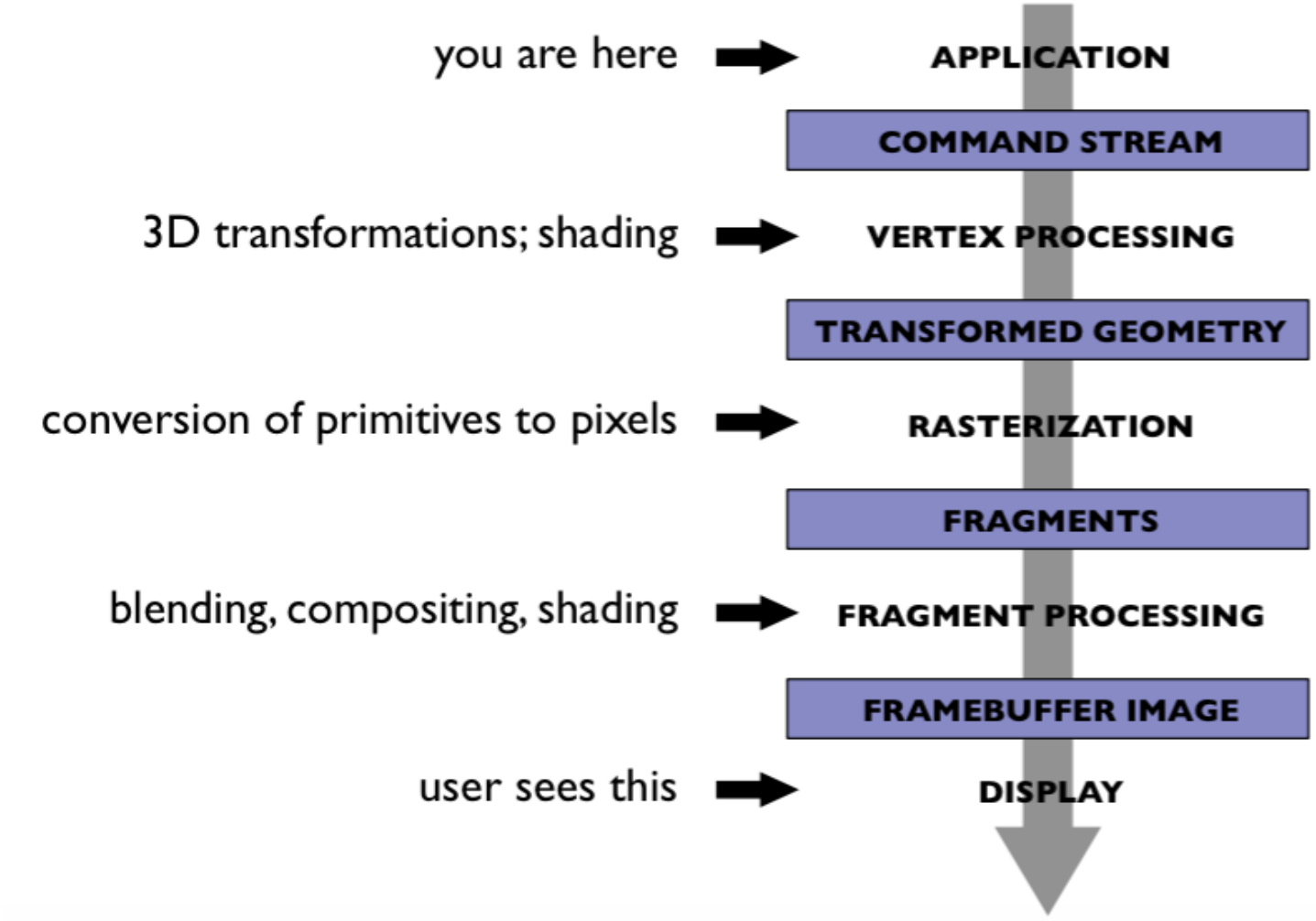
MAC420/5744: Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #12: Rasterização

Some slides by Steve Marschner, Cornell CS4620/5620

Pipeline


you are here
APPLICATION
COMMAND STREAM
3D transformations; shading
VERTEX PROCESSING
TRANSFORMED GEOMETRY
conversion of primitives to pixels
RASTERIZATION
FRAGMENTS
blending, compositing, shading
FRAGMENT PROCESSING
FRAMEBUFFER IMAGE
user sees this
DISPLAY

Primitivas

- Points
- Line segments
  - and chains of connected line segments
- Triangles
- And that's all!
  - Curves? Approximate them with chains of line segments
  - Polygons? Break them up into triangles
  - Curved regions? Approximate them with triangles
- Trend has been toward minimal primitives
  - simple, uniform, repetitive: good for parallelism

