

## Preenchimento de Áreas

### Algoritmo *Scanline*

Fontes: Hearn & Baker, Cap. 33

Curso CG, University of Leeds (Ken Brodlie):

<http://www.comp.leeds.ac.uk/kwb/gi21/lectures.html>

Apostila CG, Cap. 4

## Preenchimento de Áreas

- Problema de conversão matricial de áreas geométricas
  - Aproximar uma primitiva geométrica por pixels
- Primitivas 2D (quadrado, retângulo, círculo, ...)
- Polígonos em geral (2D)
- O último é o que nos interessa mais... porque?

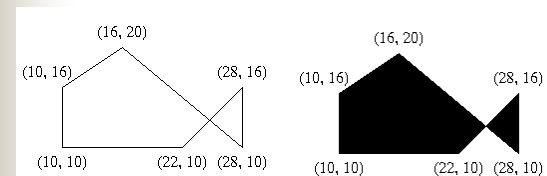
2

## Preenchimento de Áreas

- Assim como no traçado de linhas e curvas, precisamos de um mecanismo eficiente de determinar quais pixels estão no interior de uma certa área, e devem ser 'pintados'
- Tipicamente, solução é implementada em hardware
- Mas, como determinar o que está no 'interior' de uma área?

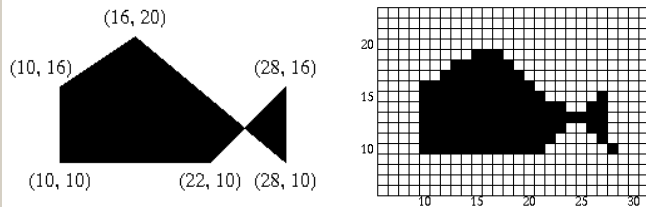
3

## Preenchimento



4

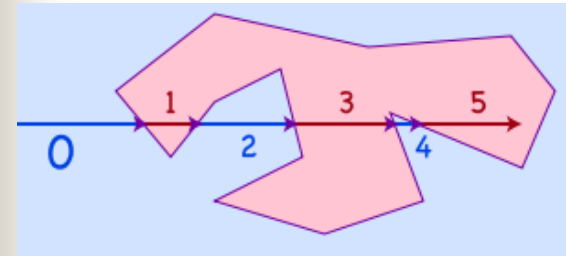
## Preenchimento



Como determinar quais pixels estão 'dentro' do polígono?

5

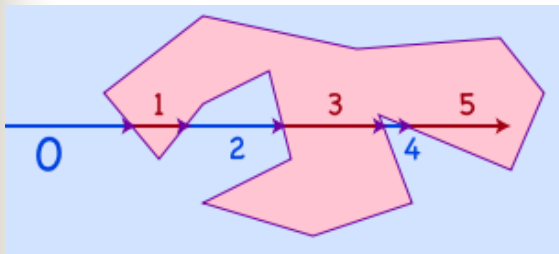
## Teste dentro-fora



Dado um polígono qqr e um ponto, como saber se o ponto está no interior ou no exterior do polígono?

6

## Teste dentro-fora



Quando o número de arestas do polígono interceptadas é ímpar, está dentro; quando é par, está fora...

Regra da paridade

7

## Determinando Interior

- Problema de preenchimento é resolvido para cada linha de varredura da grade de pixels
- Usando a 'Regra da Paridade'
  - Paridade inicialmente par, a cada intersecção da linha de varredura com uma aresta do polígono a paridade é invertida
  - Ponto está no interior do (está 'dentro') quando a paridade é ímpar, nesse caso ele é pintado; e não é pintado (está fora) quando a paridade é par

8

## Determinando Interior

- Entretanto, nem sempre é tão óbvio!
- E quando a linha de varredura intercepta um vértice? Duas arestas interceptadas...
- Fig. 4.21, Hearn & Baker
- Linha  $y'$ : precisa contar as 2 intersecções
- Linha  $y$ : precisa contar só uma...
  - Convenção: percorre a fronteira do polígono na direção horária (ou anti-horária): se as coordenadas  $y$  das 3 extremidades de duas arestas consecutivas aumentam ou diminuem monotonicamente, isso indica que o vértice compartilhado deve ser contado uma só vez

