

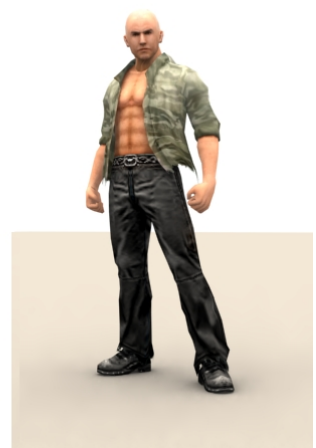
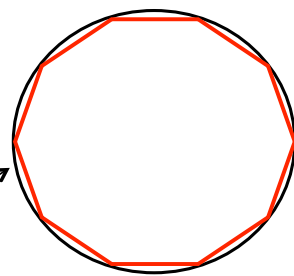
# - INF01047 -

## Rasterização de Triângulos e Polígonos em Geral



### Rasterização de Polígonos

- Em Computação Gráfica, os polígonos “dominam o mundo!”
- Duas razões:
  - Podemos representar qualquer superfície com erro controlável
  - Simplicidade e entendimento matemático ajudam nos algoritmos



# Rasterização de Polígonos

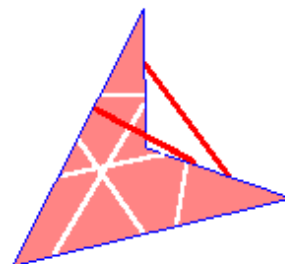
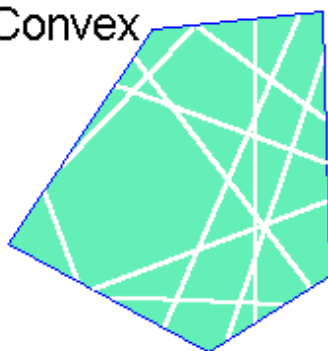
- Triângulo é a unidade mínima para representar um polígono
- Todo polígono pode ser dividido em triângulos
- Triângulos são garantidamente
  - Planares
  - Convexos

O que exatamente significa ser convexo?

## Formas Convexas

- Uma figura geométrica é convexa se e somente se qualquer segmento ligando dois pontos quaisquer da borda estiver inteiramente contido na figura

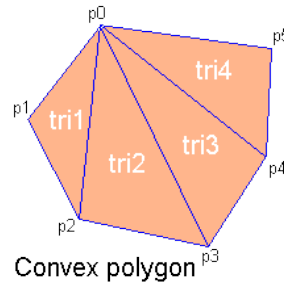
Convex



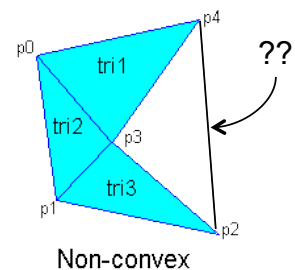
Non-convex

# Triangularização

- Polígonos convexos são facilmente *triangularizáveis*

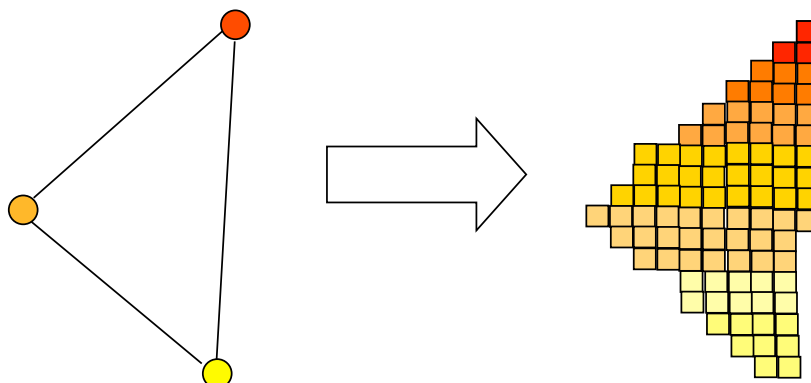


- Polígonos côncavos são mais difíceis



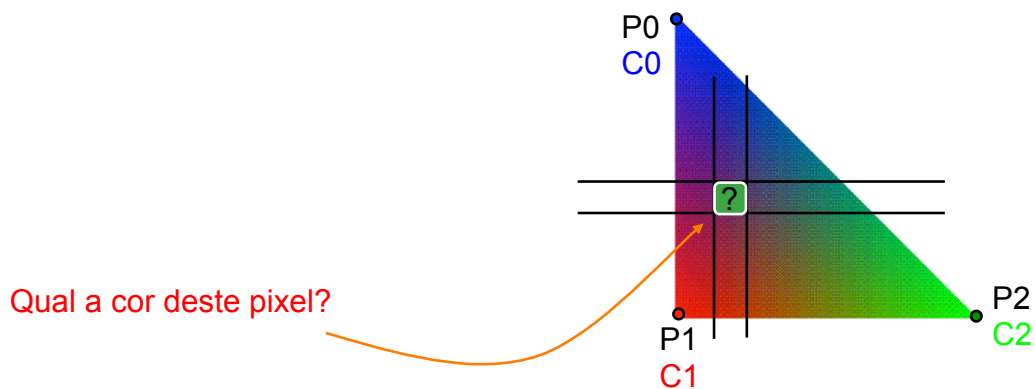
## Rasterização de Triângulos

- Converter a representação matemática do triângulo em pixels
- Preencher o interior do triângulo