Rasterização

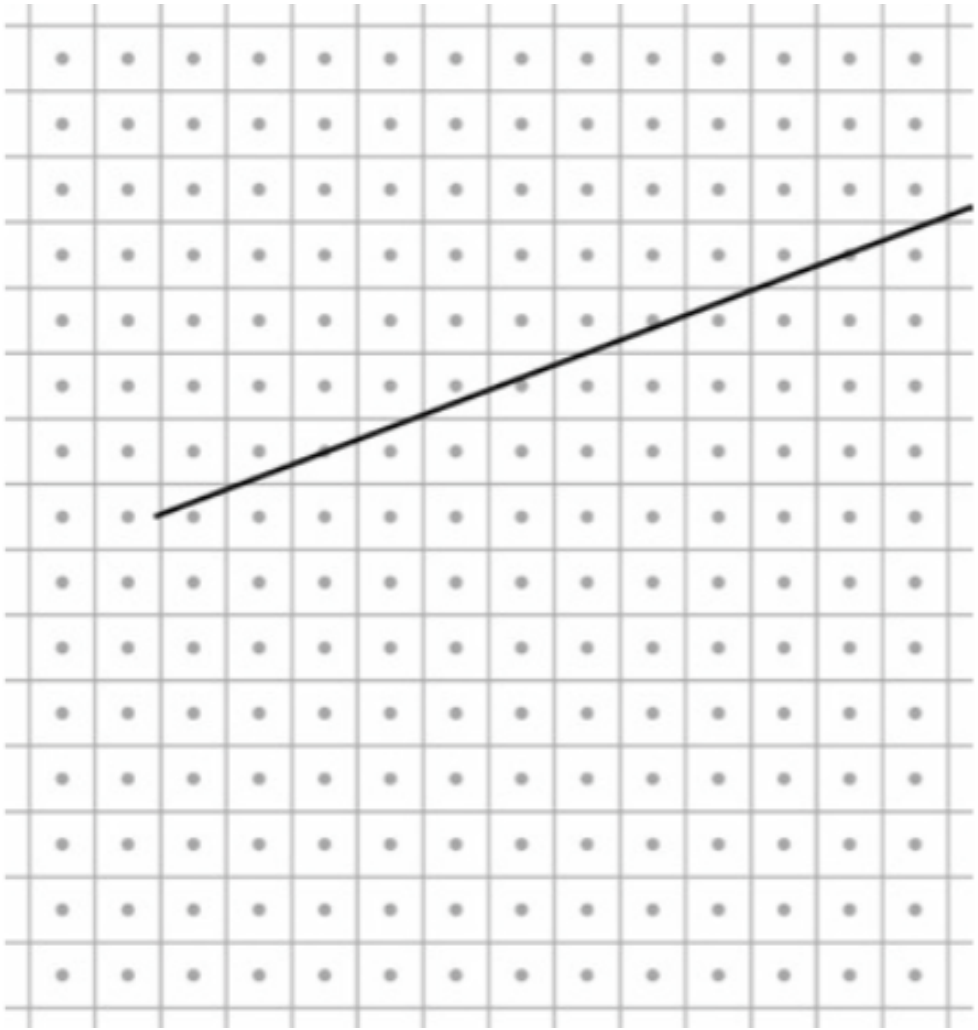
- Rasterização ("scan conversion")
  - Determina quais pixels que pertencem a primitiva especificada através de vértices
  - Produz um conjunto de fragmentos
- Fragmentos possuem uma posição e outros atributos como cor e coordenadas de textura que são interpolados entre os vértices
- Cores finais dos pixels são determinadas usando cor, textura e outras propriedades dos vértices.

Rasterização

- First job: enumerate the pixels covered by a primitive
  - simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across the primitive
  - e.g. colors computed at vertices
  - e.g. normals at vertices

