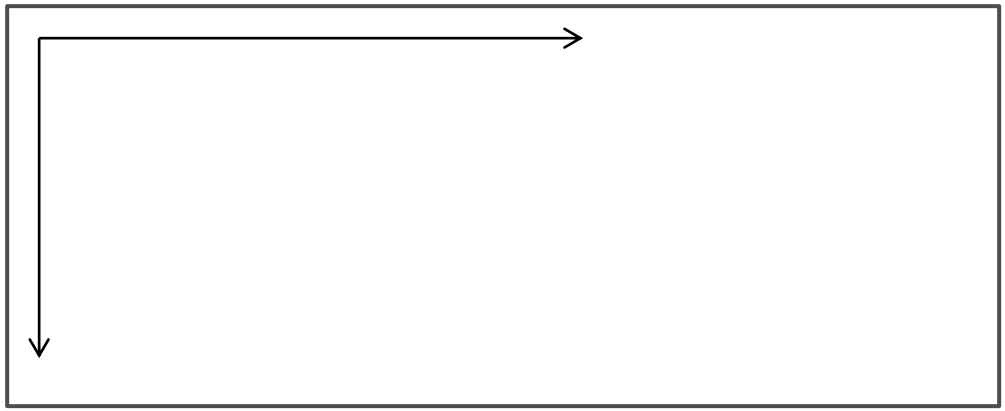
PREENCHIMENTO

Prof. Dr. Bianchi Serique Meiguins
Prof. Dr. Carlos Gustavo Resque dos Santos

Preenchimento de formas triviais

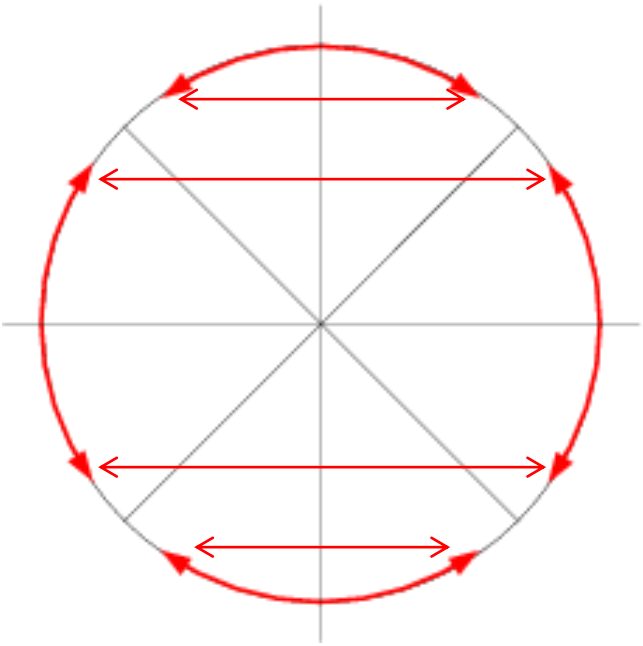
- Retângulo



- Basta pintar todos os pixels dentro do retângulo
- O caso mais simples

Preenchimento de formas triviais

- Círculo
- Elipse



- A ideia é preencher linhas usando os pontos encontrados na rasterização

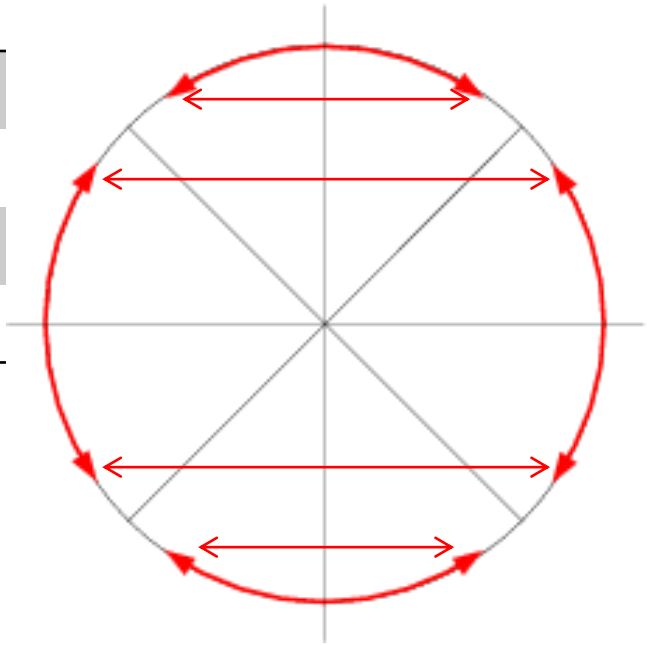
Preenchimento de formas triviais

- Para os Círculos

x	y
x	$-y$
y	x
y	$-x$

$\longleftrightarrow$

$-x$	y
$-x$	$-y$
$-y$	x
$-y$	$-x$



Preenchimento de Triângulos

Triângulos

- Triângulos são os principais blocos de construção de formas mais complexas
- Por definição: são planares e convexos mesmo em ambientes 3D
- Além disso, existem algoritmos para converter representações de superfície para uma malha de triângulos

Preenchimento de Triângulos

- Para preencher triângulos podemos utilizar a equação da reta de cada aresta do triângulo no mesmo sentido (horário ou anti-horário)

- Para cada equação verifica-se um erro para saber se o ponto avaliada está dentro ou fora do triangulo.
  - Esse erro pode ser avaliado pelo sinal do valor do erro

- Utiliza-se uma forma recursiva para atualizar o valor do erro pixel a pixel

Equação Geral da Reta

- A equação geral da reta deixa a equação igualando a zero, o que será utilizado para verificar o erro.