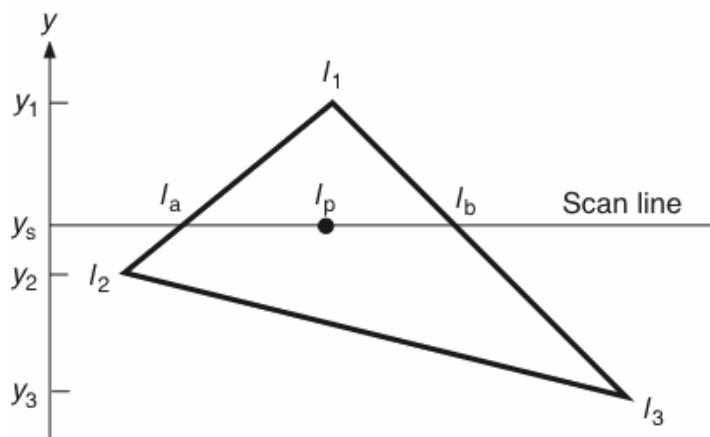
# Interpolação dos Valores

- Usualmente neste processo interpolam-se valores definidos apenas para os vértices (ex: cores RGB)
- Cálculo da cor nos demais pixels que compõem o triângulo a partir das cores nos vértices



# Interpolação dos Valores Internos

- Interpolação Linear, a partir dos valores nos vértices
- Conhecidos:  $I_1$ ,  $I_2$  e  $I_3$



$$I_a = I_1 - (I_1 - I_2) \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 - (I_1 - I_3) \frac{y_1 - y_s}{y_1 - y_3}$$

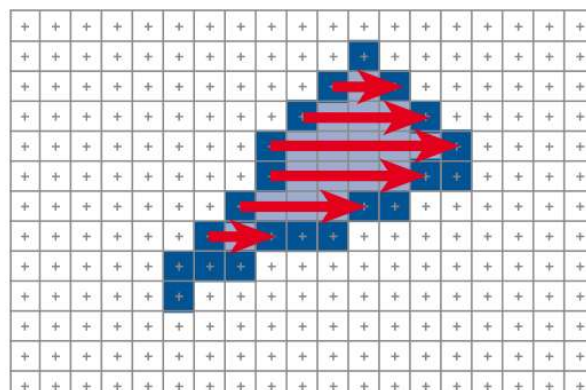
$$I_p = I_b - (I_b - I_a) \frac{x_b - x_p}{x_b - x_a}$$

# Rasterização de Triângulos

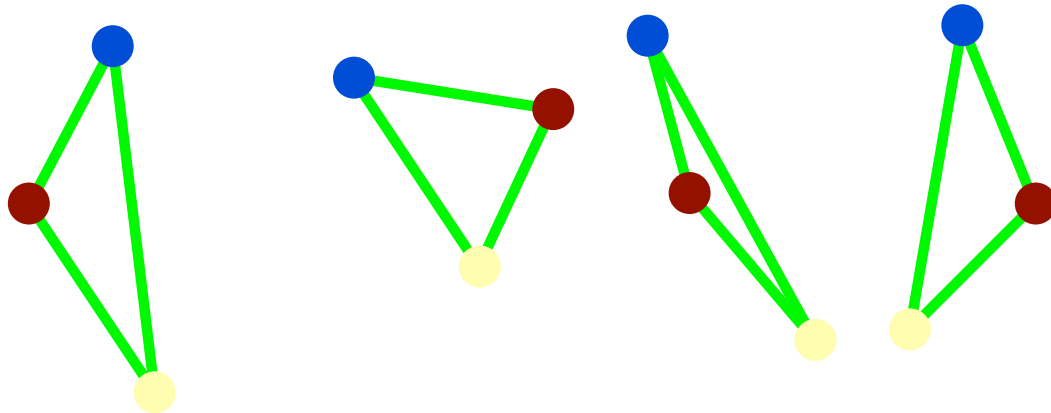
- O hardware atual utiliza basicamente duas técnicas para rasterização de triângulos:
  - Edge walking (“caminhamento pelas arestas”)
  - Edge equation (“equações de arestas”)

## Edge Walking

- Idéia Básica:
  - Desenha arestas da fronteira do triângulo
  - Preenche os intervalos horizontais (chamados *spans*) para cada linha de *scan* que pertence às arestas
  - Interpola a medida que avança



# Tipos de Triângulos

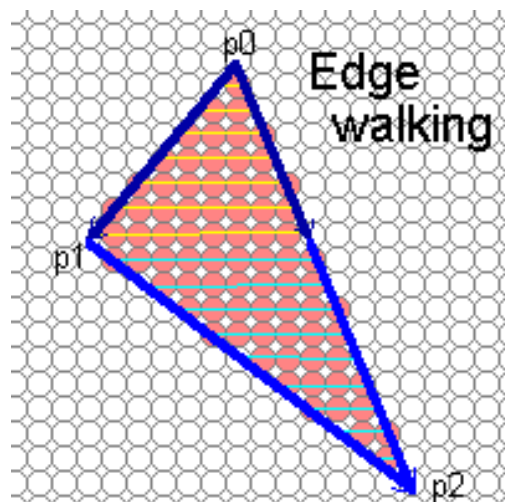


Como determinar os spans? Onde começar?

