e G_{Hx} , G_{Hy} e G_{Hz} são as matrizes geométricas para x, y e z:

$$G_{Hx} = \begin{bmatrix} x(0,0) & x(0,1) & \frac{\partial}{\partial t}x(0,0) & \frac{\partial}{\partial t}x(0,1) \\ x(1,0) & x(1,1) & \frac{\partial}{\partial t}x(1,0) & \frac{\partial}{\partial s}x(1,1) \\ \frac{\partial}{\partial s}x(0,0) & \frac{\partial}{\partial s}x(0,1) & \frac{\partial^2}{\partial s \partial t}x(0,0) & \frac{\partial^2}{\partial s \partial t}x(0,1) \\ \frac{\partial}{\partial s}x(1,0) & \frac{\partial}{\partial s}x(1,1) & \frac{\partial^2}{\partial s \partial t}x(1,0) & \frac{\partial^2}{\partial s \partial t}x(1,1) \end{bmatrix} \tag{4.154}$$

$$G_{Hy} = \begin{bmatrix} y(0,0) & y(0,1) & \frac{\partial}{\partial t}y(0,0) & \frac{\partial}{\partial t}y(0,1) \\ y(1,0) & y(1,1) & \frac{\partial}{\partial t}y(1,0) & \frac{\partial}{\partial s}y(1,1) \\ \frac{\partial}{\partial s}y(0,0) & \frac{\partial}{\partial s}y(0,1) & \frac{\partial^2}{\partial s \partial t}y(0,0) & \frac{\partial^2}{\partial s \partial t}y(0,1) \\ \frac{\partial}{\partial s}y(1,0) & \frac{\partial}{\partial s}y(1,1) & \frac{\partial^2}{\partial s \partial t}y(1,0) & \frac{\partial^2}{\partial s \partial t}y(1,1) \end{bmatrix} \quad (4.155)$$

$$G_{Hz} = \begin{bmatrix} z(0,0) & z(0,1) & \frac{\partial}{\partial t}z(0,0) & \frac{\partial}{\partial t}z(0,1) \\ z(1,0) & z(1,1) & \frac{\partial}{\partial t}z(1,0) & \frac{\partial}{\partial s}z(1,1) \\ \frac{\partial}{\partial s}z(0,0) & \frac{\partial}{\partial s}z(0,1) & \frac{\partial^2}{\partial s \partial t}z(0,0) & \frac{\partial^2}{\partial s \partial t}z(0,1) \\ \frac{\partial}{\partial s}z(1,0) & \frac{\partial}{\partial s}z(1,1) & \frac{\partial^2}{\partial s \partial t}z(1,0) & \frac{\partial^2}{\partial s \partial t}z(1,1) \end{bmatrix} \quad (4.156)$$

4.2.3 Superfícies de Bézier

Da mesma forma que as superfícies de Hermite, as superfícies de Bézier são obtidas através de expressões do tipo:

$$x(s,t) = S.M_B.G_{Bx}.M_B^t.T^t \quad (4.157)$$

$$y(s,t) = S.M_B.G_{By}.M_B^t.T^t \quad (4.158)$$

$$z(s,t) = S.M_B.G_{Bz}.M_B^t.T^t \quad (4.159)$$

onde M_B é a matriz base de (4.67), e as matrizes geométricas G_{Bx} G_{By} e G_{Bz} contém os 16 pontos de controle da superfície (Figura 4.18):

$$G_{Bx} = \begin{bmatrix} P_{11,x} & P_{12,x} & P_{13,x} & P_{14,x} \\ P_{21,x} & P_{22,x} & P_{23,x} & P_{24,x} \\ P_{31,x} & P_{32,x} & P_{33,x} & P_{34,x} \\ P_{41,x} & P_{42,x} & P_{43,x} & P_{44,x} \end{bmatrix} \quad (4.160)$$

$$G_{By} = \begin{bmatrix} P_{11,y} & P_{12,y} & P_{13,y} & P_{14,y} \\ P_{21,y} & P_{22,y} & P_{23,y} & P_{24,y} \\ P_{31,y} & P_{32,y} & P_{33,y} & P_{34,y} \\ P_{41,y} & P_{42,y} & P_{43,y} & P_{44,y} \end{bmatrix} \tag{4.161}$$

$$G_{Bz} = \begin{bmatrix} P_{11,z} & P_{12,z} & P_{13,z} & P_{14,z} \\ P_{21,z} & P_{22,z} & P_{23,z} & P_{24,z} \\ P_{31,z} & P_{32,z} & P_{33,z} & P_{34,z} \\ P_{41,z} & P_{42,z} & P_{43,z} & P_{44,z} \end{bmatrix} \tag{4.162}$$