$$ax + by + c = 0$$

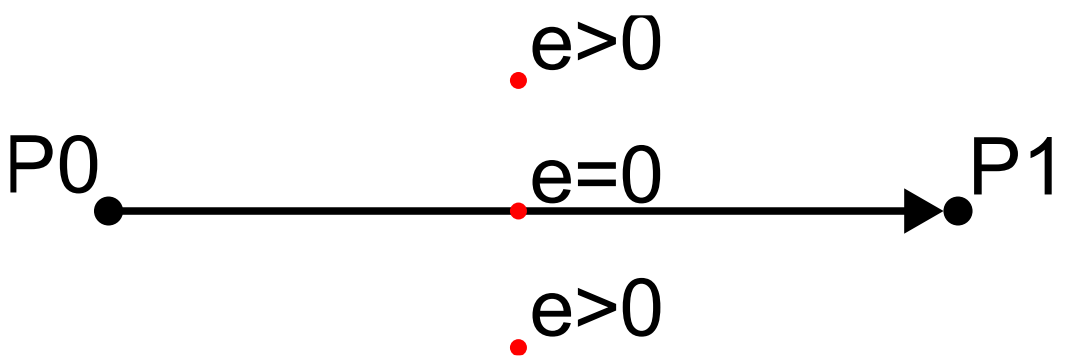
- Se considerarmos que um ponto (x, y) não está na reta, então essa equação resultará em um erro e.

$$ax + by + c = e$$

Equação Geral da Reta

- Considerando as seguintes situações:
  - $e = 0$ para o ponto **na** reta
  - $e > 0$ para o ponto a **esquerda** da reta
  - $e < 0$ para o ponto a **direita** da reta

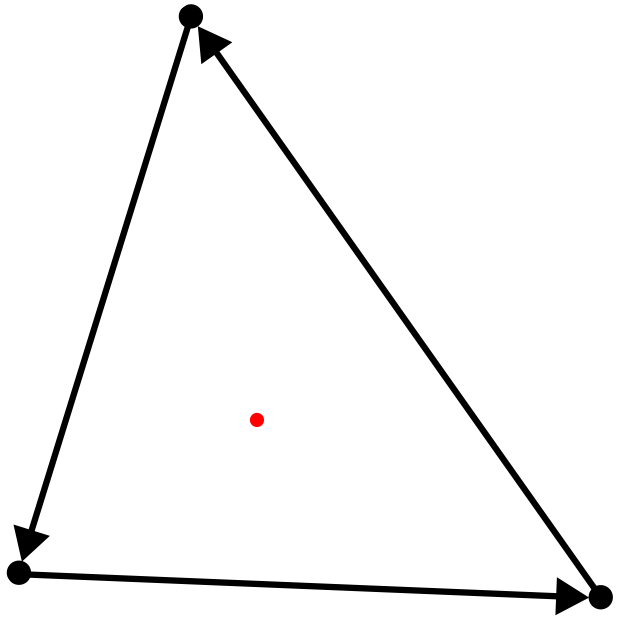
- O que nos dá uma noção de direção para a reta



Equação Geral da Reta

- Sendo assim, podemos usar essa equação para verificar se o ponto está dentro ou fora do triângulo utilizando a equação geral da reta para cada aresta

- Ao considerar as arestas no sentido anti-horário basta verificar se para um determinado ponto o sinal do erro é positivo para todas as arestas



Equação Geral da Reta (Recursiva)

- Deixaremos a equação geral em forma recursiva para x+1 e y+1 para economizar tempo de processamento durante o loop

$$e(x+1,y) = a(x+1) + by + c$$
$$e(x+1,y) = ax + a + by + c$$
$$\underline{e(x+1,y) = e(x,y) + a}$$

$$e(x,y+1) = ax + b(y+1) + c$$
$$e(x,y+1) = ax + by + b + c$$
$$\underline{e(x,y+1) = e(x,y) + b}$$

Calculo dos Coeficientes

- Para calcular os coeficientes da equação geral utilizando a seguinte equação

$$-\Delta_y x + \Delta_x y + \left(\Delta_y x_1 - \Delta_x y_1\right) = 0$$

- Ou seja,

$$a = -\Delta_y$$
$$b = \Delta_x$$
$$c = \Delta_y x_1 - \Delta_x y_1$$

Algoritmo de preenchimento do triângulo

- Calcula os coeficientes a, b e c para cada aresta no sentido anti-horário
- Encontra o bounding box do triângulo
- Calcula o erro inicial para o menor ponto do bounding box
- Para cada pixel no bounding box verifica o erro para cada aresta e atualiza o erro
  - Caso todos >=0 então pinta o pixel
  - Caso contrário não faz nada

Algoritmo de preenchimento do triângulo

```
//Calcula os coeficientes a, b e c para cada aresta no sentido anti-horário
int a1 = -(y2-y1), a2 = -(y3-y2), a3 = -(y1-y3);
int b1 = x2-x1, b2 = x3-x2, b3 = x1-x3;
int c1 = -a1*x1-b1*y1, c2 = -a2*x2-b2*y2, c3 = -a3*x3-b3*y3;

//Encontra o bouding box do triângulo
int x_min = Math.min(Math.min(x1,x2),x3);
int x_max = Math.max(Math.max(x1,x2),x3);
int y_min = Math.min(Math.min(y1,y2),y3);
int y_max = Math.max(Math.max(y1,y2),y3);

//Calcula o erro inicial para o início do bouding box
int e1 = a1*x_min + b1*y_min + c1;
int e2 = a2*x_min + b2*y_min + c2;
int e3 = a3*x_min + b3*y_min + c3;
```

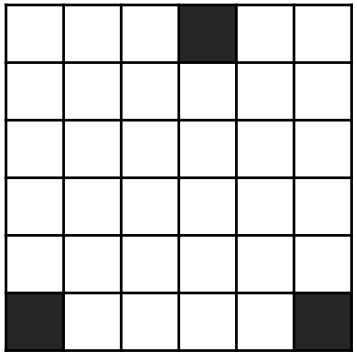
Algoritmo de preenchimento do triângulo

```
//loop no bounding box
for(int y=y_min;y<=y_max;y++){
    int e1aux = e1, e2aux = e2, e3aux = e3;
    for(int x=x_min;x<=x_max;x++){
        if(e1>=0 && e2>=0 && e3>=0) {
            fb.setPixel(x, y, color);
        }

        e1+=a1;
        e2+=a2;
        e3+=a3;
    }
    e1=e1aux+b1;
    e2=e2aux+b2;
    e3=e3aux+b3;
}
```

Algoritmo de preenchimento do triângulo (Exemplo)