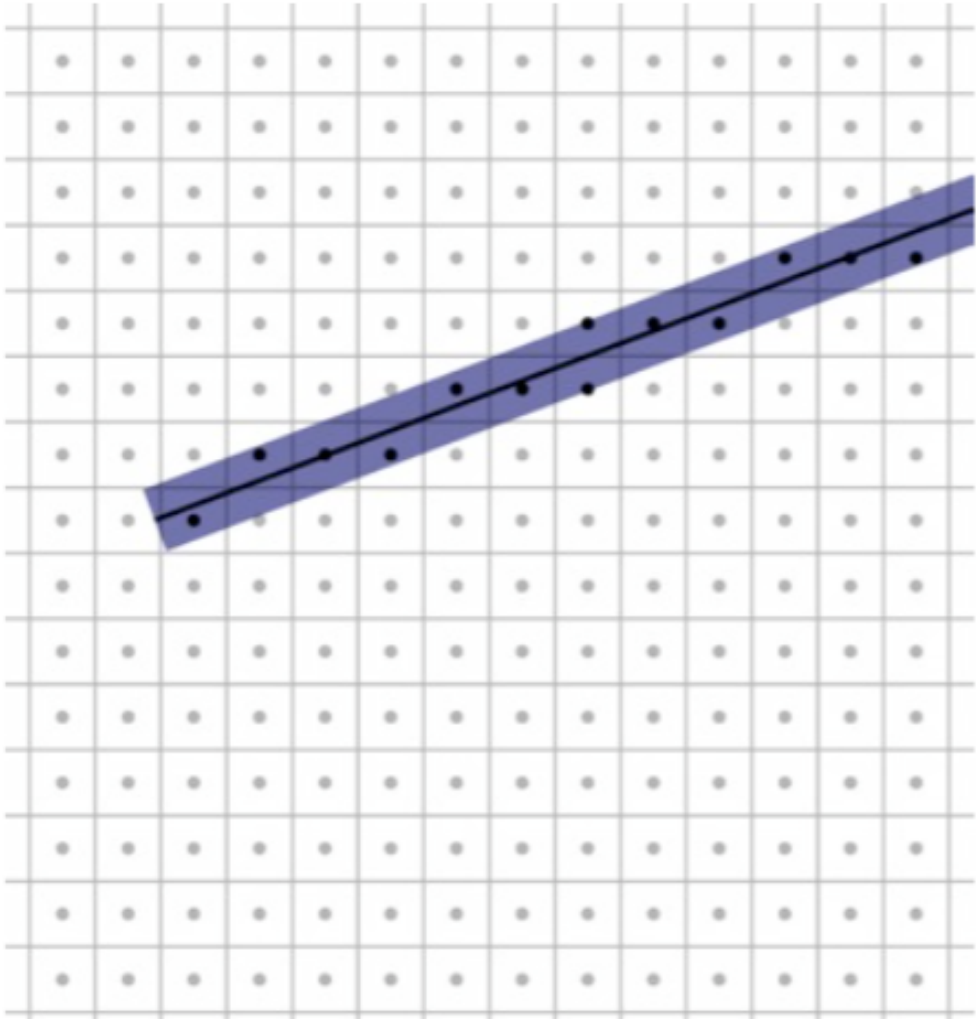
Rasterização retas

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



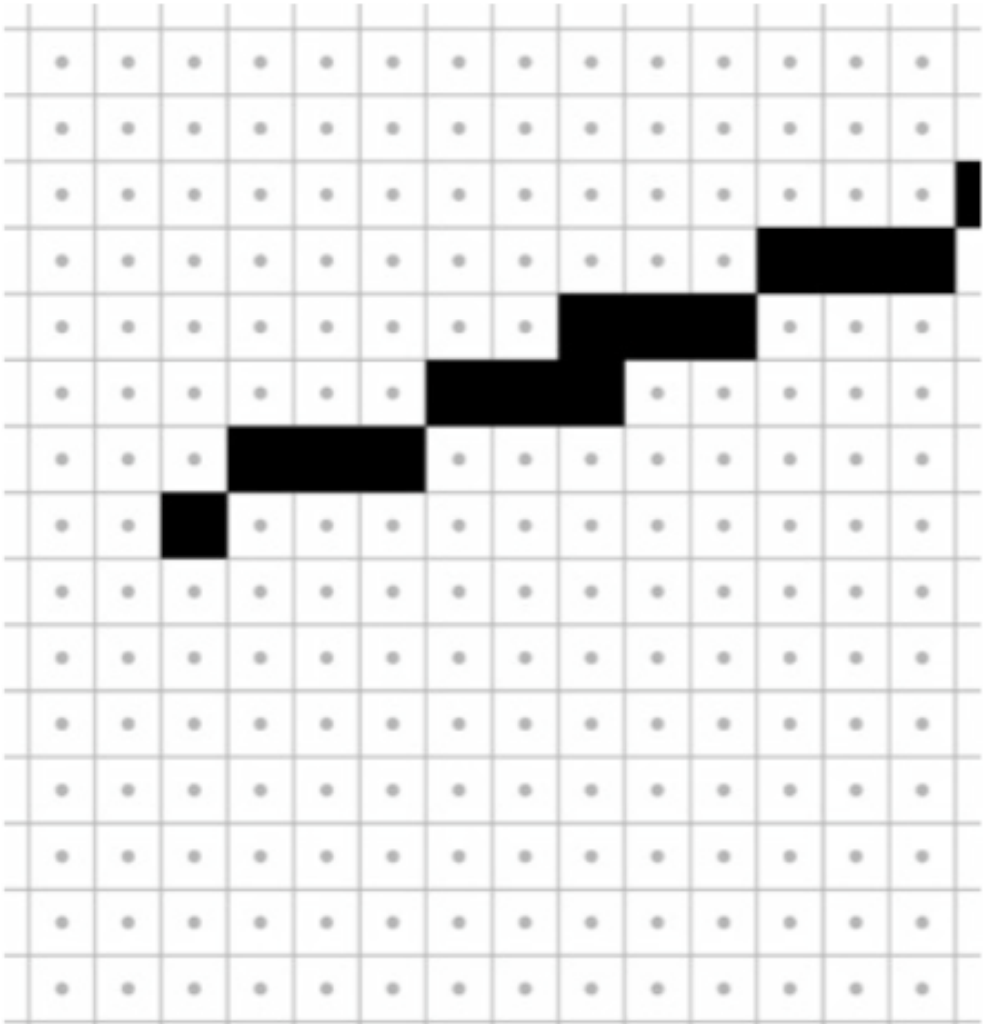
Rasterização de retas