9

## Calculando intersecções

- Precisa achar a intersecção entre a linha de varredura e cada aresta do polígono
  - Equação da linha de varredura  $i$ :  $y = i$
  - Equação de uma aresta, dados os seus 2 vértices  $(x_i, y_i), (x_f, y_f)$ :

$$m = \frac{dy}{dx} = \frac{(y_f - y_i)}{(x_f - x_i)} \quad y = mx + b$$

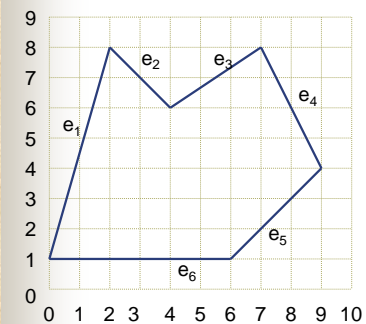
10

## Calculando intersecções

- Note, entretanto, que pode-se aproveitar a coerência do processo para simplificar os cálculos...
- Fig. 4.23, Hearn & Baker
- Entre duas linhas de varredura consecutivas  $y(k+1) = y(k) + 1$
- Portanto, a coordenada  $x$  da intersecção é obtida facilmente;
  - $x(k+1) = x(k) + 1/m$ , i.e.;
  - $x(k+1) = x + dx / dy$

11

## Exemplo



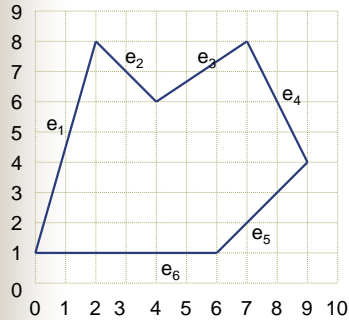
Este polígono tem como vértices:

(0,1)  
(2,8)  
(4,6)  
(7,8)  
(9,4)  
(6,1)

e 6 arestas  
 $e_1, \dots, e_6$

12

## Algoritmo Scanline



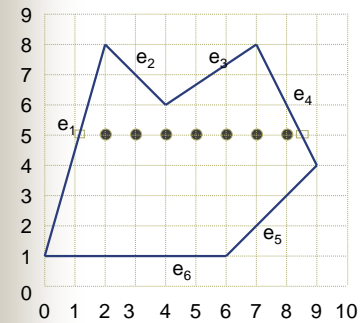
Para preencher a área, processa as linhas de varredura (*scanlines*)

Para um dado valor  $y$ ; acha intersecções com arestas

Pares de intersecções sucessivas definem um bloco (*span*) de pixels

13

## Casos a tratar: 1. normal

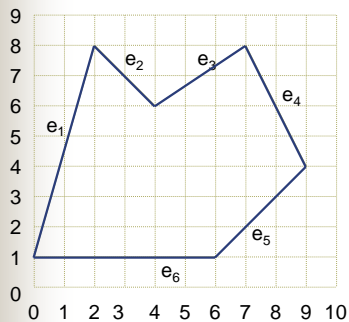


Scanline 5:  
intersecção  $e_1 = 1.14$   
intersecção  $e_4 = 8.5$

Pixels 2-8 na *scanline* são preenchidos

14

## Casos a tratar: 2. Intercepta vértices

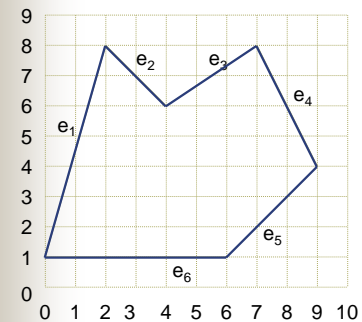


*scanline* 6 intercepta  $e_1, e_2, e_3, e_4$  em 1.4, 4, 4, 8 respectiva/e - dois spans traçados

*scanline* 4 intercepta  $e_1, e_4, e_5$  em 0.9, 9, 9 respectiva/e - linha intercepta duas arestas, considera só a superior, i.e.  $e_4$

15

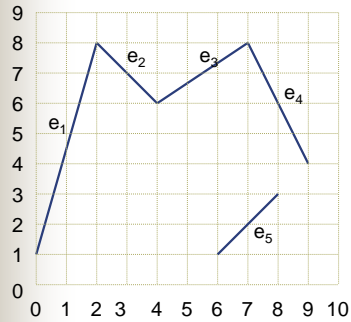
## Casos a tratar: 3. Arestas horizontais



Uma aresta horizontal, como  $e_6$  pode ser ignorada: vai ser traçada automaticamente