- Preencha o triângulo com os seguintes vértices

$$p_1 = (0,0), p_2 = (5,0), p_3 = (3,5)$$



- Calculando os coeficientes temos:

$$a_1 = -(0-0) = \underline{0}$$
$$b_1 = 5-0 = \underline{5}$$
$$c_1 = -0 \times 0 - 5 \times 0 = \underline{0}$$

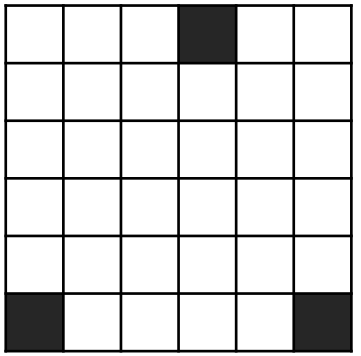
$$a_3 = -(0-5) = \underline{5}$$
$$b_3 = 0-3 = \underline{-3}$$
$$c_3 = -5 \times 3 + 3 \times 5 = \underline{0}$$

$$a_2 = -(5-0) = \underline{-5}$$
$$b_2 = 3-5 = \underline{-2}$$
$$c_2 = 5 \times 5 + 2 \times 0 = \underline{25}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Calculando o bouding box temos:

$x_{min} = \min(0,5,3) = \underline{0}$
$x_{max} = max(0,5,3) = \underline{5}$
$y_{min} = \min(0,0,5) = \underline{0}$
$x_{max} = max(0,0,5) = \underline{5}$



$a_1 = \underline{0}$
$b_1 = \underline{5}$
$c_1 = \underline{0}$

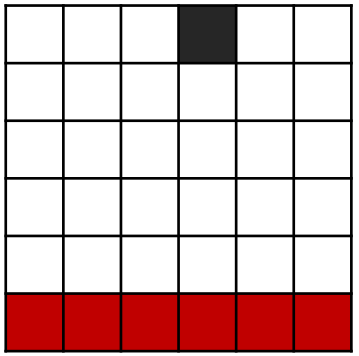
- Calculando o erro inicial temos:

$e_1 = 0 \times 0 + 0 \times 0 + 0 = \underline{0}$
$e_2 = -5 \times 0 - 2 \times 0 + 25 = \underline{25}$
$e_3 = 5 \times 0 - 3 \times 0 + 0 = \underline{0}$

$a_2 = \underline{-5}$ $a_3 = \underline{5}$
$b_2 = \underline{-2}$ $b_3 = \underline{-3}$
$c_2 = \underline{25}$ $c_3 = \underline{0}$

Algoritmo de preenchimento do triângulo (Exemplo)

- Armazena os coeficientes para x inicial e começa o loop:



$$e_{1\_aux} = \underline{0}$$
$$e_{2\_aux} = \underline{25}$$
$$e_{3\_aux} = \underline{0}$$

$$y = 0$$

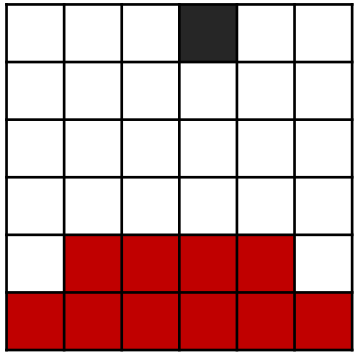
x	e_1	e_2	e_3	Pinta?
0	0	25	0	Sim
1	0	20	5	Sim
2	0	15	10	Sim
3	0	10	15	Sim
4	0	5	20	Sim
5	0	0	25	Sim

$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5} \qquad a_3 = \underline{5}$$
$$b_2 = \underline{-2} \qquad b_3 = \underline{-3}$$
$$c_2 = \underline{25} \qquad c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Armazena os coeficientes para x inicial e começa o loop:



$$y = 1$$

x	e_1	e_2	e_3	Pinta?
0	5	23	-3	Não
1	5	18	2	Sim
2	5	13	7	Sim
3	5	8	12	Sim
4	5	3	17	Sim
5	5	-2	22	Não

$$e_{1\_aux} = 0 + 5 = \underline{5}$$
$$e_{2\_aux} = 25 - 2 = \underline{23}$$
$$e_{3\_aux} = 0 - 3 = \underline{-3}$$

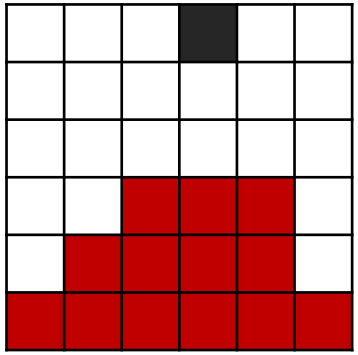
$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5}$$
$$b_2 = \underline{-2}$$
$$c_2 = \underline{25}$$

$$a_3 = \underline{5}$$
$$b_3 = \underline{-3}$$
$$c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Atualiza o valor para x inicial e continua o loop:



$$y = 2$$

x	e_1	e_2	e_3	Pinta?
0	10	21	-6	Não
1	10	16	-1	Não
2	10	11	4	Sim
3	10	6	9	Sim
4	10	1	14	Sim
5	10	-4	19	Não

$$e_{1\_aux} = 5 + 5 = \underline{10}$$
$$e_{2\_aux} = 23 - 2 = \underline{21}$$
$$e_{3\_aux} = -3 - 3 = \underline{-6}$$

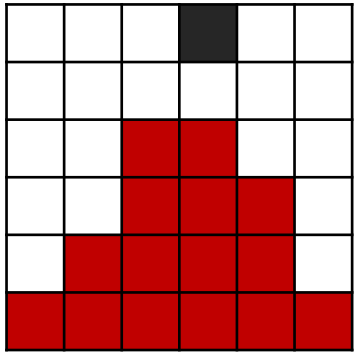
$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5}$$
$$b_2 = \underline{-2}$$
$$c_2 = \underline{25}$$

$$a_3 = \underline{5}$$
$$b_3 = \underline{-3}$$
$$c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Atualiza o valor para x inicial e continua o loop:



$$y = 3$$

x	e_1	e_2	e_3	Pinta?
0	15	19	-9	Não
1	15	14	-4	Não
2	15	9	1	Sim
3	15	4	6	Sim
4	15	-1	11	Não
5	15	-6	16	Não

$$e_{1\_aux} = 10 + 5 = \underline{15}$$
$$e_{2\_aux} = 21 - 2 = \underline{19}$$
$$e_{3\_aux} = -6 - 3 = \underline{-9}$$

