3. ALGORITMOS DE COMPUTAÇÃO GRÁFICA MATRICIAL

Os dispositivos de computação gráfica matricial requerem procedimentos especiais para gerar a sua exibição. Neste capítulo apresentaremos algoritmos para traçamento de linhas e círculos, preenchimento de polígonos, transformação e visualização 2D.

3.1 HARDWARE DE EXIBIÇÃO

Existem vários dispositivos de saída para desenhos gerados por computação gráfica. Exemplos típicos são plotters planos, plotters de tambor, impressoras matriciais, impressoras laser, monitores de tubos de raios catódicos (CRT), dentre outros. Como a grande maioria dos sistemas de CAD/CAM utilizam um tipo de monitor CRT e porque a maioria dos conceitos fundamentais de exibição estão incorporados na tecnologia de exibição de dispositivos gráficos matriciais, limitaremos a nossa discussão aos monitores CRT. Para que uma imagem seja gerada no monitor CRT é necessária a atuação de três componentes básicos: um monitor de vídeo CRT, uma memória digital (*frame-buffer*) e uma interface entre estes dois elementos.

- *O Monitor de Vídeo CRT*: os monitores que se utilizam não podem ser simples monitores de televisão, ainda que exista uma grande semelhança, pois os dois possuem um tubo de raios catódicos como componente principal. O monitor para computadores necessita de características mais avançadas, como alta resolução de tela, intensidade de brilho controlada, precisão da frequência de varredura de tela, etc. (vide Figura 3-1). O canhão de elétrons emite um feixe que é acelerado em direção à tela dotada de revestimento de fósforo. No caminho os elétrons são forçados através de um estreito túnel, que é o mecanismo de foco do sistema; em seguida o feixe é direcionado para um ponto particular da tela por um campo eletrostático ou eletromagnético produzido pelo sistema de deflexão (horizontal e vertical). Quando os elétrons atingem a tela, o

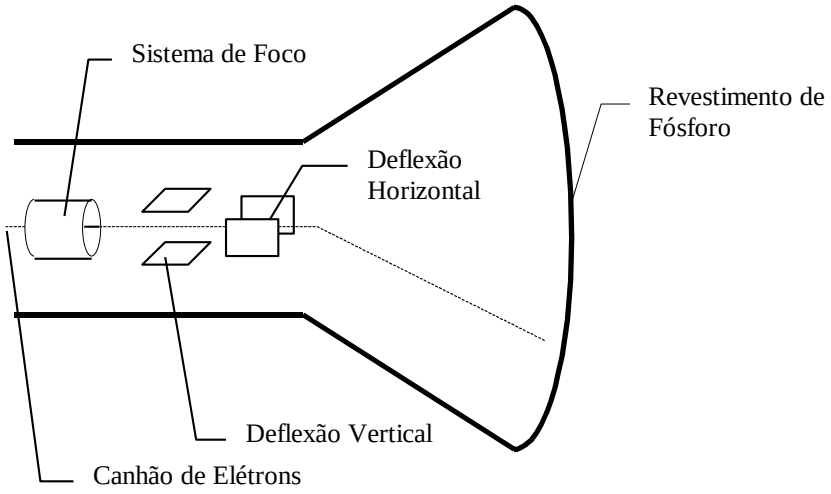

Figura 3-1. Esquema estilizado de um monitor CRT.

fósforo emite luz visível. Como a luz emitida decai exponencialmente com o tempo, toda a imagem deve ser redesenhada várias vezes por segundo para que a figura na tela pareça constante e sem oscilações. O redesenhamento é feito sempre de cima para baixo e da esquerda para a direita, linha por linha. Esta operação é denominada rastreamento ou varredura da tela (vide Figura 3-2). Nos monitores de vídeo, não se emprega o mesmo tipo de CRT utilizado em TV. Dá-se preferência ao CRT com fósforo de alta persistência, geralmente o verde, pois ele é capaz de transformar um feixe de elétrons de intensidade muito menor em luz. Dessa forma a tensão requisitada para o anodo do CRT pode ser menor e a emissão de raios X diminui consequentemente, fato que possibilita uma exposição mais prolongada do operador de computador (que fica bem próximo ao vídeo) prejudicando lhe menos a saúde.

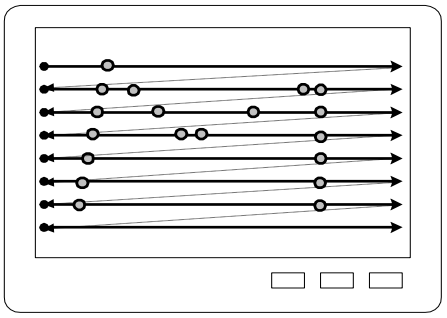

Figura 3-2. Rastreamento da tela.

- *A Memória Digital*: Esta memória é uma matriz de bits onde são armazenados valores binários (0 ou 1), que representam a condição apagado ou acesso do ponto do monitor. Cada posição determinada por uma linha e uma coluna da matriz guarda um valor de brilho para o endereço de tela (coordenadas x e y), vide Figura 3-3. Note que cada endereço de memória define um elemento pontual de uma imagem na tela, denominado *pixel* (abreviatura para *picture element*, ou elemento de figura). Em alguns dispositivos, a definição da cor não é feita de modo direto, possuindo um índice para uma palheta de cores (vetor de cores) que definirá a cor – são os chamados sistemas com cores simultâneas.