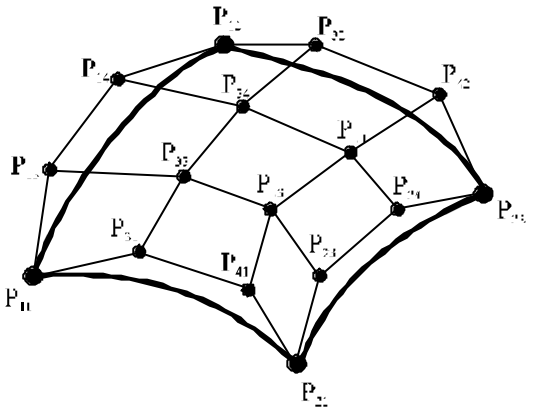


Figura 4.18 - Pontos de controle de uma superfície cúbica de Bézier.

4.3 SUPERFÍCIES POLIÉDRICAS

Uma superfície poliédrica é representada por um conjunto de arestas, vértices e polígonos. Os vértices estão ligados por arestas e os polígonos, designados faces, são sequências fechadas de arestas pertencentes a um mesmo plano. A seguir são examinados três métodos de representação de uma superfície ou malha poliédrica:

- lista explícita de vértices

- lista de polígonos

- lista explícita de arestas

Na lista explícita de vértices, cada polígono é representado por uma lista de coordenadas dos vértices:

$$P = [\ (x_1 , y_1 , z_1), (x_2 , y_2 , z_2), \ldots\ldots , (x_n , y_n , z_n)\ ] \tag{4.163}$$

Os vértices são armazenados em um sentido pré-estabelecido. Por exemplo, pode-se estabelecer um sentido que identifique a face externa da superfície, como mostra mostra a Figura 4.19. Apesar de ser funcional para um polígono isolado, esta forma de representação é bastante ineficiente para uma malha completa, em que vértices partilhados são repetidos várias vezes.

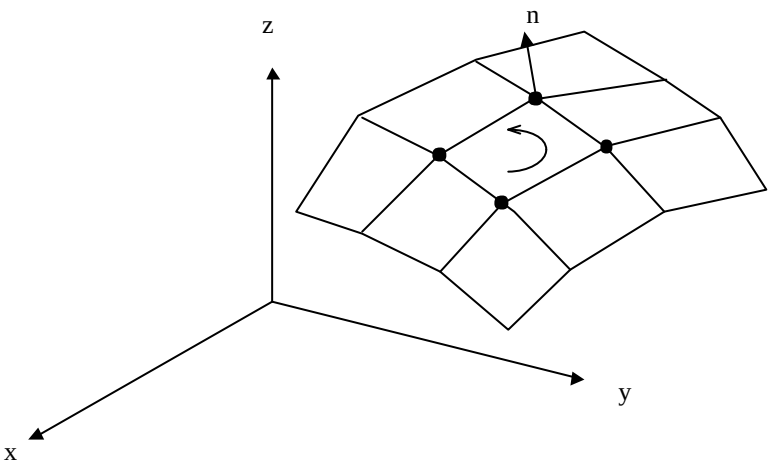


Figura 4.19 - Sentido de numeração de um polígono, identificando o sentido da normal externa à superfície.

Na lista de polígonos, cada vértice é armazenado em uma lista V de vértices:

$$V = (V_1 , V_2 , ... , V_n) \tag{4.164}$$

$$V_i = (x_i , y_i , z_i) \tag{4.165}$$

Cada polígono é definido por uma lista de índices apontadores da lista de vértices (Figura 4.20). Neste método, as coordenadas de cada vértice são armazenadas apenas uma vez, e podem ser modificadas facilmente. No entanto, algumas dificuldades persistem, como por exemplo, o problema de traçar duas vezes arestas partilhadas.