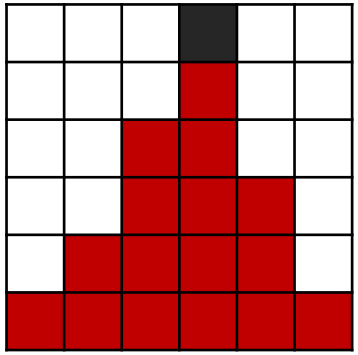
$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5}$$
$$b_2 = \underline{-2}$$
$$c_2 = \underline{25}$$

$$a_3 = \underline{5}$$
$$b_3 = \underline{-3}$$
$$c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Atualiza o valor para x inicial e continua o loop:



$$y = 4$$

x	e_1	e_2	e_3	Pinta?
0	20	17	-12	Não
1	20	12	-7	Não
2	20	7	-2	Não
3	20	2	3	Sim
4	20	-3	8	Não
5	20	-8	13	Não

$$e_{1\_aux} = 15 + 5 = \underline{20}$$
$$e_{2\_aux} = 19 - 2 = \underline{17}$$
$$e_{3\_aux} = -9 - 3 = \underline{-12}$$

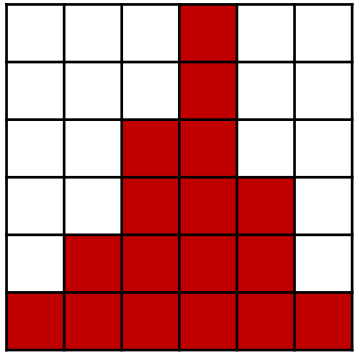
$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5}$$
$$b_2 = \underline{-2}$$
$$c_2 = \underline{25}$$

$$a_3 = \underline{5}$$
$$b_3 = \underline{-3}$$
$$c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Exemplo)

- Atualiza o valor para x inicial e continua o loop:



$$y = 5$$

x	e_1	e_2	e_3	Pinta?
0	25	15	-15	Não
1	25	10	-10	Não
2	25	5	-5	Não
3	25	0	0	Sim
4	25	-5	5	Não
5	25	-10	10	Não

$$e_{1\_aux} = 20 + 5 = \underline{25}$$
$$e_{2\_aux} = 17 - 2 = \underline{15}$$
$$e_{3\_aux} = -12 - 3 = \underline{-15}$$

$$a_1 = \underline{0}$$
$$b_1 = \underline{5}$$
$$c_1 = \underline{0}$$

$$a_2 = \underline{-5}$$
$$b_2 = \underline{-2}$$
$$c_2 = \underline{25}$$

$$a_3 = \underline{5}$$
$$b_3 = \underline{-3}$$
$$c_3 = \underline{0}$$

Algoritmo de preenchimento do triângulo (Conclusões)

- Este algoritmo é eficiente pois utiliza apenas soma dentro do loop
- Sua desvantagem consistem em triângulos muito finos e diagonais, pois geram um *bounding box* grande e poucos pixels são efetivamente pintados
- Existem outros algoritmos eficientes para o preenchimento de triângulos

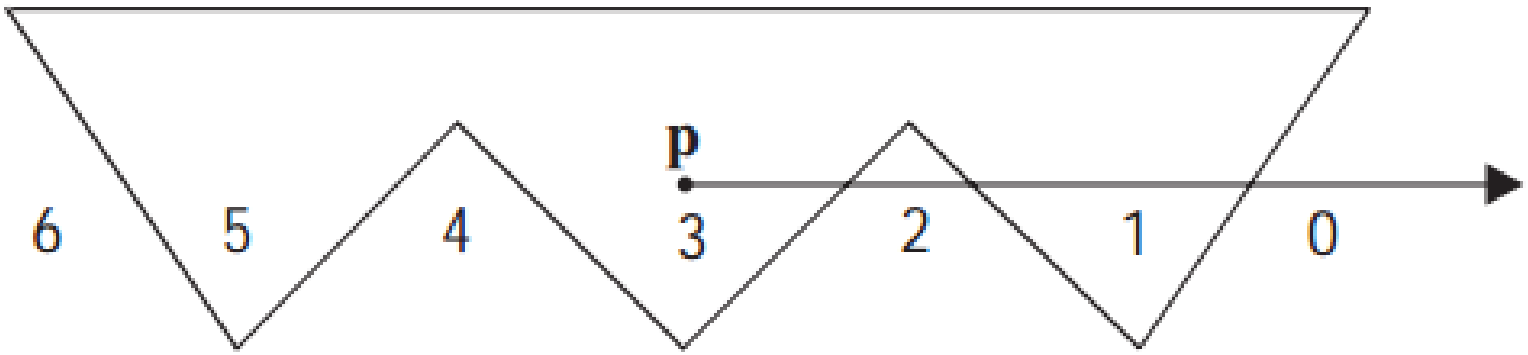
Preenchimento de Polígonos

Preenchimento de Polígonos

- O preenchimento de polígonos exigem mais cuidados, pois essa forma pode ser arbitrariamente complexa
  - Diferente dos triângulos os polígonos podem ser côncavos e não planares
- Isso faz com que os algoritmos para desenhar polígonos sejam mais complexos
- Para verificar se um ponto está dentro de um polígono, podemos utilizar o teste da paridade

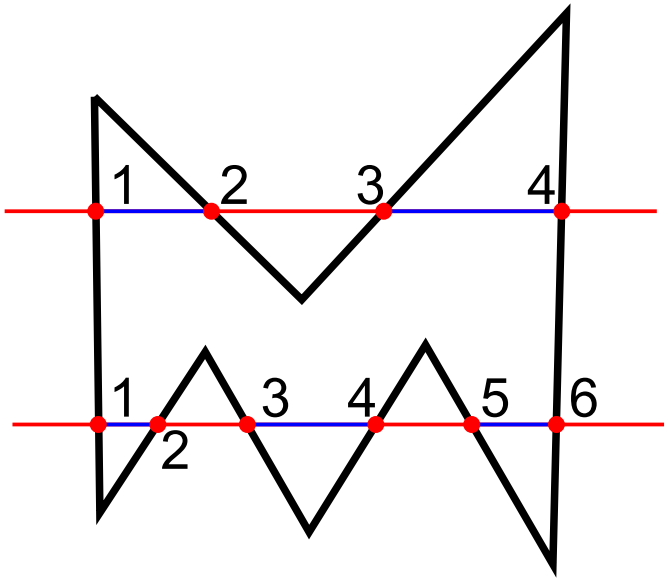
Preenchimento de Polígonos

- Teste de Paridade
  - Consistem em verificar o número de vezes que um segmento de reta intersecta com as bordas do polígono, partindo de uma ponto p até o fim do polígono