- Approximate rectangle by drawing all pixels whose centers fall within the line

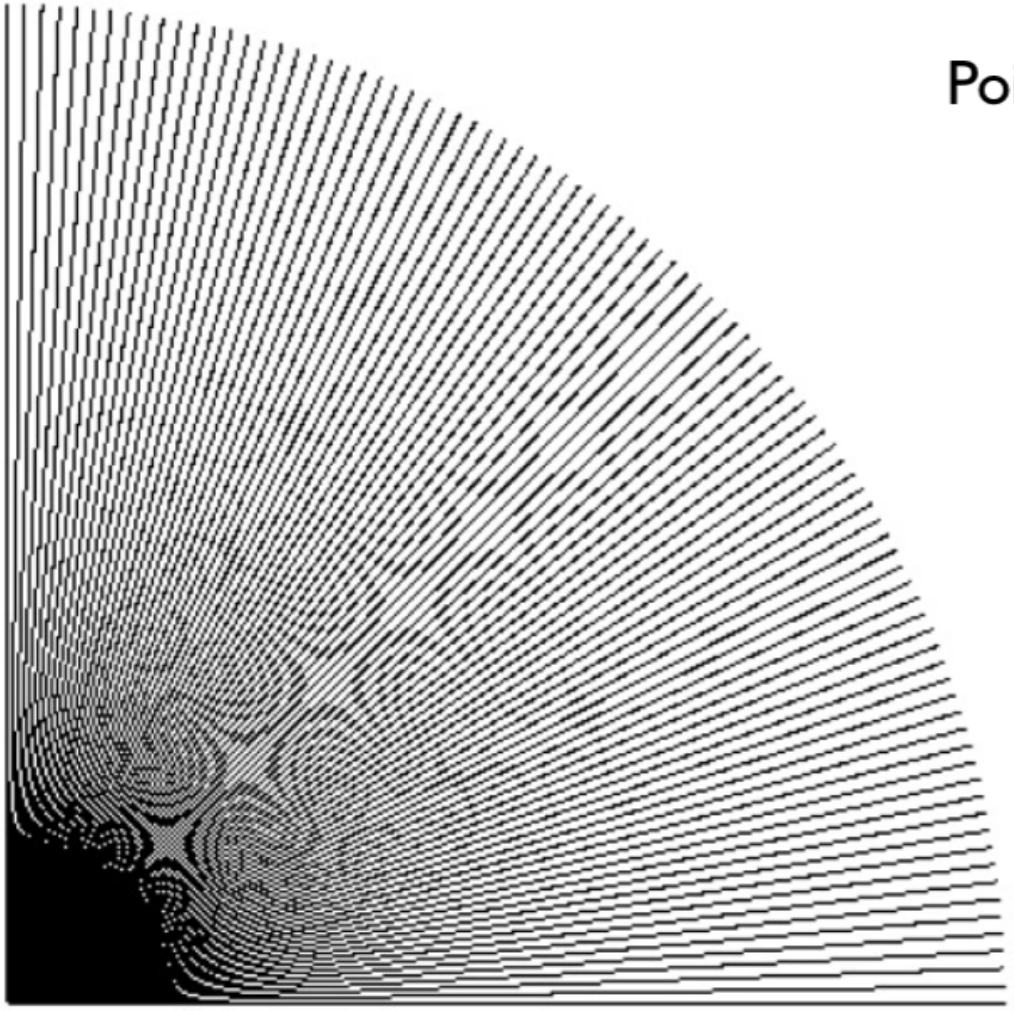


Amostragem de pontos

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: sometimes turns on adjacent pixels