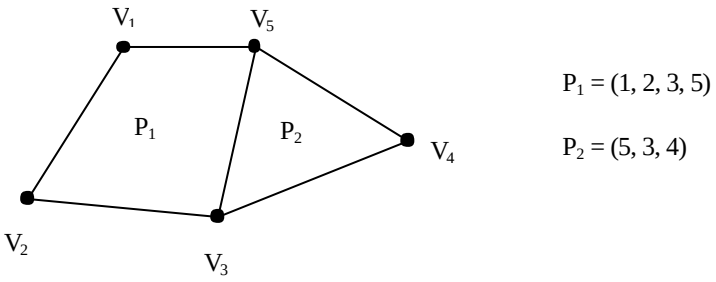


Figura 4.20 - Lista de polígonos.

No método da lista explícita de arestas são mantidas uma lista V de vértices e uma lista E de arestas:

$$V = (V_1 , V_2 , ... , V_n) \tag{4.166}$$

$$E = (E_1 , E_2 , ... , E_m) \tag{4.167}$$

Cada elemento da lista de arestas contém dois apontadores para a lista de vértices, que representam suas extremidades, e também k apontadores para a lista de polígonos, que indicam o polígono ou polígonos que contém a aresta:

$$E_i = (v_1 , v_2 , p_1 , ... , p_k) \tag{4.168}$$

Sendo e_i apontadores da lista de arestas, um polígono com q arestas é definido por:

$$P_i = (e_1 , ..., e_q) \tag{4.169}$$

Superfícies poliédricas podem ser geradas ou obtidas a partir de superfícies paramétricas de grau superior. A subdivisão da superfície é feita no domínio regular definido pelos parâmetros s e t, onde a topologia da malha, ou seja, as relações de conetividade entre vértices, arestas e polígonos, é facilmente determinada. Em seguida, a malha criada no domínio regular é mapeada para o espaço tridimensional, através das equações paramétricas da superfície.

5 ELIMINAÇÃO DE SUPERFÍCIES OCULTAS

A determinação da visibilidade de um objeto consiste em eliminar as partes de sua superfície que são ocultas por outras, sob o ponto de vista do observador. Este processo é conhecido por *eliminação de superfícies ocultas*, ou *eliminação de linhas ocultas*, se as linhas são consideradas como sendo as arestas das superfícies.

Os algoritmos de eliminação de superfícies ocultas classificam-se em duas categorias principais. Na primeira categoria estão os algoritmos que trabalham diretamente com o objeto (*object-precision algorithms*), sem levar em conta a sua imagem representada por pixels. Nesta categoria inclui-se o *algoritmo de prioridades*. Estes algoritmos operam, convenientemente, sobre o volume de visualização normalizado, após as operações de recorte terem sido realizadas. A tarefa destes algoritmos é geométrica, e consiste em determinar as partes do objeto visíveis ao observador, levando em conta as profundidades das partes e verificando suas interseções.

A segunda categoria compreende os métodos que trabalham diretamente com a imagem convertida do objeto (*image-precision algorithms*). A idéia destes algoritmos é determinar, para cada pixel da tela, quais os pontos do espaço normalizado que, após transformados em coordenadas de dispositivo, produzem a sua conversão. Dentre estes, o mais próximo ao observador é escolhido, e o pixel é convertido com a cor de sua primitiva de origem. Os algoritmos *Z-buffer* e *ray-tracing* são exemplos de algoritmos pertencentes a esta categoria.

5.1 ALGORITMO DE PRIORIDADES

A idéia básica do algoritmo de prioridades é ordenar as primitivas de acordo com suas profundidades z, ou distâncias em relação ao ponto de observação. As primitivas são convertidas na ordem decrescente de suas profundidades, fazendo com que as mais próximas ao observador se sobreponham às mais distantes. Considerando o modelo de visualização do item 3.3, a ordenação de profundidades é feita no sistema de coordenadas normalizadas r-s-t, tendo em conta que a distância de um ponto ao observador é (1- t) na projeção perspectiva e (∞ - t) na projeção paralela. Portanto, as primitivas devem ser ordenadas no sentido crescente de t.

Na versão mais simples deste algoritmo, denominada *algoritmo do pintor*, as primitivas são ordenadas de acordo com as profundidades de seus centros geométricos ou de acordo com seus valores mínimos de z (z_{min}). Para alguns casos, esta versão simplificada produz bons resultados, como por exemplo no caso de superfícies suaves (superfícies sem grandes variações locais da normal à superfície), aproximadas por faces planas. Entretanto,

em geral a ordenação pelo centro geométrico ou por z_{min} não é suficiente para determinar a visibilidade de uma primitiva. O algoritmo descrito a seguir, desenvolvido por Newell, Newell e Sancha [9], é aplicável a polígonos planos e realiza, além da ordenação inicial, alguns testes para determinar a visibilidade de uma primitiva.