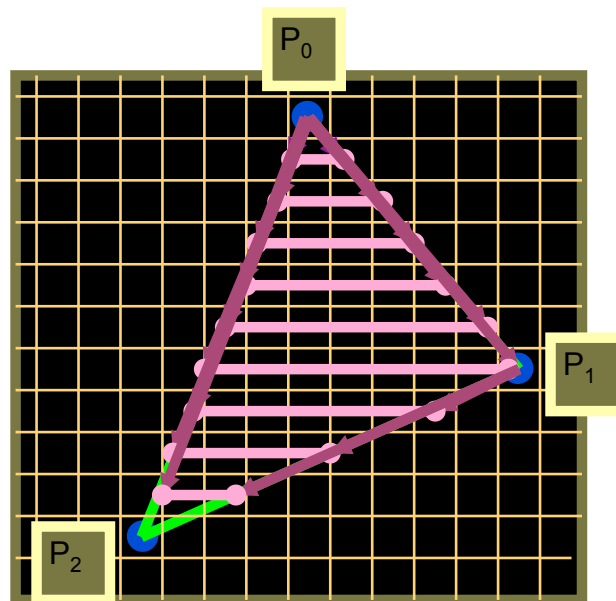
Qual o vértice especial nestes triângulos?

## Edge Walking: Detalhes

- Ordena os 3 vértices em  $x$  e  $y$
- A partir do vértice com maior  $y$ 
  - Para cada linha de *scan*, encontrar os pixels à esquerda e à direita
  - Preenche cada linha até que limite inferior seja atingido
  - Proceder no sentido para baixo (diminui  $y$ )

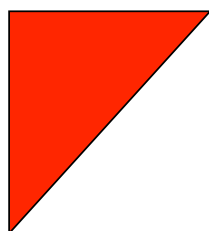


# Edge Walking



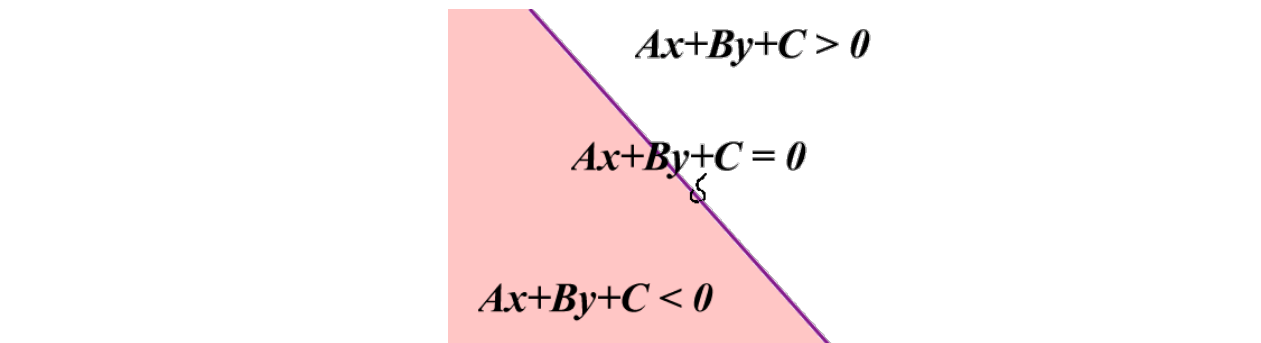
# Edge Walking

- Vantagem
  - rápido
  - pode ser implementado em hardware
- Desvantagens
  - Muitos casos especiais para serem testados e tratados separadamente
  - Ex:  $y$  constante (uma aresta paralela as linhas de *scan*)



## Edge Equations

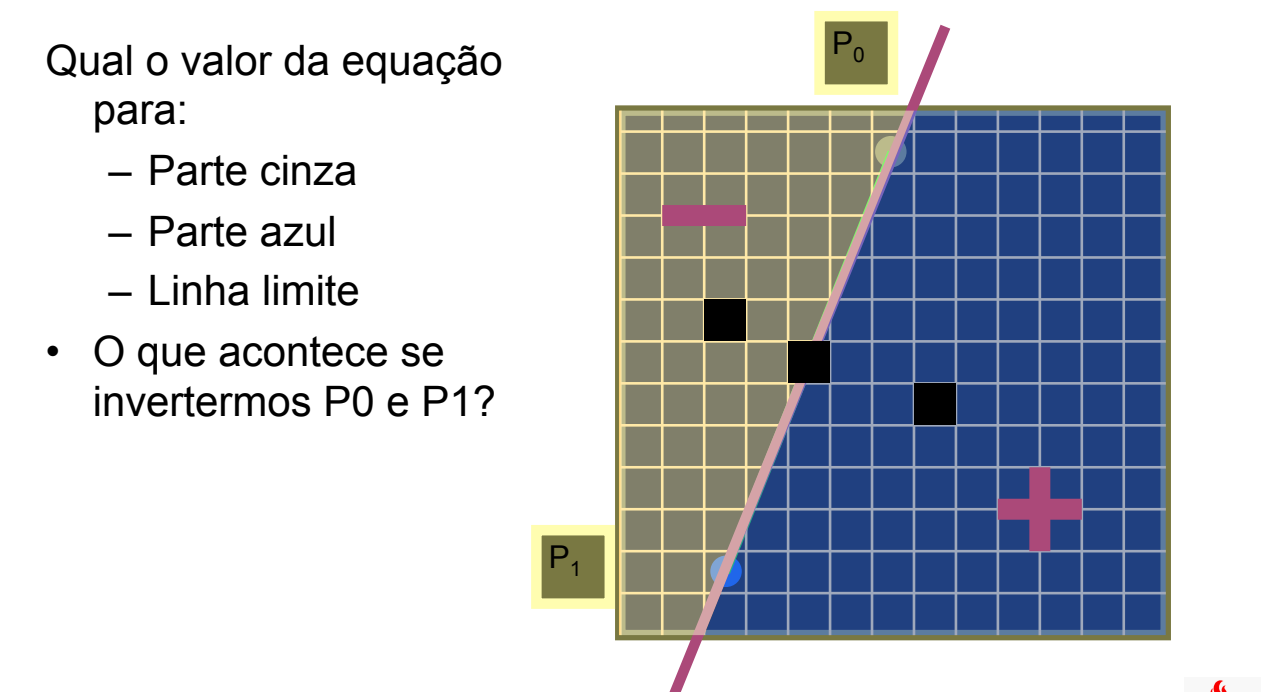
- Uma equação de aresta nada mais é do que a equação da linha que define a aresta
- Usamos a propriedade de divisão do espaço pelas retas para saber em que lado do triângulo estamos