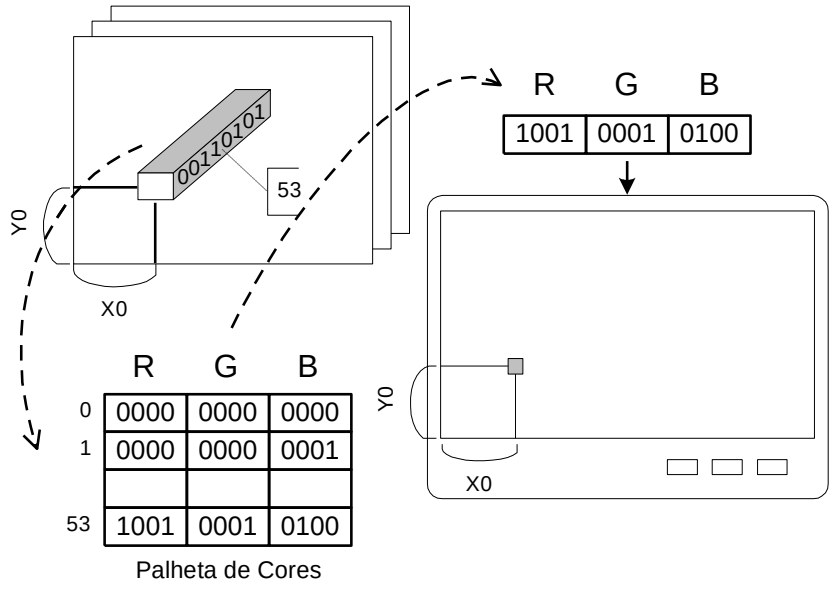

Figura 3-3. Framebuffer e a palheta de cores.

- *A Interface de Controle*: como no monitor CRT utilizam-se sinais analógicos enquanto na memória de tela trabalha-se com sinais digitais, é necessária a existência de uma interface entre estes dois dispositivos que também controle o fluxo de informações de um extremo a outro. Ela recebe o nome de sistema controlador de exibição (*display controler*) e sua função é sincronizar a leitura dos endereços de memória com a geração das coordenadas de rastreamento da tela de modo a utilizar o conteúdo do frame-buffer para controlar a intensidade do pixel. A Figura 3-4 mostra a organização geral do sistema controlador de exibição.

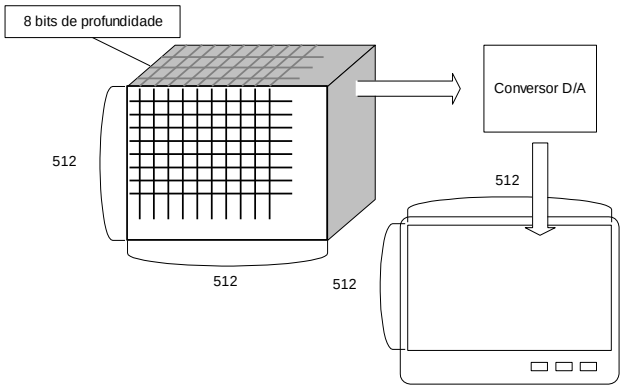

Figura 3-4. Sistema controlador de exibição.

3.2 ALGORITMOS DE TRAÇAMENTO DE LINHAS

As linhas são utilizadas em muitas figuras geradas por computador. Elas aparecem em diagramas de blocos, em histogramas e gráficos, desenhos de engenharia e arquitetura, esquemas lógicos, e muitas outras aplicações. Além disso, linhas de pequenos comprimentos podem ser utilizadas para gerar aproximações de outros tipos de curvas (por exemplo círculos, elipses, etc). Devido a sua grande utilização deve-se ter muito cuidado ao traçá-las. Os requisitos básicos para um bom traçado são quatro:

- As linhas devem parecer realmente retas: as técnicas de plotagem de pontos são ótimas para traçar linhas paralelas ou a 45 graus dos eixos cartesianos. Nos outros casos, entre os pontos inicial e final da linha passará por pontos não endereçáveis. A qualidade da linha dependerá da escolha dos pontos intermediários, podendo parecer "quebrada" ou não.
- As linhas devem terminar corretamente: quando se desenha uma figura geométrica é importante que cada linha termine exatamente onde começar a próxima, para evitar falhas representadas por pequenos saltos.
- As linhas devem ter densidade constante: independentemente do comprimento ou da direção da linha é necessário que o brilho seja uniforme para evitar a sensação de variação de espessura da linha.
- As linhas devem ser traçadas rapidamente: como as linhas são normalmente usadas em grandes quantidades em aplicações gráficas elas devem ser desenhadas rapidamente. Isto implica que se deve usar o mínimo de cálculo para traçá-las.

3.2.1 MÉTODO DDA - DIGITAL DIFFERENTIAL ANALYZER (ANALIZADOR DIFERENCIAL DIGITAL) PARA LINHAS

Esta técnica baseia-se na solução de equações diferenciais. No caso de uma linha temos:

$$\frac{dy}{dx} = cte = \frac{y_f - y_p}{x_f - x_p} \qquad (1)$$

Aonde (x_p, y_p) são as coordenadas do princípio da linha e (x_f, y_f) são as coordenadas do final da linha. O método consiste em incrementar as coordenadas intermediárias (x_i, y_i) a partir do ponto (x_p, y_p) até o ponto (x_f, y_f) de modo a se obter a aproximação para a linha. Definindo os incrementos nas direções x e y como Δx e Δy temos a seguinte solução de recursão:

$$\begin{aligned} x_{i+1} &= x_i + \Delta x \\ y_{i+1} &= y_i + \Delta y \end{aligned} \qquad (2)$$

A relação entre os incrementos deve obedecer à equação diferencial para que a inclinação da linha seja preservada, dessa forma:

$$\frac{\Delta y}{\Delta x} = cte = \frac{y_f - y_p}{x_f - x_p} \qquad (3)$$

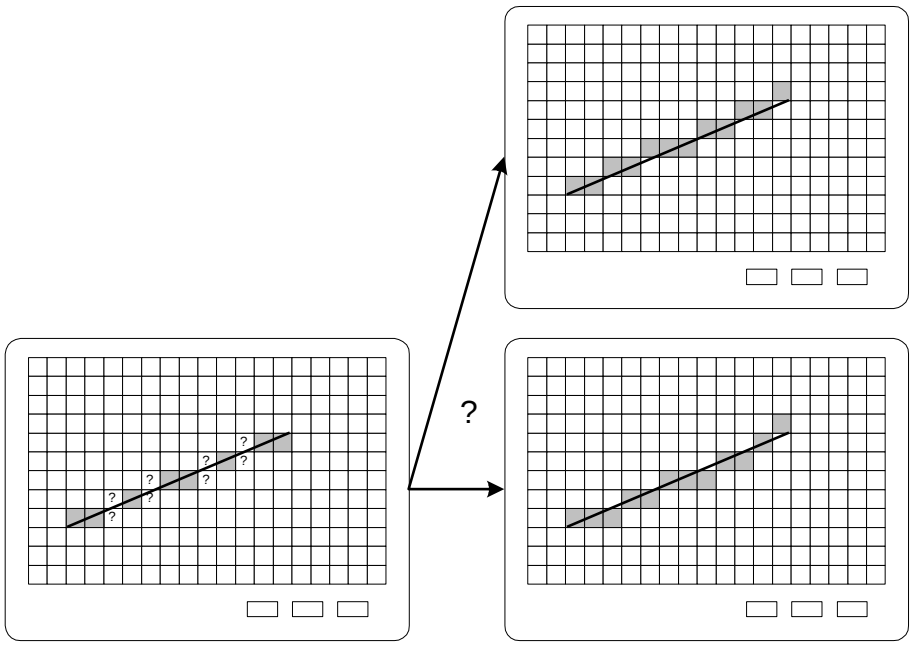

Figura 3-5. Traçamento de linhas.

Pela Figura 3-5 pode-se notar que a coordenada x deve receber sempre um incremento unitário enquanto a coordenada y, ora é incrementada ora se mantém constante. Por essa característica a direção x é denominada direção principal de incremento, que define o comprimento principal da reta; no caso, o comprimento principal é a componente horizontal da reta $(x_f - x_p)$. Como consequência deste fato e devido à relação (3) deduz-se que:

$$\begin{aligned} \Delta x &= \frac{x_f - x_p}{comprimento \quad principal} = 1 \\ \Delta y &= \frac{y_f - y_p}{comprimento \quad principal} < 1 \end{aligned} \qquad (4)$$

Como Δy pode ser um número real e a matriz de pontos só admite valores discretos, a cada iteração, a coordenada y deve ser truncada ou arredondada. Para que o arredondamento seja efetuado de maneira correta é preciso inicializar as coordenadas com o valor 0.5. O traçamento da reta propriamente dito é composto de um simples contador como pode ser observado abaixo:

```
void DDA(float x1, float y1, float x2, float y2)
{
  float Length, delta_x, delta_y, x, y, i;
  /* Determina o maior comprimento */
  if (abs(x2 - x1) > abs(y2 - y1))
    Length = abs(x2 - x1);
  else
    Length = abs(y2 - y1);

  /* Calcula os incrementos de x e y */
  delta_x = (x2 - x1) / Length;
  delta_y = (y2 - y1) / Length;

  /* Inicializa os valores antes do arredondamento */
  x = x1 + 0.5 * Sinal(delta_x);
  y = y1 + 0.5 * Sinal(delta_y);

  /* Inicio do laço principal */
  i = 1
  while (i < Length) {
    Plot(inteiro(x), inteiro(y), 1)
    x += delta_x;
    y += delta_y;
    i ++;
  }
}
```

3.2.2 O ALGORITMO DE BRESENHAM PARA TRAÇAMENTO DE LINHA

O método de Bresenham para traçamento de retas consiste em sempre incrementar de uma unidade a coordenada x, enquanto o incremento da variável y, é determinado examinando-se a distância entre a posição da reta real e a posição associada ao meio do pixel mais próximo. Esta distância é denominada por erro e a eficiência deste algoritmo reside na sua correta manipulação. A Figura 3-6 ilustra uma linha traçada por este algoritmo.

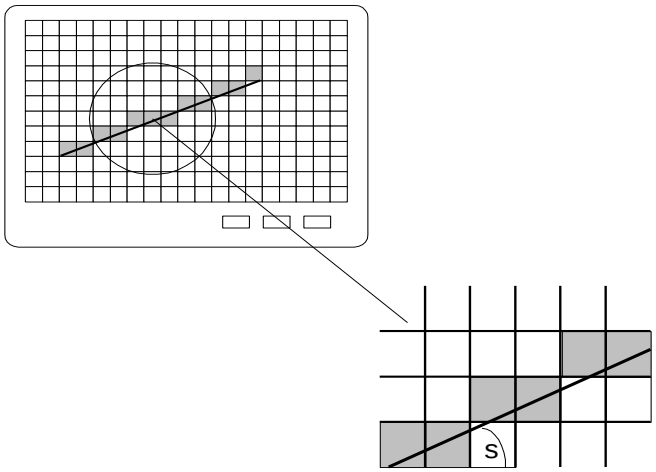

Figura 3-6. Exemplo de traçamento pelo algoritmo de Bresenham.

A Figura 3-7 ilustra um exemplo de traçamento de linha para uma inclinação de $5/12$ da linha real. Supondo que a linha se inicia em $(0,0)$, quando avançarmos para $x = 1$ avançaremos para uma posição de $y = 5/12 < 0.5$ e portanto a coordenada y não será incrementada.

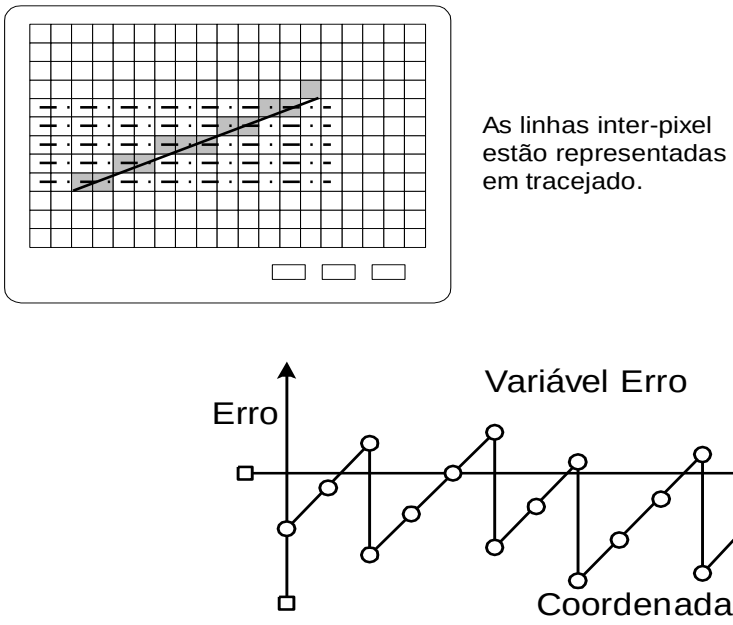

Figura 3-7. Relação da variável erro associada ao algoritmo de Bresenham.