Seja um conjunto de polígonos ordenados em ordem decrescente de acordo com suas profundidades mínimas. A posição inicial do arranjo ordenado aponta para o polígono P, sendo este, portanto, o polígono mais distante do observador. Antes de ser convertido, este polígono deve ser comparado com todos os polígonos Q cujas extensões, na direção z, se sobreponham à extensão de P, como é o caso da Figura 5.1. Se não há interseção entre as extensões em z de P e Q, pode-se afirmar que o polígono Q não é oculto por P. Caso contrário, cinco testes adicionais, comparando P e Q, devem ser realizados nesta ordem:

1. A interseção entre as extensões de P e Q na direção x é nula ?

2. A interseção entre as extensões de P e Q na direção y é nula ?

3. P está inteiramente situado do lado oposto do plano de Q, em relação ao observador (Figura 5.2a) ?

4. Q está inteiramente situado entre o observador e o plano de P (Figura 5.2b)?

5. A interseção entre as projeções de P e Q no plano de projeção é nula ?

Se algum destes testes for positivo, os demais não são realizados e pode-se afirmar que o polígono P não interfere na visibilidade de Q. Se os 5 testes forem negativos assume-se que Q é oculto por P e os testes 3 e 4 são aplicados novamente, trocando-se P por Q:

3. Q está inteiramente situado do lado oposto do plano de P, em relação ao observador ?

4. P está inteiramente situado entre o observador e o plano de Q ?

Se algum dos testes acima for positivo, como ocorre no exemplo da Figura 5.3a, então Q é movido para o início da lista ordenada e o algoritmo prossegue com Q assumindo o lugar de P. Em caso contrário (Figura 5.3b), deve-se particionar um dos polígonos utilizando o plano do outro como plano de recorte. As novas partes são inseridas na lista ordenada nas posições apropriadas, conforme suas profundidades mínimas, e o algoritmo prossegue normalmente.

Os polígonos que são movidos para o início da lista devem ser marcados de modo a evitar um loop infinito, como ocorreria no caso da Figura 5.4. Neste exemplo, é impossível estabeler uma ordem correta de visibilidade. Assim, quando um polígono P marcado não passar nos cinco testes, quando comparado com algum polígono Q, os testes 3 e 4 não devem ser repetidos. Quando esta situação ocorre, procede-se como se os testes 3 e 4 fossem ambos negativos, particionando-se um dos polígonos.

A implementação dos testes para verificar a interseção das extensões de dois polígonos nas direções x, y e z é trivial. Os testes 3 e 4 consistem em saber se um determinado polígono se situa em um mesmo lado de um plano. Isto pode ser feito aplicando-se a expressão (3.84), que considera a equação de um plano e a posição de um ponto, para cada vértice do polígono. No teste 5, as arestas de um polígono são comparadas com todas as arestas do outro, no plano de projeção. Se houver cruzamento de arestas, então as projeções dos polígonos se sobrepoem.

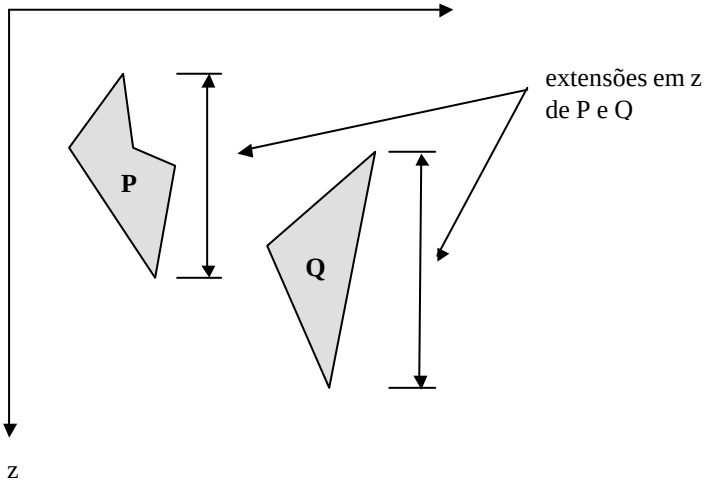


Figura 5.1 - Extensões em z de dois polígonos P e Q.