© 2004 Blackwell Publishing Ltd *Journal of Internal Medicine* 255: 105–112

Qual o valor da equação para:

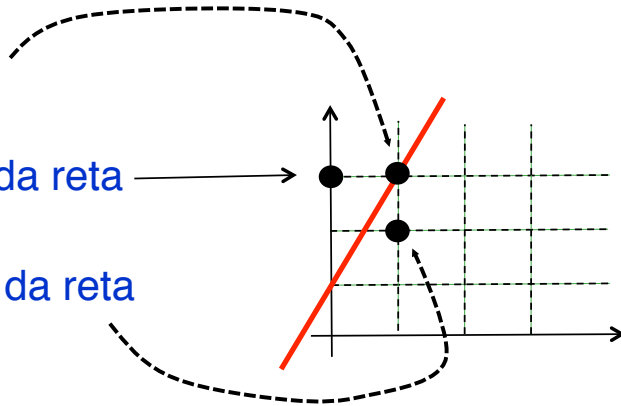
- Parte cinza
- Parte azul
- Linha limite
- O que acontece se invertermos P0 e P1?





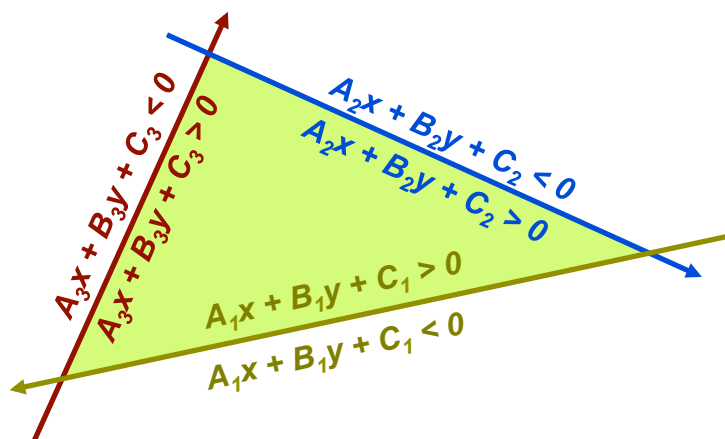
# Divisão do Espaço por Retas

- Uma reta divide o espaço em 3 situações possíveis:
  - Pontos na reta
  - Pontos “acima” da reta
  - Pontos “abaixo” da reta
- Ex:  $y - 2x - 1 = 0$ 
  - P1(1,3) na reta  
 $3 - 2 \cdot 1 - 1 = 0$
  - P2(0,3) acima da reta  
 $3 - 2 \cdot 0 - 1 > 0$
  - P3(1,2) abaixo da reta  
 $2 - 2 \cdot 1 - 1 < 0$



