10. 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 10.1.

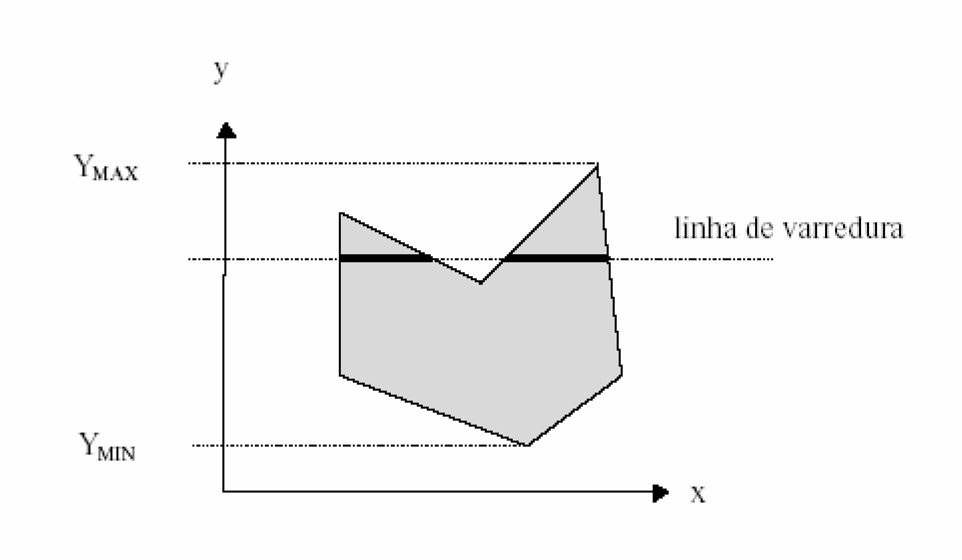

Figura 10.1 - Linhas de varredura para preenchimento de polígonos.

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 10.2).

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 10.3). 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 10.2 - Lista de arestas não horizontais.

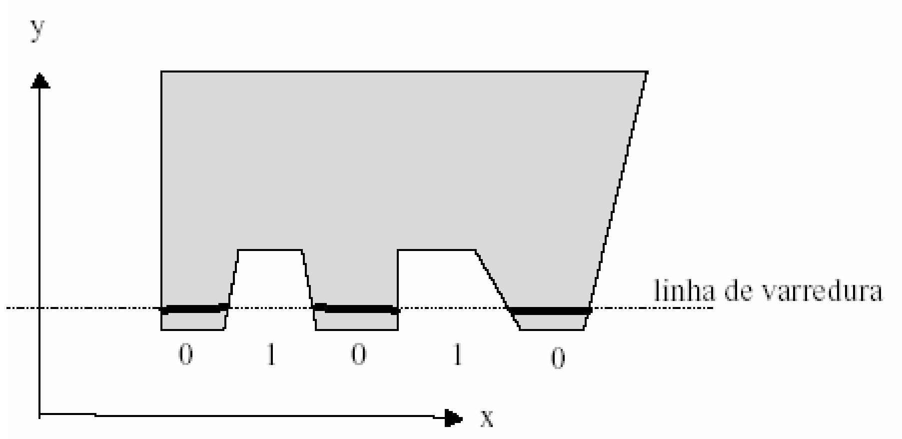

Figura 10.3 - Paridade de segmentos de reta horizontais entre arestas.

Este algoritmo é representado pela função *fillarea* da Figura 10.6. São chamadas as funções *ordena_*x_{min}, 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 10.2. 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 10.4. Na Figura 10.4a, 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 10.4b, 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 10.4a e 10.4b 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 10.5. 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.