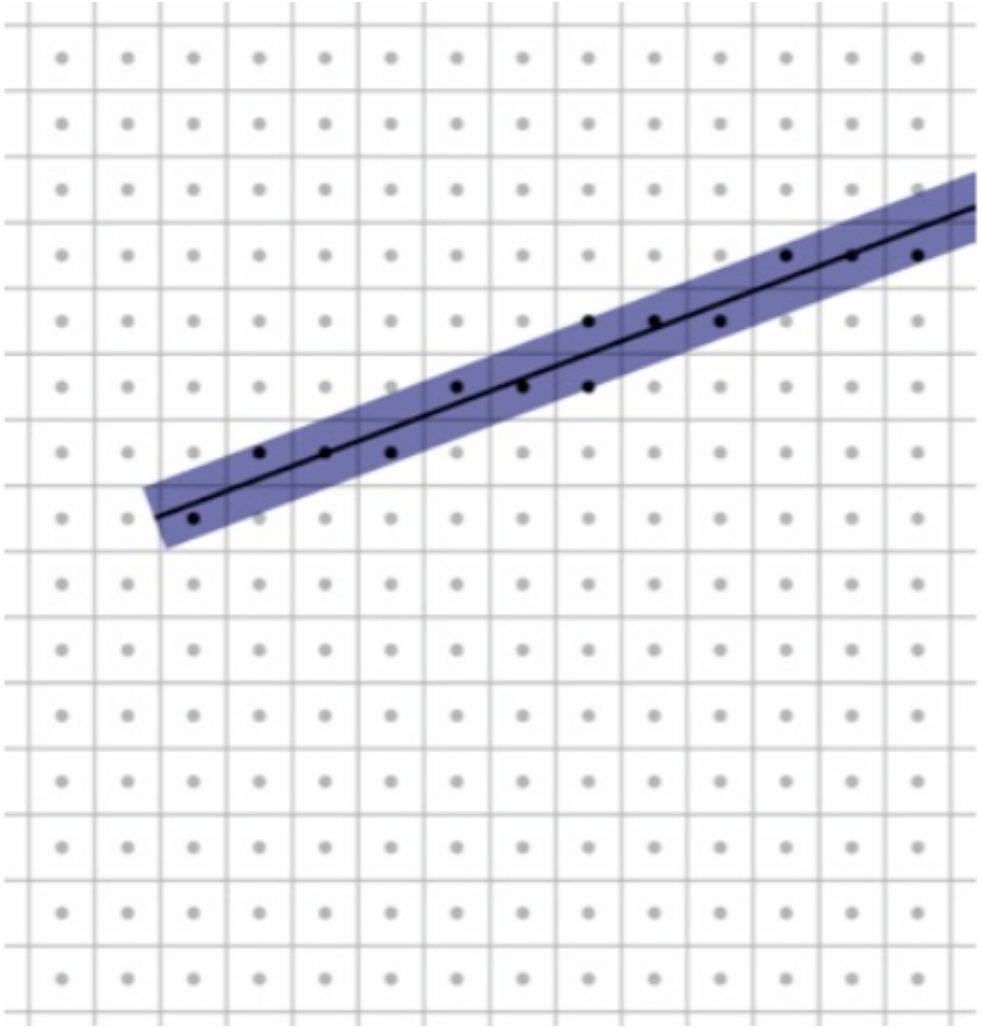
Amostragem de pontos



Point sampling
in action

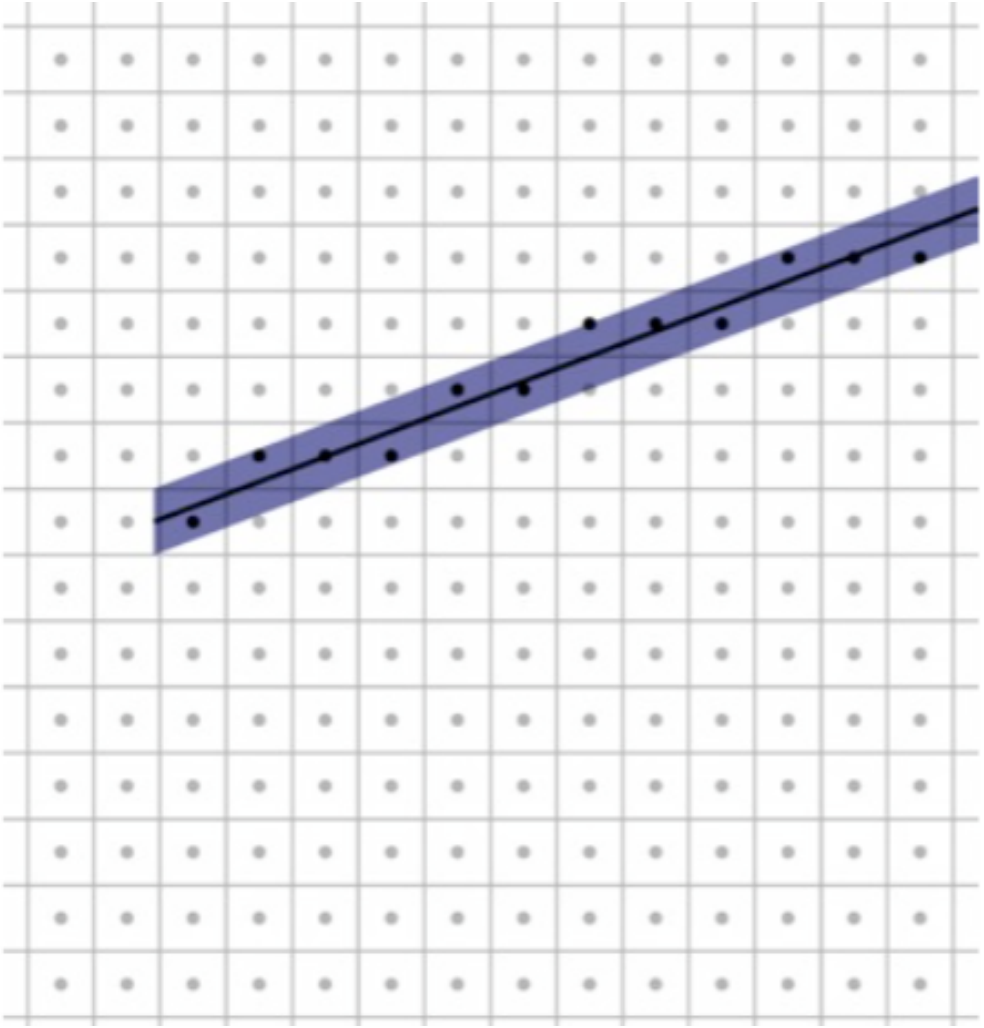
Bresenham (Midpoint)

- Point sampling unit width rectangle leads to uneven line width
- Define line width parallel to pixel grid