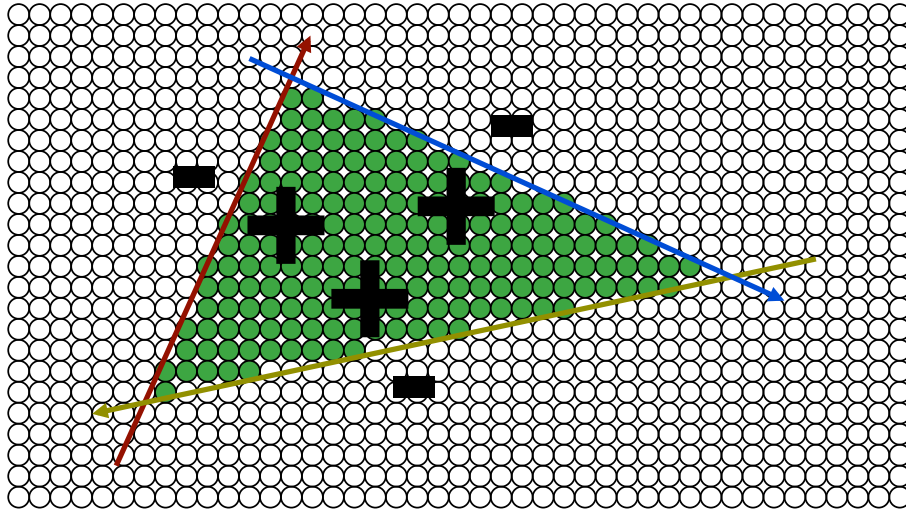
## Edge Equations

- Um triângulo pode ser definido como a intersecção de 3 semi-espacos positivos



# Edge Equations

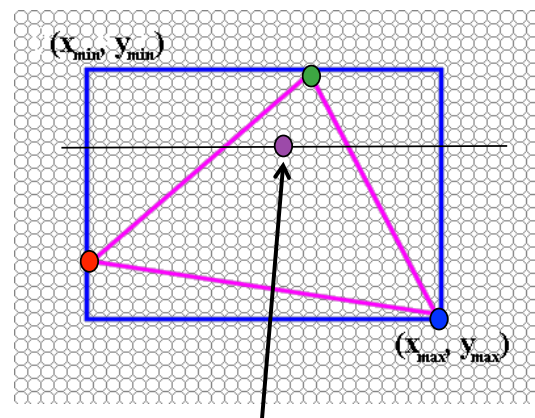
- Vamos renderizar apenas os pixels (na fig abaixo em verde) para os quais as equações dão resultado positivo



Contra quais pixels testar?

# Edge Equations

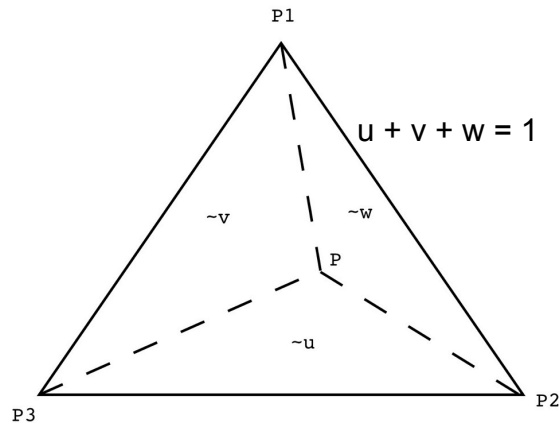
- Calcular a caixa-limite do triângulo
- Calcular as equações das retas a partir dos vértices
  - Ser consistente na definição do lado interno ou positivo
- Para todos os pixels dentro da caixa-limite:
  - “pinta” apenas os pixels para os quais as 3 equações derem resultado positivo



Como calcular a cor do pixel interno se não temos as cores nos extremos da scan line (como acontece com o “edge walking”)?

# Alternativa usando Coord. Baricêntricas

- Ponto P pode ser expresso por  $(u, v, w)$  que é invariante
- Permanecem as mesmas após transformações afim
- Cálculo:



$$u = \frac{\text{area}(PP_2P_3)}{\text{area}(P_1P_2P_3)}$$

$$v = \frac{\text{area}(P_1PP_3)}{\text{area}(P_1P_2P_3)}$$

$$w = \frac{\text{area}(P_1P_2P)}{\text{area}(P_1P_2P_3)}$$

Na revisão matemática vimos em aula uma maneira fácil de encontrar todas estas áreas.

21

# Rasterização com Coord Baricêntricas

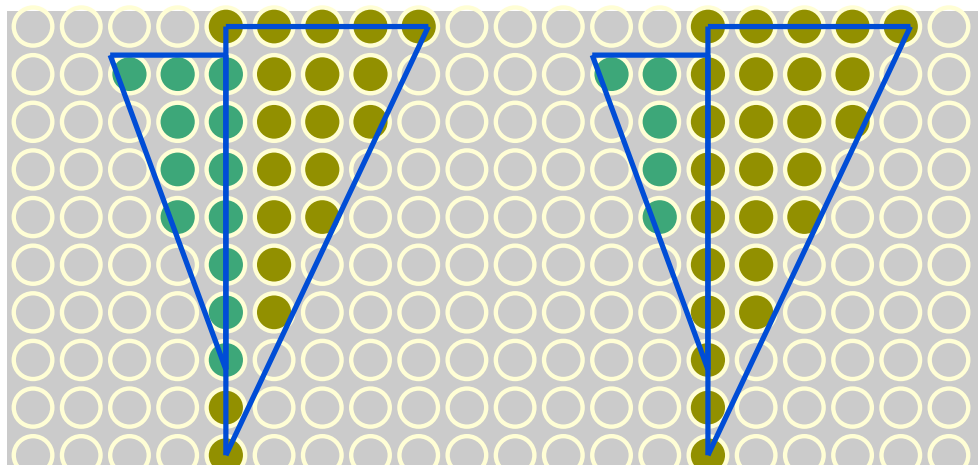
- Calcula caixa-limite
- Para cada pixel nas linhas de *scan* da caixa-limite calcula coordenadas baricêntricas
  - Se a soma das coord baricêntricas = 1 desenha o pixel
  - Senão, ponto é externo ao triângulo

# Problemas: Rasterização de Triângulos

- Exatamente quais pixels devem ser “ligados”?
  - Pixels dentro dos limites das arestas
- E como ficam os pixels exatamente NAS arestas?  
E compartilhados?