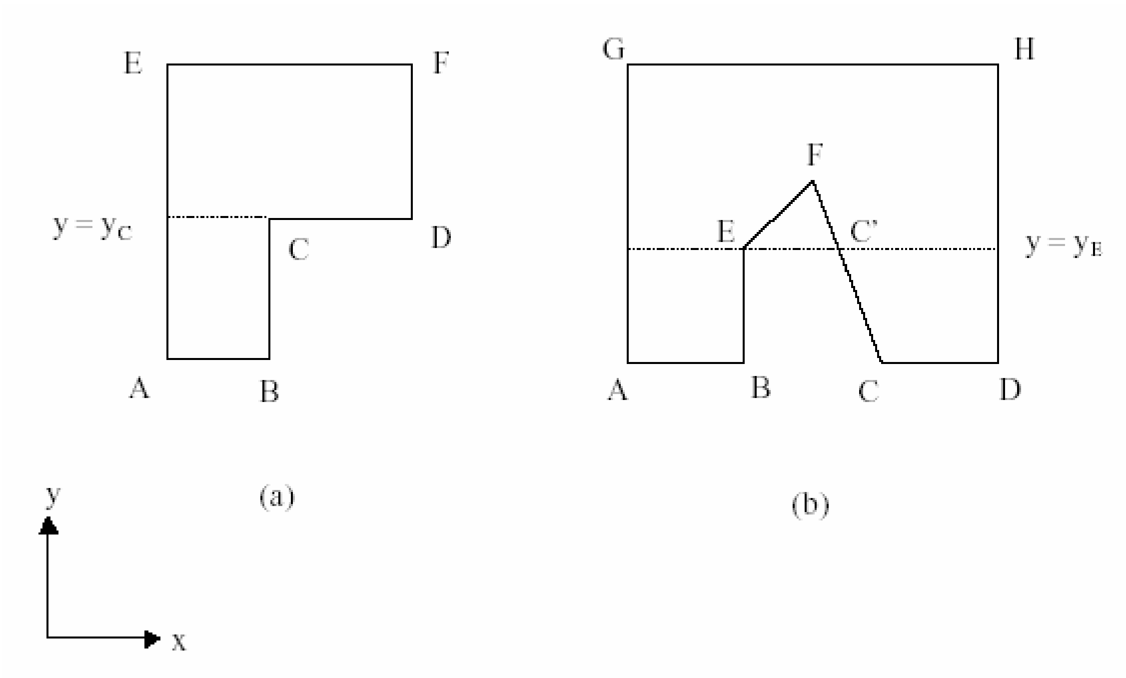

Figura 10.4 - Exemplos de erros que ocorrem quando os vértices de valores máximos em y são considerados. O segmento CD em (a) não seria convertido e o segmento EC' em (b) seria convertido.

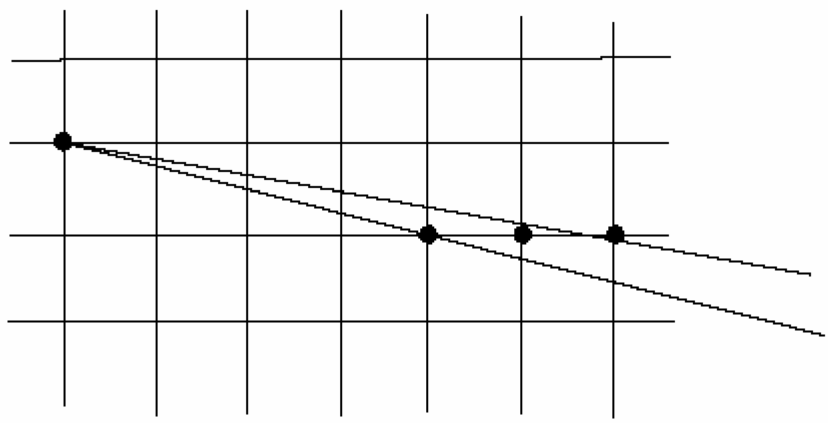

Figura 10.5 - Exemplo de falha no preenchimento de cantos: os *pixels* correspondentes às interseções das três verticais subseqüentes ao vértice esquerdo com as arestas não são convertidos.

```c
#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 10.6 - Algoritmo para preenchimento de polígonos.