Preenchimento de Polígonos

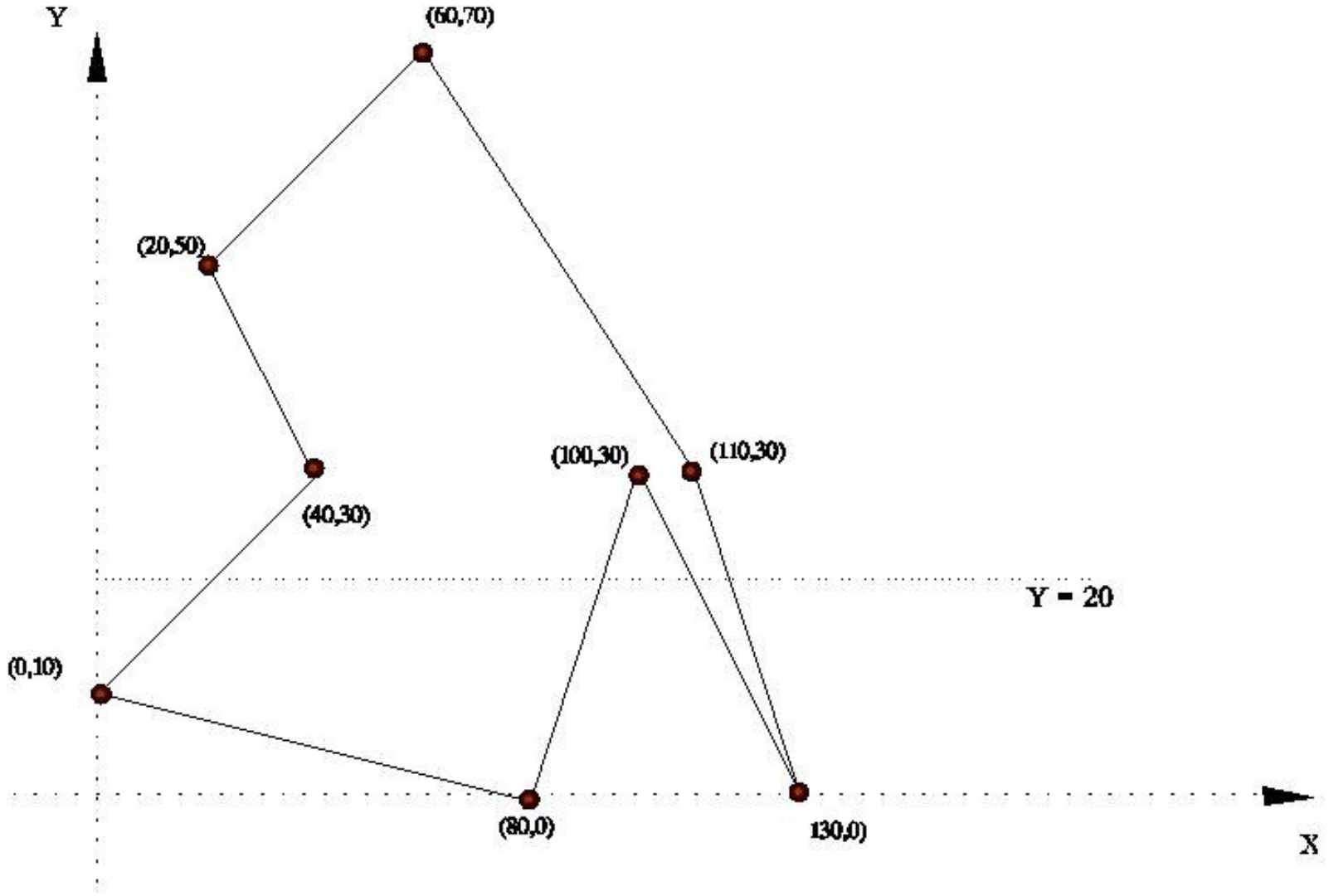
- Teste de Paridade
  - De forma análoga podemos traçar uma linha reta, horizontal da esquerda para a direita, contando as intersecções
  - Assim, quando o número de intersecções for **ímpar** implica que a partir dessa intersecção está **dentro** do polígono e quando for **par** está **fora**



Varredura ou Análise Geométrica (Algoritmo Simples)

- Baseia-se na descrição geométrica e no teste da paridade
- Utiliza linhas de varredura (y = constante)
- Identifica pontos internos do polígono e as interseções das arestas dos polígonos com as linhas de varredura
- Constrói uma tabela de lados para descrição do polígono em questão

Varredura ou Análise Geométrica (Algoritmo Simples)