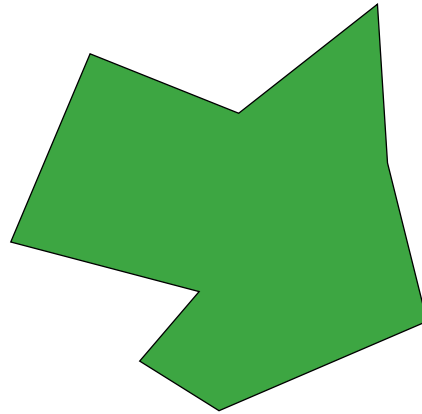
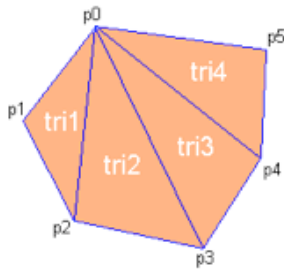
# Problemas: Rasterização de Triângulos

- E como ficam os pixels exatamente NAS arestas?
  - A ordem de desenho não deve importar
  - Utilizar uma regra para manter consistência
  - Exemplo: desenha apenas pixels na aresta esquerda e topo, mas não na direita ou base



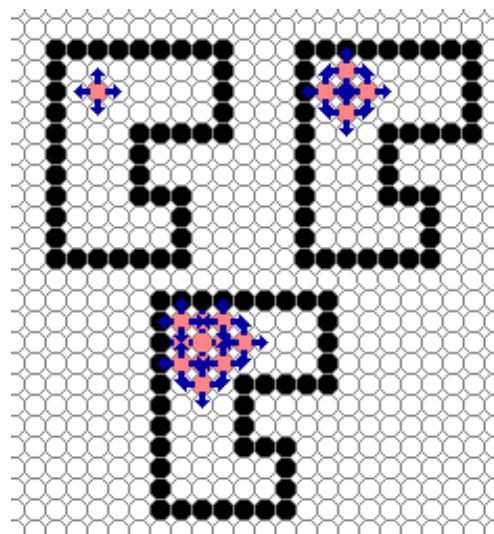
# Rasterização de Polígonos em Geral

- Triangularizar o polígono e utilizar uma das técnicas de rasterização de triângulos
- Rasterizar diretamente o polígono



## Rasterizando diretamente o polígono

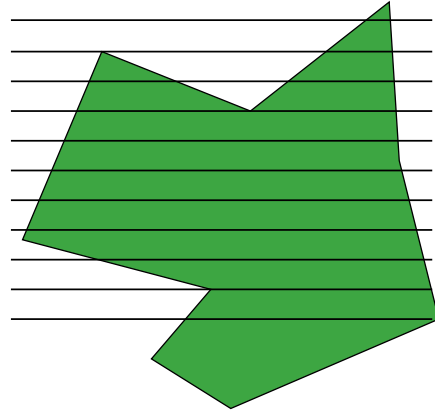
- Métodos de preenchimento
  - Exs: *Boundary Fill*, *Flood Fill*
  - “Atuam” ao nível do pixel
  - Interessantes para aplicações do tipo *painting*
  - A partir de um pixel garantidamente dentro do polígono, visita **recursivamente** os vizinhos até encontrar a borda ou pixels já visitados



# Rasterizando diretamente o polígono

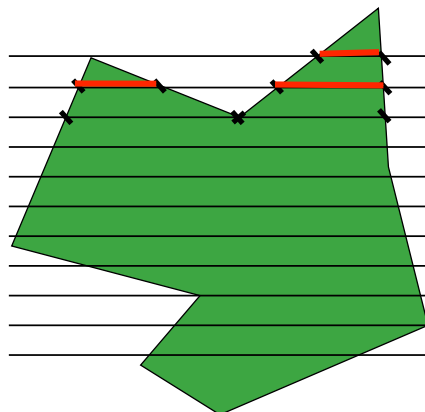
- Métodos Scanline

- “Atuam” no nível dos polígonos, trabalham com a definição geométrica dos objetos
- Melhor performance



## Algoritmos *scanline-fill*

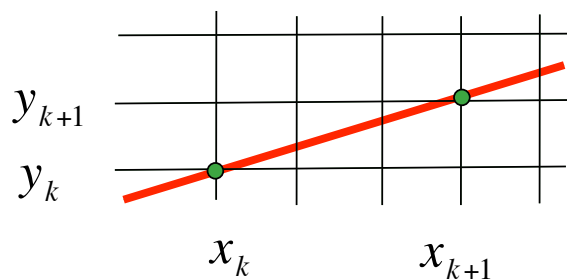
- Calcula a intersecção entre a scanline com as arestas do polígono
- Preencher entre pares de intersecções



Interior determinado  
por pares de  
intersecção

## Algoritmos *scanline-fill*

- Explora coerência (partes se relacionam)
- Inclinação da aresta é constante de uma *scanline* para a próxima
- A intersecção da aresta com o próximo x pode ser obtida incrementalmente. Porque?