16

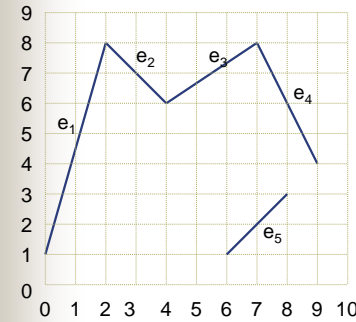
## Pré-Processamento do Polígono



Removemos  $e_6$ , e encurtamos  $e_5$  em uma *scanline*

17

## Otimização

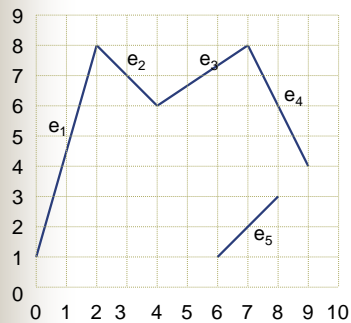


Para acelerar os cálculos de intersecções é útil saber aonde começam/terminam as arestas: e.g.,  $e_1$  vai da *scanline* 1 até a *scanline* 8;  $e_2$  da *scanline* 8 até a 6

Assim, sabemos que arestas testar para cada *scanline*

18

## Cestos Ordenados de Arestas

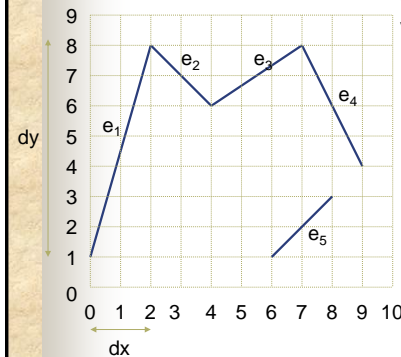


Interessante ordenar as arestas pelo seu ponto de mínimo; cada *scanline* é associada a um cesto de arestas, ordenado segundo a coord. x da intersecção (**bucket sort**)

- 0:
- 1:  $e_1, e_5$
- 2:
- 3:
- 4:  $e_4$
- 5:
- 6:  $e_2, e_3$
- 7:
- 8:
- 9:

19

## Otimização - Coerência

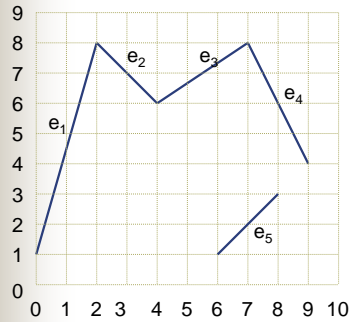


Coerência de linhas de varredura: *scanline coherence*

Ex.: observe  $e_1$   
Assuma intersecção com linha 1 conhecida ( $x=0$ )  
- então, intersecção com *scanline* 2 é:  
 $x^* = x + 1/m$  ( $m=\text{inclin.}$ )  
i.e.:  $x^* = x + dx / dy$   
i.e.:  $x^* = 0 + 2 / 7 = 2/7$

20

## Tabela de Arestas



Útil armazenar informação sobre arestas em uma tabela:

e	prim x	max y	dx	dy
1	0	8	2	7
2	4	8	-2	2
3				
4				
5				

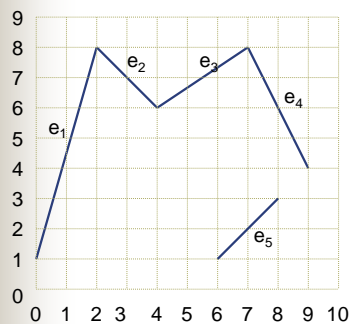
21

## Estrutura de dados

- Tabela de arestas global: descreve as arestas do polígono a serem processadas explicitamente
  - Um cesto para cada linha de varredura
  - Mantém-se ao longo de todo o processamento
- Tabela de arestas ativas (as as que estão sendo interceptadas pela linha de varredura corrente)
  - É atualizada a medida que a varredura da cena prossegue

22

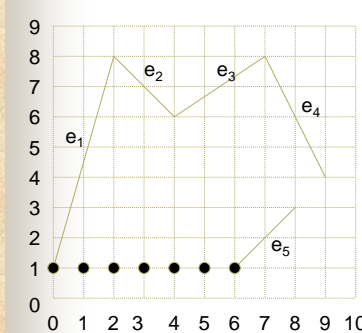
## Algoritmo de Preenchimento



- (1) Set  $y = 0$
- (2) Verifique y-bucket
- (3) Insira as arestas necessárias na **active edge table (AET)**
- (4) If AET vazia then set  $y = y + 1$ , go to (2)