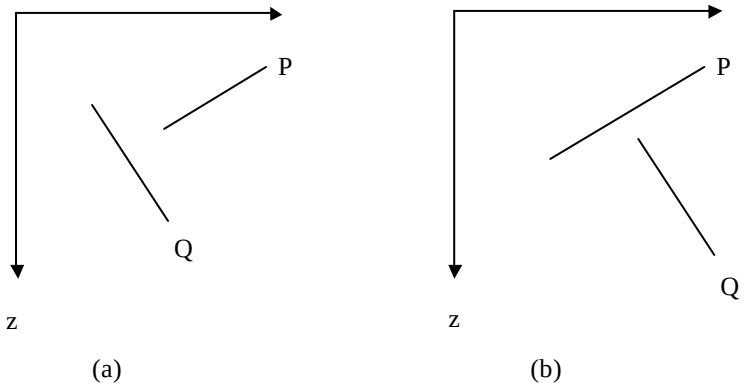


Figura 5.2 - (a) P inteiramente no lado oposto do plano de Q, em relação ao observador. (b) Q inteiramente entre o observador e o plano de P.

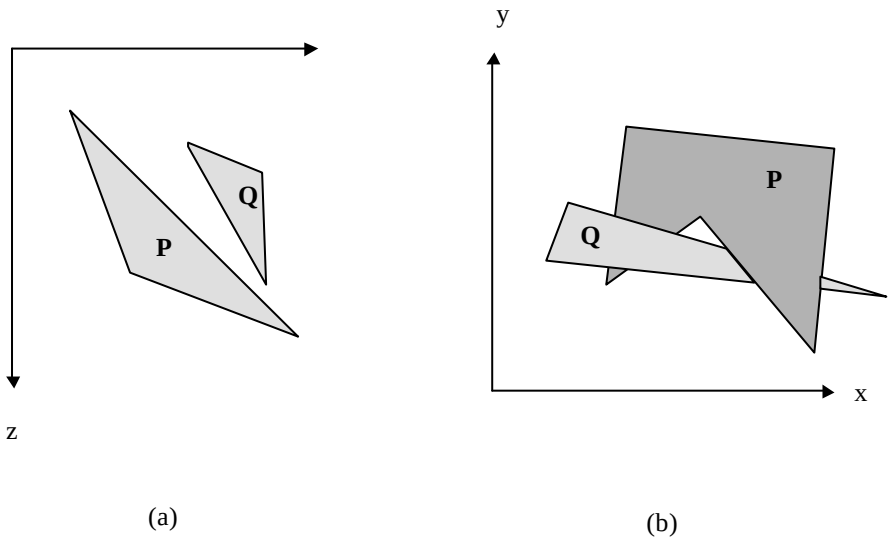


Figura 5.3- (a) Q situado inteiramente no lado oposto de P, em relação ao observador. (b) Dois polígonos P e Q, um interferindo na visibilidade do outro.

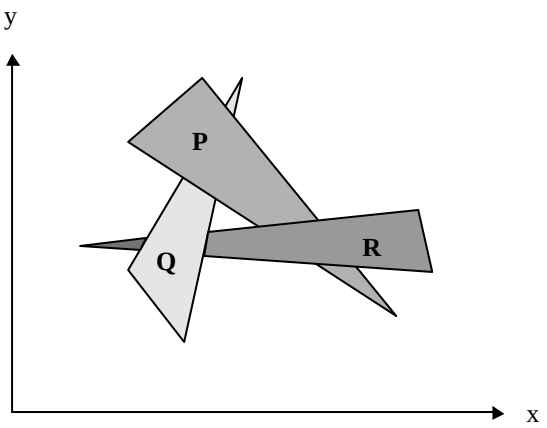


Figura 5.4 - Entrelaçamento de polígonos.

5.2 ALGORITMO Z-BUFFER

Para implementar o algoritmo Z-buffer [14] é necessário dispor de um arranjo bidimensional, denominado Z-buffer, onde cada elemento Z-buffer(i, j) corresponde ao pixel de coordenadas (i, j) no dispositivo gráfico. Portanto, para um dispositivo de NxM pixels, necessita-se de um arranjo com NxM elementos.