Agora, sendo $x = 2$ atingiremos $y = {}^{10}\!/_{12} > 0.5$ provocando um incremento na coordenada y. Caso inicializemos o erro com -0.5 bastará verificarmos o sinal para determinar se é necessário ocorrer o incremento da coordenada y. Note que para isto, após incrementar a coordenada y é preciso reinicializar o erro, subtraindo-se uma unidade do seu valor. Portanto tem-se o seguinte algoritmo:

```
DesenhaLinhaBresenham(float x1, float y1, float x2, float y2)
{
	int x = x1;
	int y = y1;
	float DeltaX = x2 - x1;
	float DeltaY = y2 - y1;
	float e = DeltaY/DeltaX – 0.5;
	for (i = 1 ; i <= DeltaX ; i++) {
		plot(x, y);
		if (e > 0) {
			y ++;
			e --;
		}
		x ++;
		e += DeltaY/DeltaX;
	}
}
```

É possível melhorar a velocidade deste algoritmo eliminando a operação de divisão e utilizando-se somente variáveis inteiras ao invés de variáveis reais. Uma vez que apenas o sinal do erro é importante, a seguinte transformação é suficiente para atingir este objetivo:

$$e' = 2 \cdot e \cdot \Delta x \qquad (5)$$

Isto permite que este algoritmo seja eficientemente implementado (até mesmo em "*hardware*"). O algoritmo modificado para a aritmética de inteiros é apresentado abaixo:

```
DesenhaLinha(int x1, int y1, int x2, int y2)
{
	int x = x1;
	int y = y1;
	int DeltaX = x2 - x1;
	int DeltaY = y2 - y1;
	int e’ = 2*DeltaY - DeltaX;
	for (i = 1 ; i <= DeltaX ; i++) {
		plot(x, y);
		if (e’ > 0) {
			y ++;
			e’ -= 2*DeltaX;
		}
		x ++;
		e’ += 2*DeltaY;
	}
}
```

3.3 TRAÇAMENTO DE CIRCUNFERÊNCIA

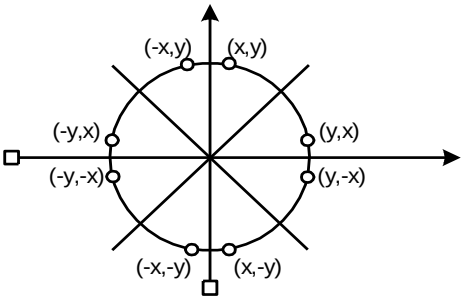

Figura 3-8. Simetria do círculo.

Dentro da computação gráfica a eficiência dos dispositivos matriciais utilizados, depende estreitamente da eficiência dos algoritmos de traçamento de figuras fundamentais, também chamados primitivos de compugrafia matricial. Sem dúvida, além das linhas, as circunferências pertencem a esta classe de elementos devido a sua extensa faixa de aplicação.

Dada uma circunferência com centro na origem, uma primeira análise desta figura geométrica deve considerar uma característica importante: sua simetria em relação aos eixos de centro. Isto implica que basta traçar um quarto da circunferência e gerar as outras partes através de simples reflexões para obter-se seu traçamento completo. Além disso existem também simetrias em relação às retas bissetrizes dos quadrantes. Dessa forma, considerando o primeiro quadrante, pode-se dividi-lo em dois octantes simétricos. Logo, conclui-se que basta gerar um oitavo da circunferência para conseguir todas as coordenadas dos pontos da circunferência completa. As reflexões necessárias podem ser facilmente implementadas como mostra a Figura 3-8.

3.3.1 ALGORITMO PARA TRAÇAMENTO DE CIRCUNFERÊNCIAS PELO PONTO INTERMEDIÁRIO

Como no algoritmo para traçamento de linha, serão realizadas amostragens em intervalos unitários para determinar a cada passo o pixel mais próximo a uma circunferência específica. O algoritmo será configurado para desenhar uma circunferência de raio r ao redor da origem, posteriormente cada ponto poderá ser deslocado para a posição adequada segundo o centro real da circunferência: (x_c, y_c). Ao longo da seção da circunferência de $x = 0$ até $x = y$ no primeiro quadrante, a inclinação varia de 0 a -1. Então, é possível fazer uso de passos unitários segundo o sentido positivo de x e segundo um parâmetro de decisão verificar qual das duas posições em y é mais próxima do contorno da circunferência em cada passo. As posições nos outros octantes são obtidas por simetria. Para aplicarmos o método do ponto intermediário, definiremos a função circunferência:

$$f_{circ}(x, y) = x^2 + y^2 - r^2 \qquad (6)$$

Qualquer ponto (x, y) no contorno da circunferência com raio r satisfaz a equação $f_{circ}(x, y) = 0$. Se o ponto estiver no interior da circunferência, então a função circunferência é negativa. E se o ponto estiver no exterior da circunferência, então a função circunferência é positiva. Resumindo, a posição relativa de qualquer ponto (x, y) pode ser determinada pela análise do sinal da função circunferência:

$$f_{circ}(x,y)=\begin{cases}<0 \Rightarrow dentro\\ =0 \Rightarrow sobre\\ >0 \Rightarrow fora\end{cases} \qquad (7)$$

Os testes da função circunferência descritos na equação (7) serão realizados para o ponto intermediário entre os pixels próximo à circunferência em cada passo de amostragem. Então, a função circunferência é o parâmetro de decisão no algoritmo do ponto intermediário, e é possível configurar os cálculos para números inteiros como realizamos para o algoritmo de traçado de linhas.