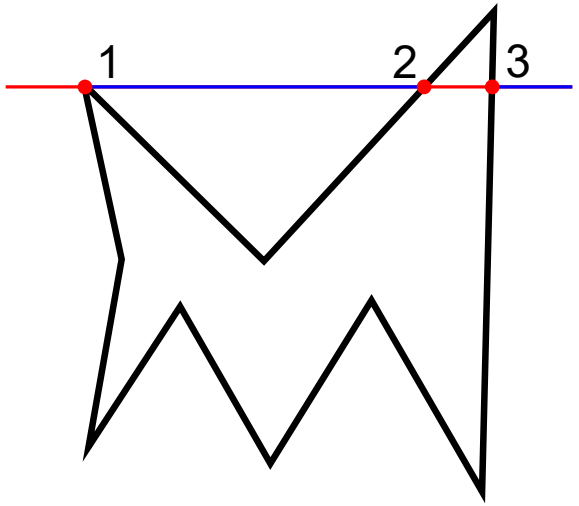


Problema nos Vértices

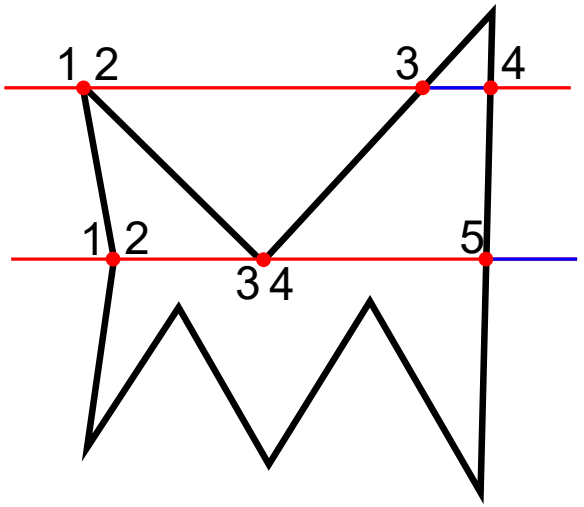
- Seguindo a lógica do algoritmo ocorrem problemas quando a linha de varredura passa por vértices do polígono

- Isso ocorre tanto se contarmos cada vértice como um ponto conjunto para duas arestas (**a**) ou como dois pontos, um para cada aresta (**b**)

(a)



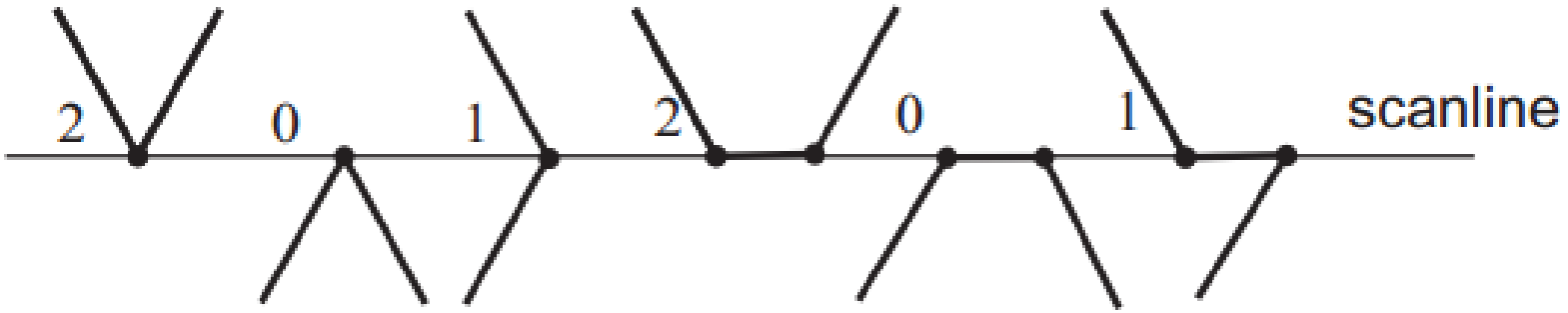
(b)



Problema nos Vértices

- Uma forma de resolver esse problema é considerar um vértice como mínimo ou máximo dependendo do valor de y para cada aresta

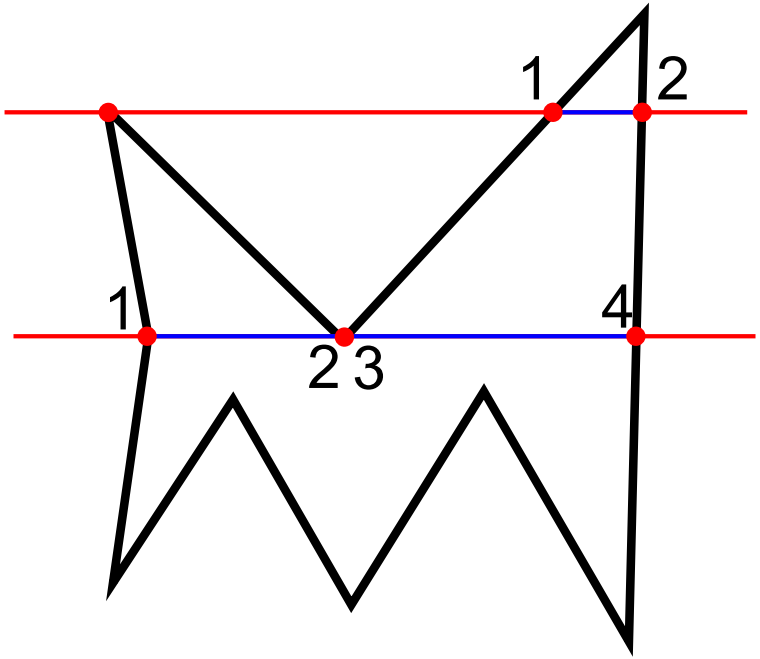
- Sendo assim, contar apenas os pontos de mínimo, por convenção, e uma vez para cada aresta.



Problema nos Vértices

- Contando somente os vértices mínimos para o teste da paridade resolve os problemas apresentados

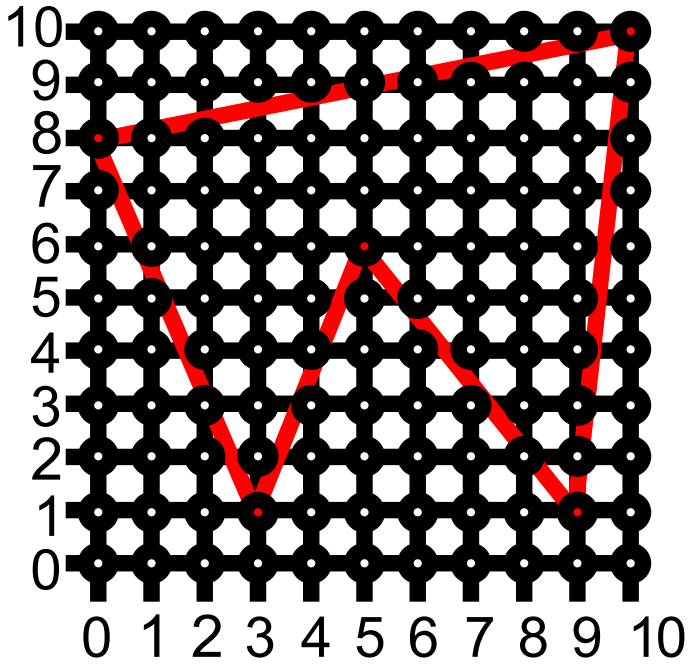
- Essa solução faz com que vértices duplo máximo não sejam preenchidos, mas quando a resolução é alta o problema se torna quase imperceptível