Inicialmente, a profundidade correspondente ao plano posterior de recorte (t=0) é atribuída a todos os elementos do arranjo Z-buffer. Durante a fase de conversão o algoritmo calcula, para cada pixel a ser convertido, a profundidade do ponto no espaço de coordenadas normalizadas correspondente a este pixel Se esta for menor do que a profundidade corrente, então o pixel é convertido com a cor de sua primitiva, e o elemento do arranjo Z-buffer é atualizado com a nova profundidade, como mostram as linhas de código abaixo:

```
if ( z(i, j)  <  Z-buffer(i, j) )
{
        writepixel(i, j, color);
        Z-buffer(i, j) = z(i, j);
}
```

As funções de conversão específicas de cada primitiva devem ser modificadas de modo que a profundidade de cada ponto convertido seja calculada. Assim, este algoritmo pode ser aplicado a qualquer primitiva, desde que as profundidades de todos os pontos possam ser determinadas. Outra vantagem deste método é que as primitivas podem ser convertidas em qualquer ordem, não sendo necessária nenhuma ordenação. Por outro lado, para representar uma matriz de 800 x 600 pixels, são necessários 800 x 600 x P bytes para armazenar as profundidades, sendo P o número de bytes do tipo de variável do arranjo Z-buffer. Utilizando reais de 4 bytes (P = 4) são necessários 1.831 Mb de memória.

5.3 RAY-TRACING

Neste método [15-19], a janela de visualização definida no plano de projeção é dividida em uma malha regular, onde cada célula corresponde a um pixel da tela (Figura 5.5). Para cada célula obtém-se as equações paramétricas da reta que passa pelo seu centro e pelo centro de projeção, ou da linha de projeção que passa pelo centro da célula. O pixel correspondente a esta célula terá a cor da primitiva mais próxima do observador (centro de projeção) que intercepta a reta.

Os cálculos efetuados por este algoritmo resumem-se aos cálculos de interseções de retas com as primitivas. Assumindo que o centro de projeção tenha coordenadas (x_0, y_0, z_0) e que o centro de uma célula seja dado por (x_1, y_1, z_1), as equações paramétricas da reta são:

$$x = x_0 + t\,\Delta x \tag{5.1}$$

$$y = y_0 + t\,\Delta y \tag{5.2}$$

$$z = z_0 + t\,\Delta z \tag{5.3}$$

sendo Δx, Δy e Δz iguais a:

$$\Delta x = x_1 - x_0 \tag{5.4}$$

$$\Delta y = y_1 - y_0 \tag{5.5}$$

$$\Delta z = z_1 - z_0 \tag{5.6}$$

O parâmetro t varia no intervalo [0, 1] para pontos situados entre o centro de projeção e o plano de projeção, assumindo valores maiores que 1 para pontos mais distantes do que o plano de projeção. Portanto, para determinar o ponto de interseção mais próximo do observador, basta saber qual é a interseção com menor valor de t. Para calcular a interseção das linhas de projeção com polígonos, deve-se primeiro saber se a reta intercepta o plano que contém o polígono. Sendo (x_p, y_p, z_p) as coordenadas de um dos vértices do polígono, e (n_x, n_y, n_z) os cossenos diretores de sua normal, o valor de t para o ponto de interseção é:

$$t = \frac{(x_p - x_0).n_x + (y_p - y_0).n_y + (z_p - z_0).n_z}{\Delta x.n_x + \Delta y.n_y + \Delta z.n_z} \tag{5.7}$$

A reta só não intercepta o plano do polígono se o denominador da equação acima for igual a zero.

Se a reta intercepta o plano, o próximo passo é verificar se o ponto de interseção pertence ao interior do polígono. Isto é feito projetando ortograficamente o ponto e o polígono na direção do eixo principal que corresponde ao cosseno diretor da equação do plano de maior valor absoluto. O plano normal a este eixo é o que terá a maior projeção do polígono. Em consequência, as projeções do ponto e do polígono são obtidas ignorando-se

a coordenada correspondente ao cosseno diretor de maior valor absoluto. Com as projeções obtidas, utiliza-se um algoritmo adequado para determinar se um ponto pertence ao interior de um polígono. Para um polígono convexo, um ponto pertence ao seu interior se estiver situado à esquerda de todas as arestas, quando os vértices são percorridos no sentido anti-horário.