23

## Tabela de Arestas Ativas



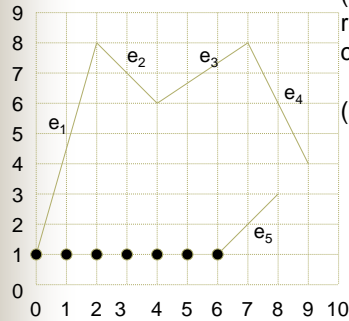
$y = 1$

e	x	max y	dx	dy
1	0	8	2	7
5	6	3	2	2

- (5) Ordene por valores-x e preencha entre pares sucessivos

24

## Tabela de Arestas Ativas



(6) Set  $y = y+1$ ;  
 remova da AET arestas  
 com  $\max y \leq y$

(7) Incrementa  $x$  de  $1/m$  ( $dx/dy$ )

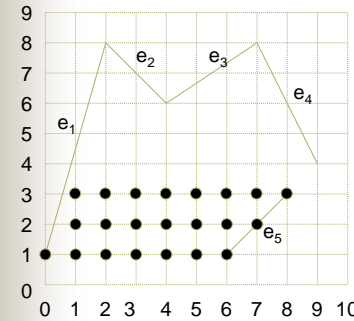
$y = 2$

e	x	max y	dx	dy
1	2/7	8	2	7
5	7	3	2	2

(8) Retorna para (2)

25

## Próxima Scanline



A cada estágio do  
 algoritmo, um (ou vários)  
*span(s)* de pixels são  
 traçados

26

## Eficiência – aritmética inteira

- Por questão de eficiência, é interessante usar apenas aritmética inteira – isso requer uma coluna extra na AET para acumularmos separadamente a parte inteira da parte fracionária de  $1/m$  para a atualização de  $x$

27

## Eficiência – aritmética inteira

Primeira intersecção é em  $x=0$ , e a inclinação da aresta é  $7/2$  – i.e.  $dy=7$ ,  $dx=2$

Os próximos pontos de intersecção são:

0      2/7      4/7      6/7      8/7    etc.

Os pixels iniciais correspondentes para o *scanline filling* são:

0      1      1      1      2

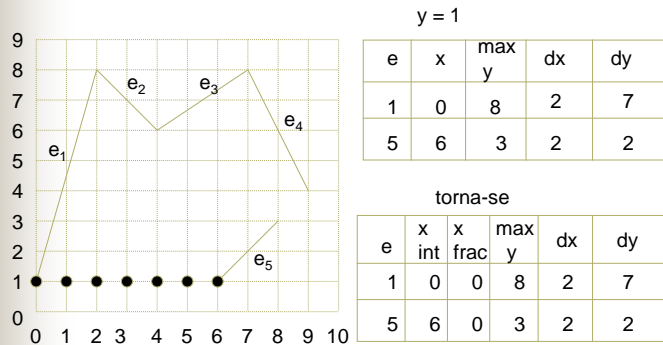
Obtemos esses valores simplesmente somando  $dx$  a cada estágio, até que  $dy$  é atingido, então  $dx$  é reduzido de  $dy$ :

0      2      4      6      1 (8-7) etc.

28

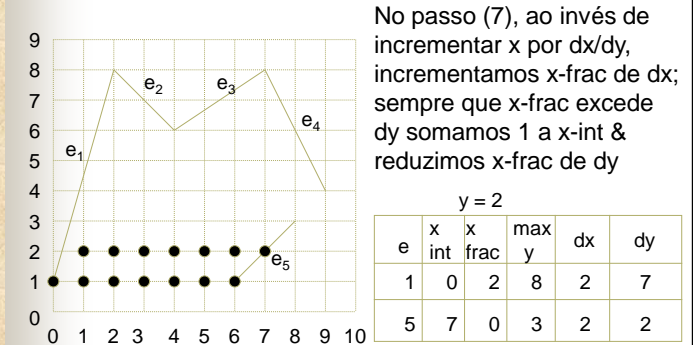


## Tabela de Arestas Ativas Modificada



29

## Eficiência – aritmética inteira



30

## Observação

- No caso de preenchimento de áreas, cada pixel está sendo 'pintado' com uma cor
- No caso de *rendering* de superfícies, cada pixel é pintado com a cor determinada pela aplicação do algoritmo de iluminação + tonalização (*shading*)

31

## Bibliografia

- Hearn & Baker, 4.10
- Ap. CG Cap. 4
- <http://www.cs.rit.edu/~icss571/filling/example.html>

32