A Figura 3-9 exibe o ponto intermediário entre os dois pixels candidatos segundo a posição x_k+1. Assumindo que acabamos de acender o pixel (x_k, y_k), precisamos determinar se é o pixel (x_k+1, y_k) ou o pixel (x_k+1, y_k-1) que está mais próximo do contorno da circunferência. O parâmetro de decisão é a função circunferência (7) avaliada no ponto intermediário entre os dois pixels:

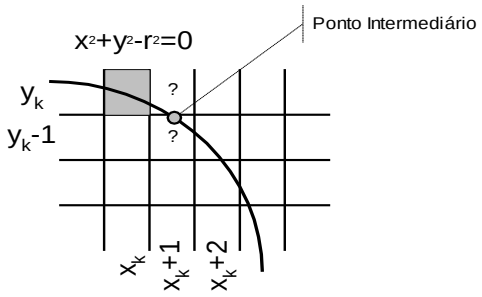

Figura 3-9. Pixels candidatos a serem o próximo a ser aceso e o ponto intermediário.

$$\begin{aligned} p_k &= f_{circ}(x_k+1, y_k-\tfrac{1}{2}) \\ &= (x_k+1)^2+(y_k-\tfrac{1}{2})^2-r^2 \end{aligned} \qquad (8)$$

Se $p_k<0$, este ponto intermediário está interno à circunferência e o pixel na linha y_k está mais próximo ao contorno da circunferência. Em caso contrário, o ponto intermediário está externo à circunferência ou sobre o contorno da circunferência, e será selecionado o ponto na linha y_k-1. Os parâmetros para as sucessivas decisões são obtidos por cálculo incremental. A expressão recursiva para o próximo parâmetro de decisão é obtida pelo desenvolvimento da função circunferência para a posição de amostragem em $x_{k+1}+1=x_k+2$:

$$\begin{aligned} p_{k+1} &= f_{circ}(x_{k+1}+1, y_{k+1}-\tfrac{1}{2}) \\ &= \left[(x_k+1)+1\right]^2+(y_{k+1}-\tfrac{1}{2})^2-r^2 \end{aligned} \qquad (9)$$

Ou

$$p_{k+1}=p_k+2\cdot(x_k+1)+(y_{k+1}^2-y_k^2)-(y_{k+1}-y_k)+1 \qquad (10)$$

Onde y_{k+1} pode ser tanto y_k como $y_k - 1$ dependendo do sinal de p_k. Os incrementos para obter p_{k+1} são $2 \cdot x_{k+1} + 1$ (se p_k for negativo) ou $2 \cdot x_{k+1} + 1 - 2 \cdot y_{k+1}$ (em caso contrário). Na posição inicial $(0, r)$, o parâmetro de decisão inicial é obtido pelo desenvolvimento da função circunferência:

$$\begin{aligned} p_0 &= f_{circ}(1, r - 1/2) \\ &= 1 + (r - 1/2)^2 - r^2 = 5/4 - r \end{aligned} \qquad (11)$$

Caso os incrementos sejam apenas inteiros, ele pode ser arredondado para:

$$p_0 = 1 - r \qquad (12)$$

O algoritmo final é apresentado abaixo:

```
DesenhaCircunferencia(int xc, int yc, int r)
{
    int x, y, p;

    x = 0;
    y = r;
    DesenhaPontoCircunferencia(xc, yc, x, y);
    P = 1 - r;
    while (x < y) {
        if (p < 0)
            x ++;
        else {
            x ++;
            y --;
        }
        if (p < 0)
            p += 2 * x + 1;
        else
            p += 2 * (x - y) + 1;
        DesenhaPontoCircunferencia(xc, yc, x, y);
    }
}
```

A rotina **DesenhaPontoCircunferencia** acende todos os oito pontos da circunferência, simétricos ao ponto (x, y) em relação ao centro da circunferência (x_c, y_c).

3.4 DISTORÇÃO DOS DISPOSITIVOS MATRICIAIS

Uma característica da maioria dos sistemas gráficos é que a densidade de pixels na horizontal difere da densidade de pixels na vertical, propriedade chamada de razão de aspecto. Por exemplo, no modo $640 x 200$, um típico monitor colorido de 200 linhas mostra aproximadamente 70 pixels por polegada na horizontal, mas somente 30 pixels por polegada vertical.

Esta diferença nas densidades de pixels, produz distorções na visualização das linhas e circunferências traçadas pelos algoritmos apresentados nas sessões anteriores. Por exemplo, na Figura 3-10, no modo $640x200$, uma linha entre o pixel $(0,0)$ no canto esquerdo superior e o pixel $(100,100)$ tem uma inclinação matemática de 1, ou seja, esta linha deveria estar disposta a 45 graus em relação aos eixos cartesianos. Entretanto esta linha (linha *a* na Figura 2-10) está comprimida na direção horizontal. Para desenhar uma linha a 45 graus é preciso ajustar o valor das coordenadas para compensar a diferença entre a densidade de pixels na horizontal e na vertical. No modo $640x200$ a razão de aspecto vale aproximadamente $2,4$. Assim, corrigindo as coordenadas finais da reta tem-se $(240,100)$. A reta corrigida (linha *b* na Figura 3-10) apresenta um ângulo de 45 graus na tela. É necessário corrigir as coordenadas de todos os pixels em todos os elementos. Se este procedimento não for seguido, os quadrados parecerão retângulos e círculos serão distorcidos para a forma de elipses.

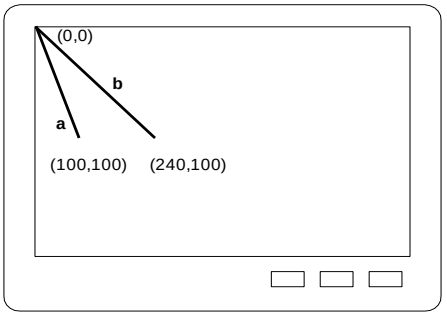

Figura 3-10. Linha distorcida em um dispositivo matricial.