Varredura ou Análise Geométrica (Algoritmo Simples)

- **1° passo:** montar a Tabela de lados - descrição do polígono:

Aresta	$Ymin$	$Ymax$	X_{Ymin}	$1/m$
1	1	8	3	-0.429
2	1	6	3	0.4
3	1	6	9	-0.8
4	1	10	9	0.111
5	8	10	0	5



Varredura (Scanline Algorithm)

- **2° passo:** Para cada linha de varredura $Y_{varredura}$, avaliar quais arestas terão intersecção com $Y_{varredura}$ e calcular o x da interseção com a seguinte fórmula

$$x = \frac{1}{m} \cdot (Y_{varredura} - Y_{min}) + X_{Ymin}$$

- **3° passo:** Pinta uma linha entre cada par de x encontrado no passo 2 de forma ordenada em x

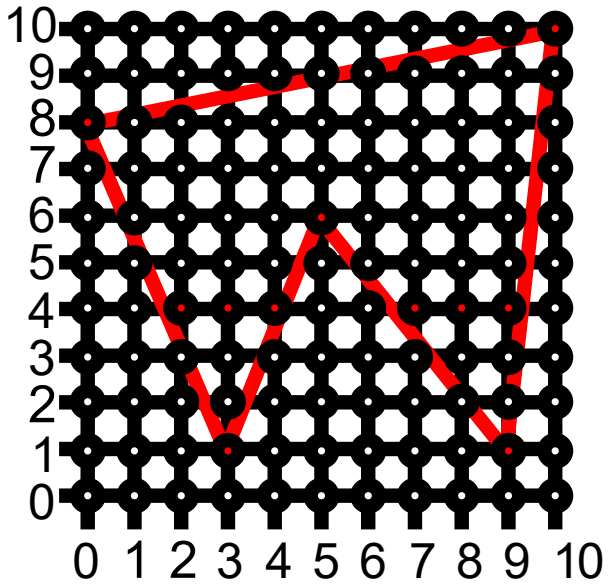
Varredura (Scanline Algorithm)

- Por exemplo, para $Y_{varredura} = 4$ temos:

Aresta	$Ymin$	$Ymax$	X_{Ymin}	$1/m$
1	1	8	3	-0.429
2	1	6	3	0.4
3	1	6	9	-0.8
4	1	10	9	0.111
5	8	10	0	5

$$x = \frac{1}{m} \cdot (Y_{varredura} - Y_{min}) + X_{Ymin}$$

$$x_1 = -0.429 \times (4 - 1) + 3 = \underline{1.713}$$
$$x_2 = 0.4 \times (4 - 1) + 3 = \underline{4.2}$$
$$x_3 = -0.8 \times (4 - 1) + 9 = \underline{6.6}$$
$$x_4 = 0.111 \times (4 - 1) + 9 = \underline{9.333}$$



Algoritmo de Varredura com Pontos Críticos

- Os cálculos de intersecção podem ser economizados dentro do loop ao utilizar a forma recursiva da equação da intersecção

$$x_{y_{var}+1} = \frac{1}{m}\left((y_{var} + 1) - y_{min}\right) + x_{y_{min}}$$

$$x_{y_{var}+1} = \frac{y_{var}}{m} + \frac{1}{m} - \frac{y_{min}}{m} + x_{y_{min}}$$

$$x_{y_{var}+1} = \frac{1}{m}(y_{var} - y_{min}) + x_{y_{min}} + \frac{1}{m}$$

$$x_{y_{var}+1} = x_{y_{var}} + \frac{1}{m}$$

Algoritmo de Varredura com Pontos Críticos

- Além disso, a estrutura da tabela pode ser substituída por uma estrutura que guarda os pontos críticos (vértices de y mínimo para cada aresta) e suas características (conjunto total)

- E uma segunda estrutura (conjunto ativo) que guarda os pontos críticos ativos para cada y de varredura y_{var}

Algoritmo de Varredura com Pontos Críticos

- As características gravadas em cada estrutura são
  - **index:** índice do ponto no array de pontos
  - **dir:** direção (anti-horária ou horária) do ponto crítico em direção ao ponto máximo da aresta
  - **x_int:** x da intersecção para o y_var corrente
  - **inv_slope:** o inverso do coeficiente angular da aresta do ponto crítico, ou seja, $\frac{1}{m}$

Algoritmo de Varredura com Pontos Críticos

- O algoritmo consistem em:

1. Armazenar todos os pontos críticos e suas características no conjunto total
2. Para cada y_var em ordem crescente, alterando sempre o conjunto ativo
   a) Atualiza os valores de x_int
   b) Adiciona os pontos críticos iguais a y_var
   c) Remove os pontos críticos com y_max = y_var
   d) Ordenas os pontos se necessário
   e) Pinta os pixels entre cada par de x_int dos pontos críticos atualmente armazenados

Algoritmo de Varredura com Pontos Críticos

- Exemplo de estrutura para guardar os pontos críticos

```
static class CriticalP {

    int index;
    int dir;
    float x_intersection;
    float inv_slope;
```

Algoritmo de Varredura com Pontos Críticos

```
//encontra o bouding box para y e os pontos criticos (minimos em y)
int y_min=Integer.MAX_VALUE, y_max=Integer.MIN_VALUE;
ArrayList<CriticalP> criticals = new ArrayList<>();
for (int i = 0; i < points.length; i++) {
    if(points[i].y<y_min){
        y_min=points[i].y;
    }else if(points[i].y>y_max){
        y_max = points[i].y;
    }
    Point p_aux = points[(i+1)%points.length];
    if(points[i].y<p_aux.y){
        criticals.add(new CriticalP(i,  dir: 1, points[i].x,
                inv_slope: (p_aux.x-points[i].x*1.0f)/(p_aux.y-points[i].y*1.0f)));
    }
    p_aux = points[(i-1+points.length)%points.length];
    if(points[i].y<p_aux.y){
        criticals.add(new CriticalP(i,  dir: -1, points[i].x,
                inv_slope: (p_aux.x-points[i].x*1.0f)/(p_aux.y-points[i].y*1.0f)));
    }
}
```