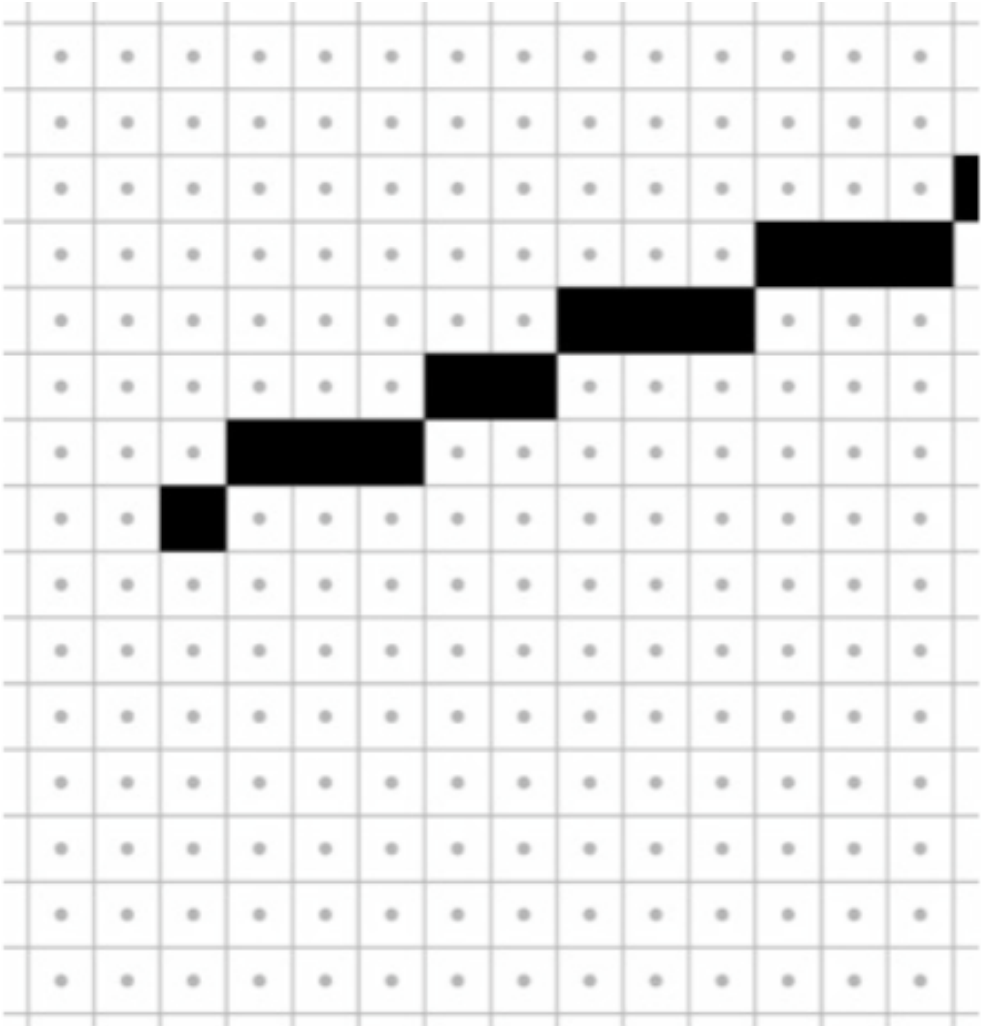
Bresenham (Midpoint)

- Point sampling unit width rectangle leads to uneven line width
- Define line width parallel to pixel grid
- That is, turn on the single nearest pixel in each column

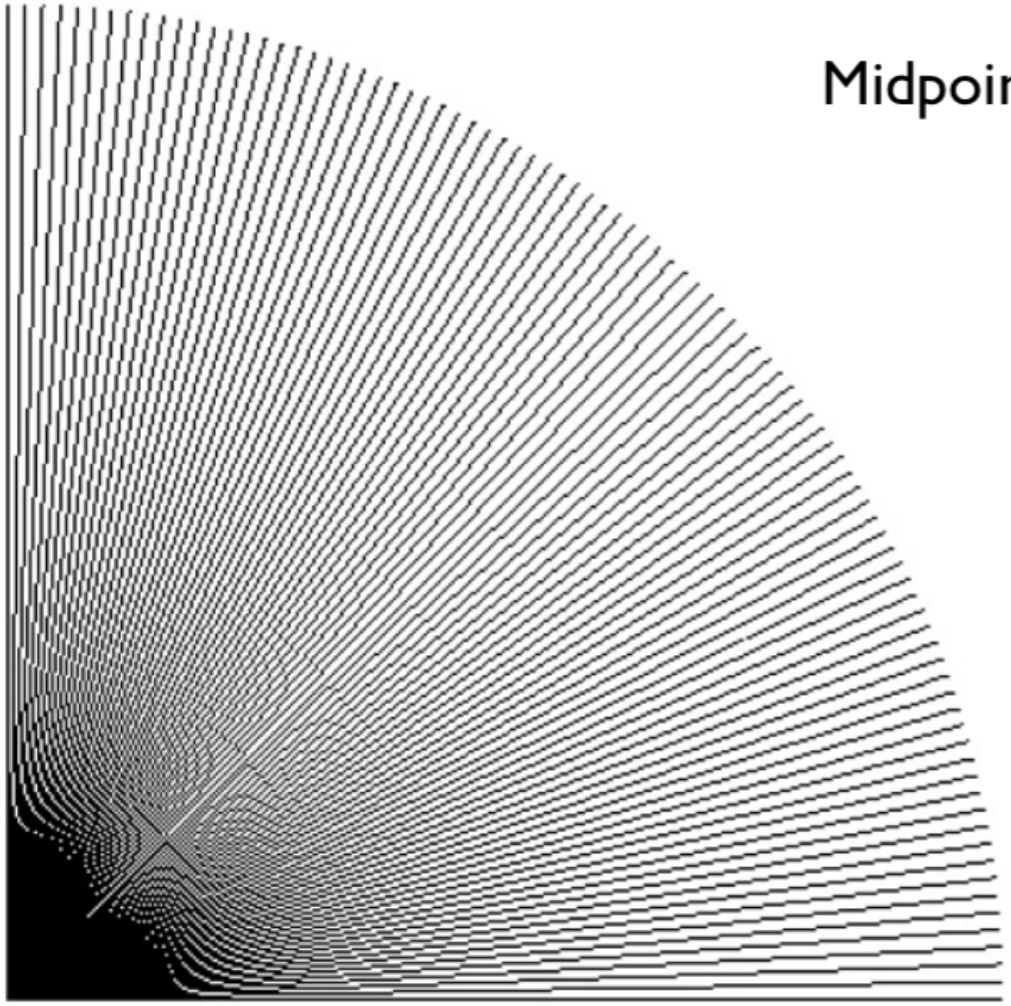


Bresenham (Midpoint)