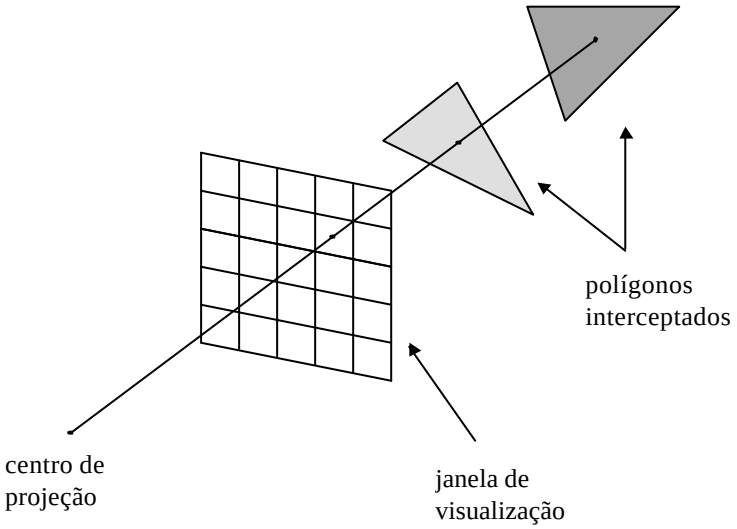


Figura 1.5 - Janela de visualização transformada em uma malha regular correspondente à matriz de pixels da tela.

6 REFERÊNCIAS

1. Foley, J. D., van Dam, J., Feiner, S. K. and Hughes, J. F., *Computer Graphics - Principles and Practice*, 2nd edition, Addison-Wesley, 1992.

2. Rogers, D. F., *Procedural Elements for Computer Graphics*, McGraw-Hill, 1988.

3. Bresenham, J. E., "Algorithm for Computer Control of a Digital Plotter", *IBM System Journal*, vol. 4(1), pp. 25-30, 1965.

4. Stollnitz, E. J., DeRose, T. D. and Salesin, D. H., "Wavelets for Computer Graphics: a Primer, Part 1", *IEEE Computer Graphics and Applications (CG & A)*, vol. 15(3), pp. 76-83, 1995.

5. Stollnitz, E. J., DeRose, T. D. and Salesin, D. H., "Wavelets for Computer Graphics: a Primer, Part 2", *CG & A*, vol. 15(4), pp. 75-85, 1995.

6. Sutherland, I. E. and Hodgman, G. W., "Reentrant Polygon Clipping", *Communications of the Association for Computing Machinery (CACM)*, vol. 17(1), pp. 32-42, 1974.

7. Maxwell, E. A., *Methods of Plane Projective Geometry Based on the use of General Homogeneous Coordinates*, Cambridge University Press, 1946.

8. Maxwell, E. A., *General Homogeneous Coordinates in Space of Three Dimensions*, Cambridge University Press, 1951.

9. Newell, M. E., Newell, R. G. and Sancha, T. L., "A Solution to the Hidden Surface Problem", *Proc. of the ACM National Conference*, pp. 443-450, 1972.

10. Watkins, C., Sadun, A. and Marenka, S., *Modern Image Processing: Warping, Morphing and Classical Techniques*, Academic Press, !993.

11. Young, T. Y. and Fu. K. S., eds., *Handbook of Pattern Recognition and Image Processing*, Academic Press, 1968.

12. Young, T. Y., ed., *Handbook of Pattern Recognition and Image Processing: Computer Vision*, Academic Press, 1994

13. Faux, I. D. and Pratt, M. J., *Computational Geometry for Design and Manufacture*, Ellis Horwood, 1979.

14. Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph.D. Thesis, University of Utah, Salt Lake City, UT, USA, December, 1974.

15. Appel, A., "Some Techniques for Shading Machine Renderings of Solids", *Proc. of the Spring Joint Computer Conference*, pp. 37-45, 1968.

16. Goldstein, R. A. and Nagel, R., "3-D Visual Simulation", *Simulation*, vol. 16(1), pp. 25-31, 1971.

17. Whitted, T., "An Improved Illumination Model for Shaded Display", *Communications of the Association for Computing Machinery (CACM)*, vol. 23(6), pp. 343-349, 1980.

18. Haines, E. A., "Essencial Ray Tracing Algorithms", Glasser, A. S., ed., *An Introduction to Ray Tracing*, pp. 33-37, Academic Press, 1989.

19. Hanrahan, P., "A Survey of Ray-Surface Intersection Algorithms", Glasser, A. S., ed., *An Introduction to Ray Tracing*, pp. 79-119, Academic Press, 1989.