3.5 PREENCHIMENTO DE POLÍGONOS

Um elemento de saída padrão (que possui área) em pacotes gráficos genéricos é uma área poligonal preenchida com um padrão ou com uma cor sólida. Outros tipos de elementos com área podem estar disponíveis, mas os polígonos são mais fáceis de processar. Existem duas propostas básicas para preencher uma área em um dispositivo matricial. Uma maneira é preencher uma área a partir de um ponto inicial (também chamado por semente) interno à área que se desenha preencher, e o algoritmo preencherá esta área a partir deste ponto, até que seja encontrada uma condição de contorno específica. Uma outra maneira é preencher uma área pela determinação da intersecção da área a ser preenchida com linhas de varredura horizontais. O método de preenchimento por linha de varredura é útil para preencher e hachurar polígonos. O método de preenchimento por semente é útil para preencher áreas com contorno complexo e para ser utilizado em sistemas de desenho interativo.

3.5.1 PREENCHIMENTO POR PONTO SEMENTE

Uma proposta para preenchimento de áreas é iniciar o preenchimento a partir de um ponto interno à área e preencher a partir deste ponto em direção ao contorno. Se o contorno for especificado por uma cor única, o algoritmo de preenchimento caminha pixel por pixel até que a cor do contorno seja encontrada. Este método, chamado por *Algoritmo de Preenchimento com Contorno*, é particularmente útil em sistemas de desenho interativo, onde o ponto interior é facilmente selecionável. O procedimento de preechimento com contorno aceita como entrada um ponto interior (x, y), uma cor de preenchimento e uma cor de contorno. Iniciando a partir do ponto (x, y), o procedimento verifica as posições vizinhas para determinar se ela possui a cor de contorno ou se ela já possui a cor de preenchimento. O algoritmo encontra-se detalhado abaixo:

```
BoundaryFill(int a, int b, color cb, color cnew)
{
   color c;

   c = read_pixel(a,b);
   if (c != cb && c != cnew) {
      write_pixel(a, b, cnew);
      BoundaryFill(a+1, b  , cb, cnew);
      BoundaryFill(a  , b+1, cb, cnew);
      BoundaryFill(a-1, b  , cb, cnew);
      BoundaryFill(a  , b-1, cb, cnew);
   }
}
```

Este algoritmo, pelo fato de ser recursivo, utiliza a pilha em demasia. Existem propostas mais eficientes que evitam o uso demasiado da pilha. Um destes métodos realiza varreduras horizontais até encontrar um pixel com a cor de preenchimento ou com a cor de contorno. Sendo que a recursão ocorre apenas para que a linha horizontal abaixo e outra acima sejam percorridas para que o mesmo procedimento seja executado, diminuindo o uso da pilha. Uma variação deste algoritmo chamada por *Algoritmo de Preenchimento por Enchente*, substitui uma cor especificada por uma cor de preenchimento. Tal algoritmo pode ser implementado com pequenas alterações a partir do algoritmo ilustrado anteriormente.

3.5.2 ALGORITMO DE PREENCHIMENTO POR LINHA DE VARREDURA

Figure 3-11 exibe o procedimento de linha de varredura para uma área poligonal. Para cada linha de varredura que intersecciona o polígono, o algoritmo de preenchimento de área determina os pontos de intersecção entre cada linha de varredura e as bordas do polígono. Estes pontos de intersecção são ordenados da esquerda para a direita, e traça-se uma linha a partir de um ponto par para um ponto ímpar segundo a lista ordenada. Na Figura, os quatro pontos de intersecção entre a linha de varredura e o contorno do polígono definem dois pares de pontos, portanto duas linhas serão traçadas.

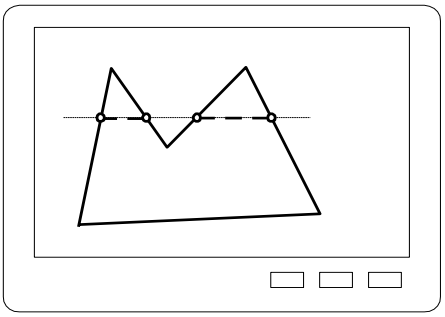

Figura 3-11. Linhas interiores em relação a uma linha de varredura passando sobre um polígono.

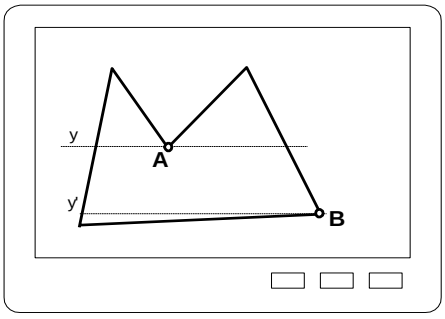

Figura 3-12. Pontos de intersecção entre o polígono e a linha de varredura que ocorrem em vértices do polígono.

Alguns pontos de intersecção entre a linha de varredura e o contorno do polígono necessitam de um processamento especial. A Figura 3-12 , exibe duas linhas de varredura que passam sobre vértices do polígono a ser preenchido: linhas y e y'. Aparentemente, a linha y' intersecciona o polígono em dois pontos, e a linha y intersecciona o polígono em apenas três pontos. A diferença topológica entre estas duas linhas reside no fato de que as arestas vizinhas ao vértice de intersecção podem estar de um mesmo lado em relação à linha de varredura ou estar em lados opostos. As arestas vizinhas ao