Algoritmo de Varredura com Pontos Críticos

- Início do loop de varredura

```
ArrayList<CriticalP> active_criticalPs = new ArrayList<>();
CriticalPComparator comparator = new CriticalPComparator();
for(int y=y_min;y<=y_max;y++){

    //Atualiza o valor de cada intersecção nos pontos ativos
    for (CriticalP e : active_criticalPs) {
        e.x_intersection += e.inv_slope;
    }

    //Adiciona as arestas com pontos críticos para o y corrente
    for (CriticalP e : criticals) {
        if(points[e.index].y==y){
            active_criticalPs.add(e);
        }
    }
```

Algoritmo de Varredura com Pontos Críticos

- Ainda no mesmo loop

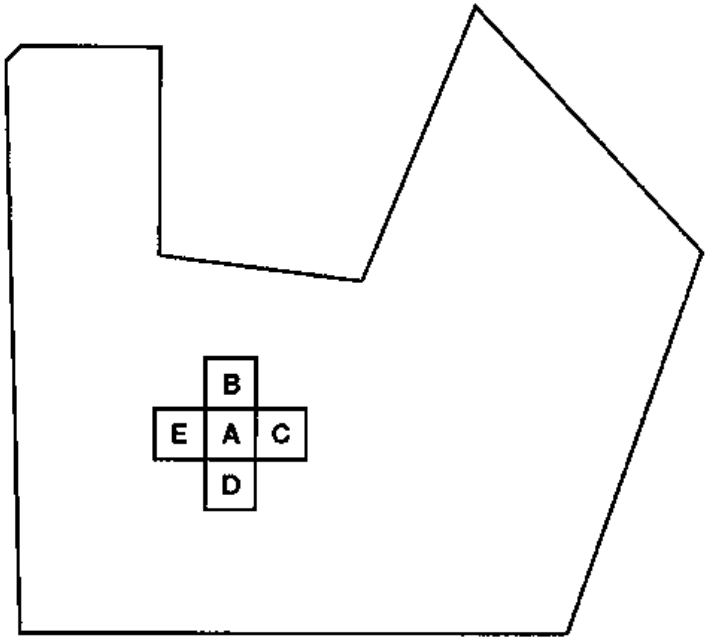
```
        //Remove os pontos com y_max=y_var
        for (int i = active_criticalPs.size()-1; i >= 0; i--) {
            CriticalP e = active_criticalPs.get(i);
            Point p_max = points[(e.index+e.dir +points.length)%points.length];
            if(p_max.y == y){
                active_criticalPs.remove(i);
            }
        }

        //Ordenas os pontos ativos conforme o valor de x para o y corrente
        active_criticalPs.sort(comparator);

        //Pinta entre cada par de pontos ativos
        for (int i = 0; i < active_criticalPs.size(); i+=2) {
            int x_start = Math.round(active_criticalPs.get(i).x_intersection);
            int x_end = Math.round(active_criticalPs.get(i+1).x_intersection);
            for (int x = x_start; x <= x_end; x++) {
                fb.setPixel(x,y,color);
            }
        }
    }
```

Preenchimento de Áreas

Preenchimento Recursivo

- Um pixel vizinho de um outro pixel, que já esta dentro do polígono, também está dentro do polígono

- Considera-se vizinho os 4 pixels:
  - Cima, Direita, baixo, esquerda.



Preenchimento Recursivo

```
FloodFill(x,y,color,edgeColor):
        current = lerPixel(x,y)
        se(current != edgeColor && current != color):
                pintarPixel(x,y,color)
                FloodFill(x+1,y, color, edgeColor)
                FloodFill(x,y+1, color, edgeColor)
                FloodFill(x-1,y, color, edgeColor)
                FloodFill(x,y-1, color, edgeColor)
```

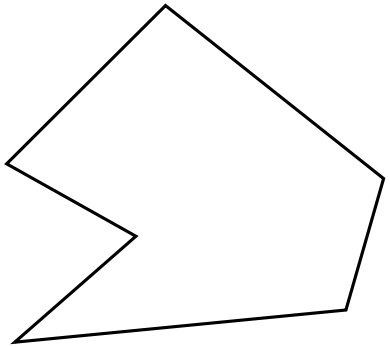
Preenchimento Recursivo

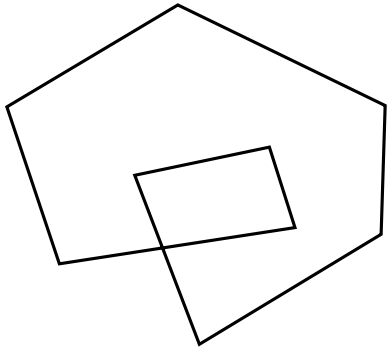
- O preenchimento recursivo preenche qualquer área fechada por mais complexa que seja
- Esse algoritmo funciona sem necessidade de conhecer as coordenadas geométricas do objeto a ser preenchido
- É necessário a pré-existência das bordas do objeto no framebuffer ou imagem
- O resultado deste algoritmo depende do ponto inicial escolhido, geralmente pelo próprio usuário
- O algoritmo possui várias versões, inclusive mais otimizadas

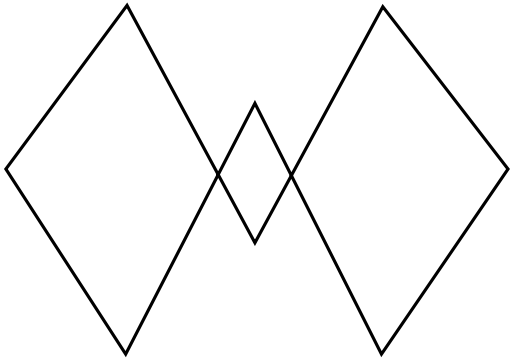
Exemplos

- Preencher os seguintes polígonos

Scanline







Recursivo