## Algoritmos *scanline-fill*

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

$$y_{k+1} - y_k = 1 \leftarrow \text{Diferença entre 2 scanlines}$$

$$m = \frac{1}{x_{k+1} - x_k}$$

$$x_{k+1} = x_k + \frac{1}{m}$$

$$x_{k+1} = x_k + \frac{dx}{dy}$$

# Algoritmos *scanline-fill*

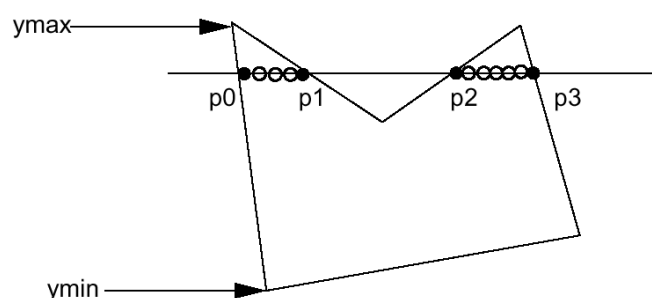
- Algoritmo básico

*for y = ymin to ymax*

*1.intercepta scanline y com cada aresta*

*2.ordena intersecções de acordo com a coordenada x*  
*[p0,p1,p2,p3]*

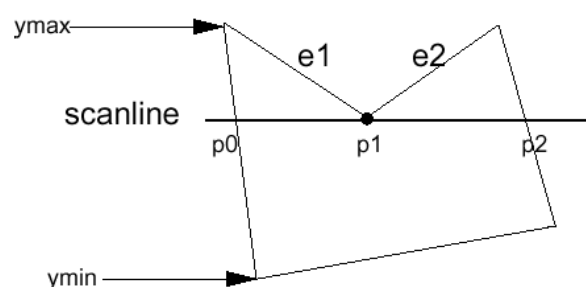
*3.preenche entre os pares de coordenadas*  
*(p0->p1, p2->p3, ...)*



## Algoritmos *scanline*: complicadores

- Intersecção é o ponto final de uma aresta
- Pontos de intersecção: (p0, p1, p2) ???
  - (p0,p1)(p1,p2): tomamos os pares
  - Se computarmos a intersecção da *scanline* com a aresta *e1* e *e2*, pegaremos o ponto *p1* duas vezes:

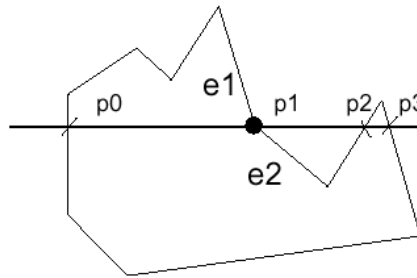
**Caso 1**





# Algoritmos *scanline*: complicadores

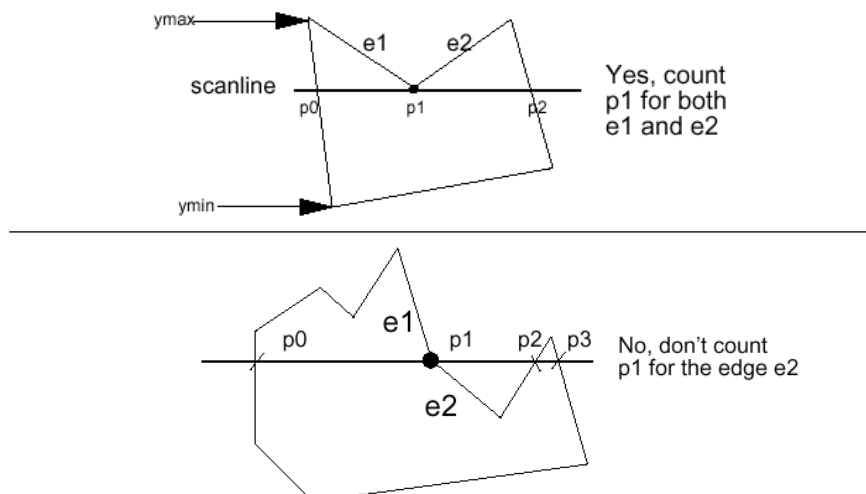
Caso 2



- $(p_0, p_1)(p_1, p_2)(p_2, p_3)$
- Não podemos contar o ponto  $p_1$  duas vezes, pois seriam pintados os pixels entre  $p_1$  e  $p_2$ .
- Deve-se considerar  $(p_0, p_1)(p_2, p_3)$ , ou seja, contar  $p_1$  apenas uma vez

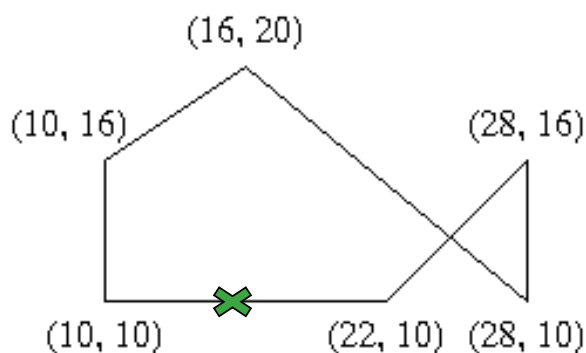
# Algoritmos *scanline*: complicadores

- Conclusão:
  - Se a intersecção corresponder a  $y_{min}$  das arestas, considere-a
  - Caso contrário, não.