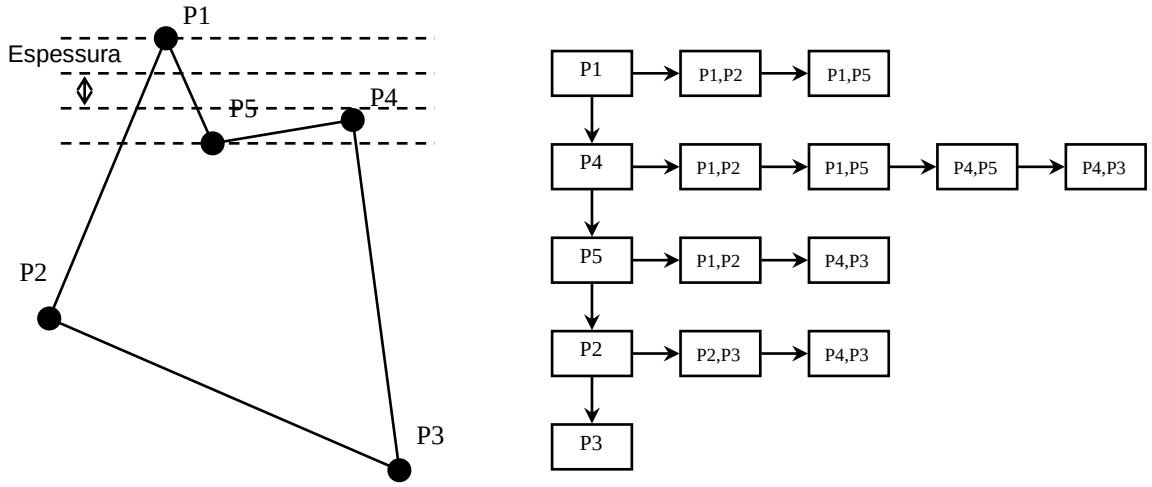

Figura 3-13. Um polígono e sua tabela ordenada de arestas.

vértice de intersecção A, segundo a linha de varredura y, estão posicionadas no mesmo lado. A mesma análise sendo feita para as arestas vizinhas ao vértice de intersecção B, segundo a linha de varredura y', estão posicionados em lados opostos. No caso do vértice A, ele é removido da lista de pontos de intersecção, fazendo com que a linha de varredura y possua apenas dois pontos de intersecção. No caso do vértice B, ele é processado normalmente.

Para executar eficientemente o algoritmo de preenchimento de polígono, é possível inicialmente armazenar o contorno do polígono em uma tabela ordenada de arestas que contém todas as informações necessárias para processar as linhas de varredura de forma eficiente. Varrendo as arestas do polígono segundo um sentido (horário ou anti-horário), ordenam-se as arestas segundo o menor valor y para o maior valor – as arestas horizontais serão ignoradas. No caso ilustrado na Figura 3-13, temos uma lista ordenada de arestas contendo a seqüência: $P1 - P4 - P5 - P2 - P3$. Em seguida produz-se uma lista de arestas ativas. Esta lista contém as arestas do polígono que são interseccionadas pelas linhas de varredura que passam pelos vértices do polígono.

Finalmente, processa-se a lista de arestas ativas para cada linha de varredura. Determinando-se pontos de intersecção que definem as linhas que serão traçadas para preencher o polígono. Caso exista um vértice conforme o caso ilustrado na Figura 3-12 em que as arestas vizinhas estão no mesmo lado em relação à linha de varredura, este vértice será ignorado.

3.6 SISTEMAS DE COORDENADAS (COORDENADAS DO USUÁRIO E COORDENADAS DE DISPOSITIVO)

Com raras exceções, os pacotes gráficos genéricos foram projetados para trabalhar com Coordenadas Cartesianas. Geralmente, vários sistemas cartesianos de referência são utilizados: sistema de *coordenadas locais*, sistema de *coordenadas do usuário*, sistema de *coordenadas normalizadas de dispositivo* e sistema de *coordenadas de dispositivo*. É possível incluir um objeto no desenho principal (por exemplo, um sofá em um pacote de arquitetura), neste caso o objeto está definido em coordenadas locais. O desenho principal está sendo definido segundo coordenadas do

usuário. Ao exibirmos o desenho em algum dispositivo é necessário converter o desenho de coordenadas do usuário para coordenadas de dispositivo. Geralmente, um sistema gráfico, primeiramente, converte as coordenadas do usuário para coordenadas normalizadas de dispositivo, que variam de 0 a 1, antes da conversão final para coordenadas de dispositivo. Isto torna o pacote independente dos vários dispositivos que possam ser utilizados para exibir um determinado desenho em particular – o mesmo desenho pode ser exibido na tela e enviado a um plotter ou impressora. A equação abaixo ilustra a seqüência de conversão de coordenadas em um pacote gráfico:

$$(x_{cl}, y_{cl}) \rightarrow (x_{cu}, y_{cu}) \rightarrow (x_{cnr}, y_{cnr}) \rightarrow (x_{cd}, y_{cd}) \qquad (13)$$

Onde *cl* significa coordenadas locais, *cu* significa coordenadas de usuário, *cnr* significa coordenadas normalizadas de dispositivo e *cd* significa coordenadas de dispositivo. A Figura 3-14 ilustra esta seqüência de transformações de coordenadas locais a coordenadas de dispositivo. Os objetos estão definidos em coordenadas locais, em seguida eles são posicionados no desenho e convertidos para coordenadas do usuário. Para fins de exibição, as coordenadas do usuário são convertidas para coordenadas normalizadas e finalmente convertidas para coordenadas de dispositivo.

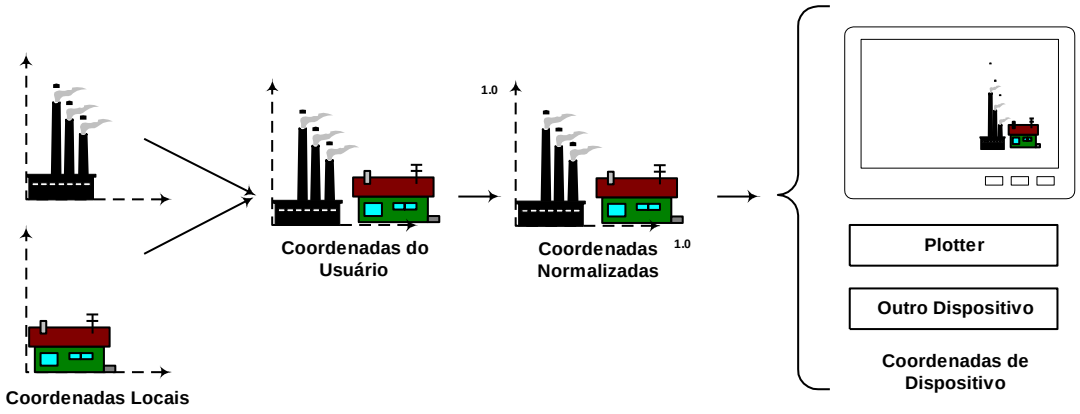

Figura 3-14. Seqüência de transformação de coordenadas locais para coordenadas de dispositivo.

3.7 TRANSFORMAÇÃO DE VISUALIZAÇÃO

Para definir a transformação de visualização, é necessário que o usuário especifique uma região retangular que ele deseja exibir, a janela – *window*, segundo o sistema de coordenadas do usuário como J_e, J_d, J_s e J_i - que significam coordenada horizontal à esquerda, coordenada horizontal à direita, coordenada vertical superior e coordenada vertical inferior. Em seguida, seleciona-se uma região retangular na tela, a *viewport*, onde o usuário

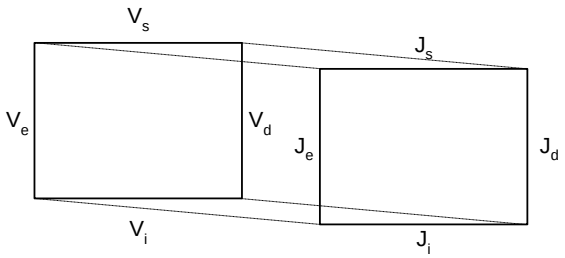

Figura 3-15. Mapeamento Janela-Viewport.

deseja que o desenho seja exibido - V_e, V_d, V_s e V_i. Isto é similar ao que está ilustrado na Figura 3.15. Assim, o pacote gráfico executa o escalamento e translação necessários para converter da janela para a *viewport* e vice-versa, segundo a equação:

$$\begin{aligned} X_{view} &= s_x \cdot X_{janela} + d_x \\ Y_{view} &= s_y \cdot Y_{janela} + d_y \end{aligned} \qquad (14)$$

Onde a escala e a translação podem ser calculados a partir dos requisitos abaixo:

$$\begin{aligned} J_e \ \alpha \ V_e \\ J_d \ \alpha \ V_d \\ J_s \ \alpha \ V_s \\ J_i \ \alpha \ V_i \end{aligned} \qquad (15)$$

O que nos leva a deduzir que:

$$\begin{aligned} s_x &= \frac{V_d - V_e}{J_d - J_e} \\ s_y &= \frac{V_s - V_i}{J_s - J_i} \\ d_x &= V_e - s_x \cdot J_e \\ d_y &= V_i - s_y \cdot J_i \end{aligned} \qquad (16)$$

De acordo com o uso original, o mapeamento janela-*viewport* corresponde a converter coordenadas de usuário convertidas para coordenadas de dispositivo. Entretanto, alguns pacotes gráficos alteraram este conceito original e convertem coordenadas de usuário em coordenadas normalizadas, para apenas depois converter as coordenadas normalizadas em coordenadas de dispositivo.
Porque todo este trabalho? Como discutimos na sessão 2-3, a densidade de pixels na horizontal é diferente da densidade de pixels na vertical – razão de aspecto, e esta diferença pode ser compensada pelo mapeamento janela-*viewport*.

3.8 RECORTE DE LINHAS – *CLIPPING*

Recorte é o processo de selecionar um conjunto específico de informações para exibir uma cena particular de um conjunto maior. A Figura 3-16 ilustra um número variado de situações que um segmento de reta pode apresentar.
O algoritmo proposto por Cohen e Sutherland possui duas partes, a primeira parte verifica se o segmento de reta é completamente visível e se ele pode ser

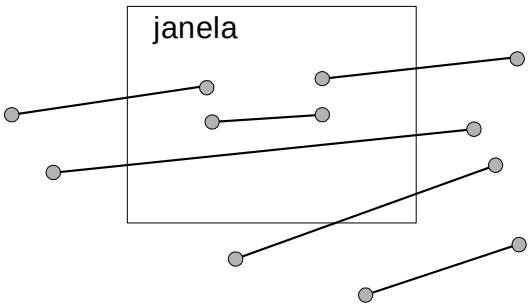

Figura 3-16. Atitudes diferentes de um segmento de linha em relação à borda da janela.

considerado invisível. Esta verificação é realizada pela divisão do espaço em 9 regiões (referencie a Figura 3-17). Cada uma destas regiões possui um código de 4 bits e cada extremidade do segmento de reta possuirá um código de posição dependendo da região a que ele pertence.

1001	1000	1010
0001	janela 0000	0010
0101	0100	0110

Figura 3-17. As nove regiões definidas pelo contorno da janela, exibindo os seus códigos.

Se os quatro bits de ambos os extremos forem zero, então a linha é totalmente visível. E caso a intersecção entre os bits dos códigos de ambos os extremos, não for nula, então o segmento de linha não é visível. Caso o segmento de linha não possa ser considerado como sendo nenhum dos casos acima, então o segmento de linha é subdividido e o teste acima é aplicado novamente.

```
float clipxl, clipxr, clipyb, clipyt;
Code(float x, float y, int *c)
{
   c = 0;
   if (x < clipxl) c = 1;
   else if (x > clipxr) c: = 2;
   if (y < clipyb) c += 4;
   else if (y > clipyt) c += 8;
}
Clip(float x1, float x2, float y1, float y2)
{
   int c, c1, c2; float x, y;
   Code(x1, y1, &c1); Code(x2, y2, &c2);
   while (c1 != 0 || c2 != 0) {
      if (c1 & c2) return;
      c = c1 ; if (c == 0) c := c2;
      if (c & 1) {
         y = y1 + (y2-y1) = (clipxl-x1)/(x2-x1);
         x = clipxl;
      } else
         if (c & 2) {
            y = y1 + (y2-y1) * (clipxr - x1)/(x2-x1);
            x = clipxr;
         } else
            if (c & 4) {
               x = x1 + (x2-x1) (clipyb-y1)/(y2-y1);
               y =clipyb;
            } else
               if (c & 8) {
                  x = x1 + (x2-x1) * (clipyt-y1)/(y2-y1);
                  y = clipyt;
               }
      if (c == c1) { x1 = x; y1 = y; code(x, y, &c1); }
      else { x2 = x; y2 = y; code(x, y, &c2); }
   }
   showline(x1, y1, x2, y2);
}
```