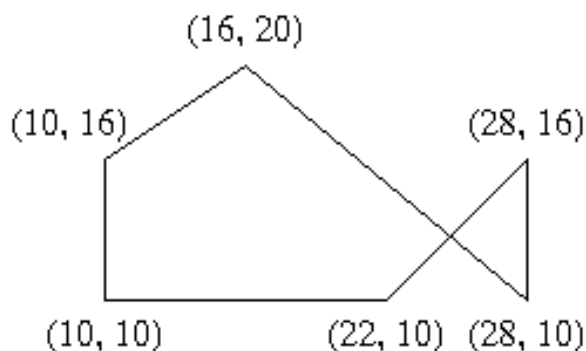
# Algoritmos *scanline*: complicadores

- Arestas horizontais não são resolvidas pelo algoritmo, tratadas em separado



## Exemplo do algoritmo

- Iniciar a tabela de arestas.
- Determinar para cada aresta:  $y_{min}$ ,  $y_{max}$ ; o valor de interseção de  $x$  (para o menor  $y$ ); e  $1/m$



ordered\_vertices

0	(10, 10)
1	(10, 16)
2	(16, 20)
3	(28, 10)
4	(28, 16)
5	(22, 10)

# Tabela Global de Arestas

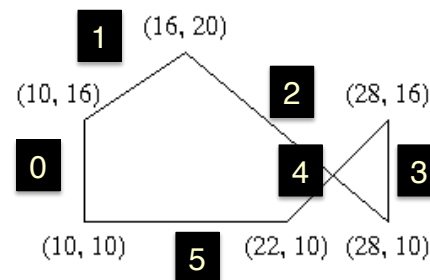
all\_edges

0	●	→	10	16	10	0
1	●	→	16	20	10	1.5
2	●	→	10	20	28	-1.2
3	●	→	10	16	28	0
4	●	→	10	16	22	1
5	●	→	10	10	10	inf

Y-min      Y-max      X-val      1/m

- A montagem da tabela global de arestas deve:

- Ordenar em ordem crescente em função de ymin e x
- Eliminar as arestas horizontais (m=0)

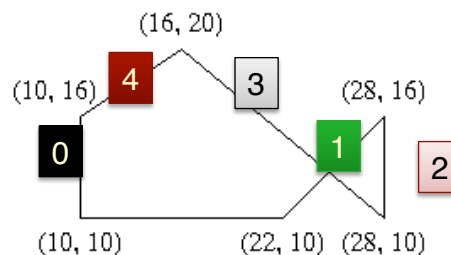


# Tabela Global de Arestas

global

0	●	→	10	16	10	0
1	●	→	10	16	22	1
2	●	→	10	16	28	0
3	●	→	10	20	28	-1.2
4	●	→	16	20	10	1.5

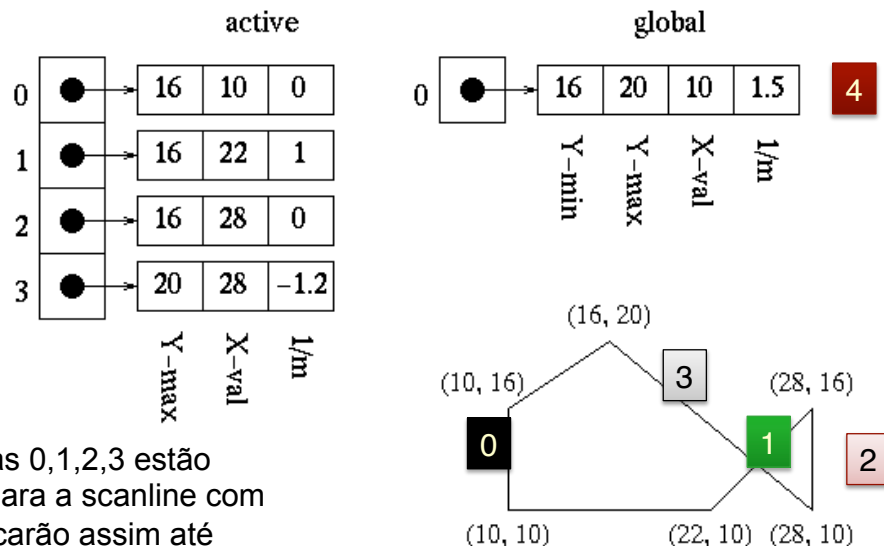
Y-min      Y-max      X-val      1/m



A seguir, será construída a lista de arestas ativas (*active edge list*) para cada linha de scanline. Esta lista contém todas as arestas que são cortadas por uma linha (*scanline*)

# Lista de Arestas Ativas

Como a lista global começa com  $y=10$ , montamos a lista de arestas ativas com todas arestas com  $y = 10$ . Resta, na tabela global, a aresta que inicia em  $y=16$ .



As arestas 0,1,2,3 estão "ativas" para a scanline com  $y=10$  e ficarão assim até  $y=16$ .

## Algoritmo

- A cada  $y$ :
  - O algoritmo ordena os "x-val" das arestas na lista de arestas ativas e preenche os trechos da scan line entre pares de  $x$
  - Atualiza x-val somando o incremento
  - Verifica se há arestas que estão terminando nessa linha de varredura ( $y = y_{\max}$ ), para excluí-las da lista de arestas ativas
- Incrementa  $y$  para o próximo passo, e verifica se há arestas iniciando nessa nova linha, incluindo-as na lista de arestas ativas