- Point sampling unit width rectangle leads to uneven line width
- Define line width parallel to pixel grid
- That is, turn on the single nearest pixel in each column
- Note that 45° lines are now thinner

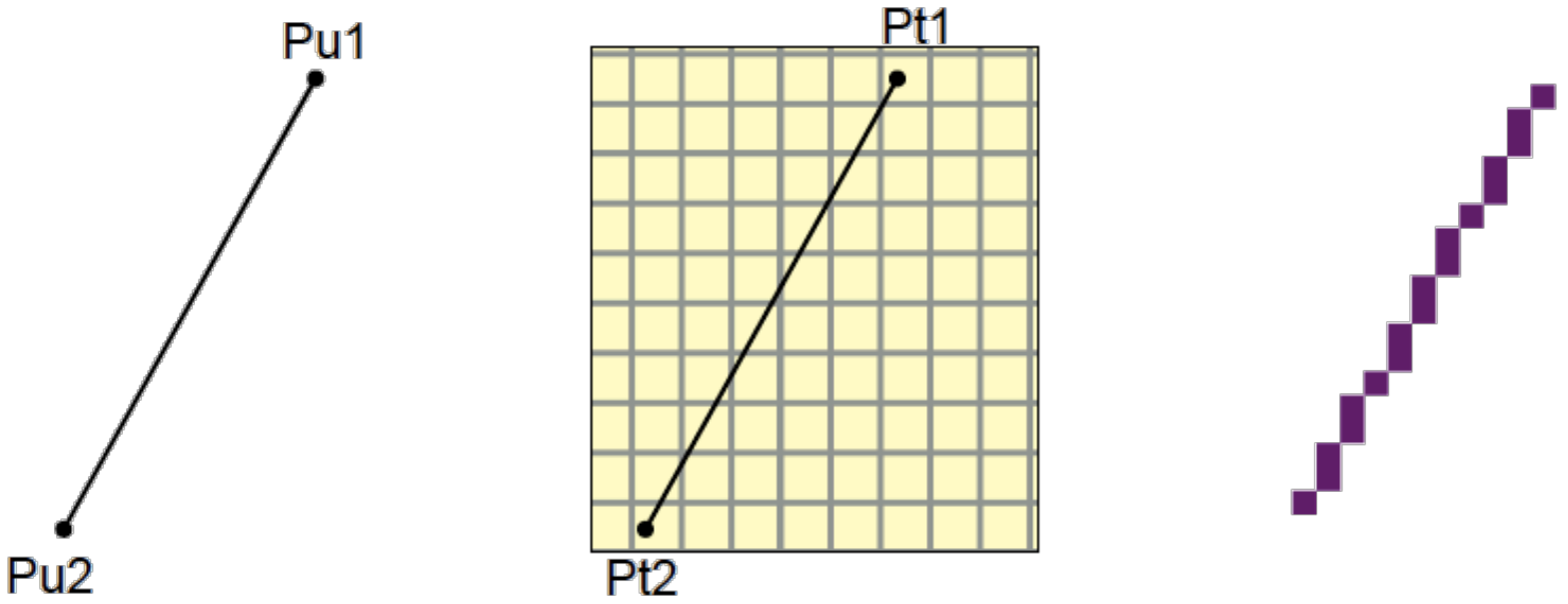


Bresenham



Midpoint algorithm
in action

Desenho de retas



Algoritmo básico

- Dados pontos extremos da linha na tela

  Pt1= (x1, y1), Pt2= (x2, y2)

- Calcula coeficientes da equação da reta:

$$y = mx + b \qquad m = \frac{y2 - y1}{x2 - x1} \qquad b = y1 - m \cdot x1$$

- Liga todos os pixels que pertencem à reta

```
for x = x1 to x2
    y = m * x + b
    ponto(x,y)
```

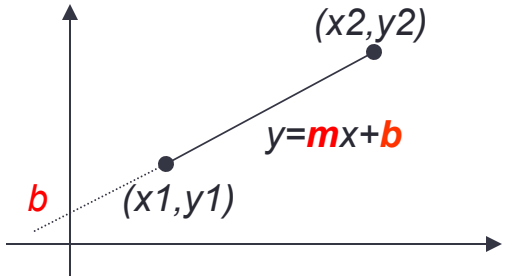
Utilizando a equação da reta

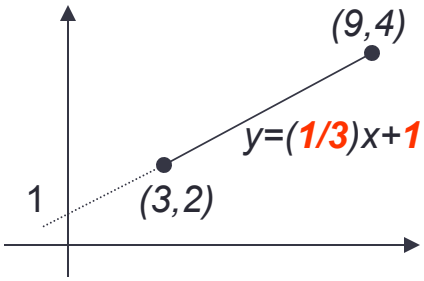
$y_i = m\,x_i + b$
onde:

$m = \Delta y/\Delta x$
$b = y_1 - m\,x_1$

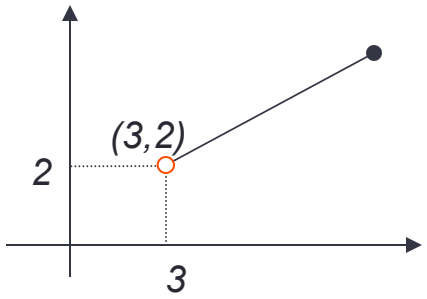
```
void Line1(int x1, int y1, int x2, int y2, int color)
{
  float m = (y2-y1)/(x2-x1);
  float b = y1 - m*x1;
  float y;
  int x;

  write_pixel(x1,y1,color);

  for (x=x1+1; x<=x2; x++)
  {
     y = m*x + b;
     write_pixel(x,round(y), color);
  }
}
```

Exemplo

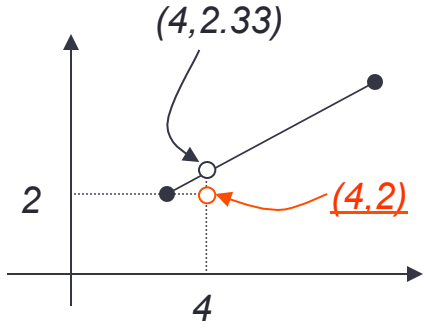




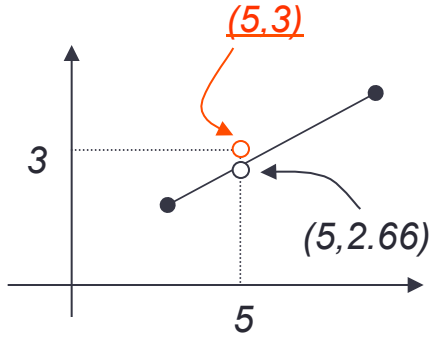
Calcula-se m e b



1ª iteração



2ª iteração



3ª iteração

Desvantagens

```
void Line2(int x1, int y1, int x2, int y2, int color)
{
  float m = (y2-y1)/(x2-x1);
  float b = y1 - m*x1;
  float y;
  int x;

  write_pixel(x1,y1,color);

  for (x=x1+1; x<=x2; x++)
  {
     y = m*x + b;
     write_pixel(x,round(y), color);
  }
}
```

① multiplicação

② adição

③ arredondamento

Como tirar a multiplicação?

Algoritmo básico (ii)

- Da equação paramétrica da reta
  - P = Pt1 + t * (Pt2 - Pt1)
  - t variando de 0 a 1

- Chega-se nas seguintes expressões
  - x = x1 + t * (x2-x1)
  - y = y1 + t * (y2-y1)

Algoritmo básico (ii)

```
y = y1;
x = x1;
for t = 0 to 1
    ponto(x,y);
    y = y1 + t * (y2-y1);
    x = x1 + t * (x2-x1);
```

- 2 operações de ponto flutuante por pixel
- 2 multiplicações por pixel
- Como melhorar?

Algoritmo DDA

- Digital Differential Analyzer
  - DDA foi uma máquina para resolver equações diferenciais de forma numérica
  - A linha y=mx + b satisfaz a equação diferencial

    $dy/dx = m = Dy/Dx = (y_2-y_1)/(x_2-x_1)$
- Podemos desenhá-la para cada passo Dx

```
for(x=x1; x<=x2, x++) {
    y+=m;
  write_pixel(x, round(y), line_color)
}
```

Algoritmo DDA

Se

$x_{i+1} = x_i + 1$

então

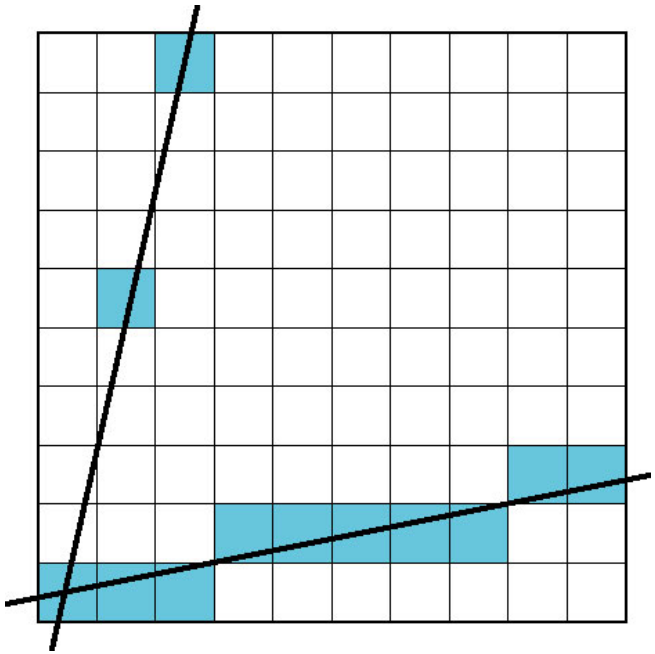
$y_{i+1} = y_i + \overbrace{\Delta y/\Delta x}^{m}$

```
void LineDDA(int x1, int y1, int x2, int y2, int color)
{
  float y;
  float m = (y2-y1)/(x2-x1);
  int x;

  write_pixel(x1,y1, color);
  y = y1;

  for (x=x1+1; x<=x2; x++)
  {
    y += m;
    write_pixel(x,round(y), color);
  }
}
```

- Ainda são necessários uma adição de floats e um arredondamento.
- Como melhorar ?

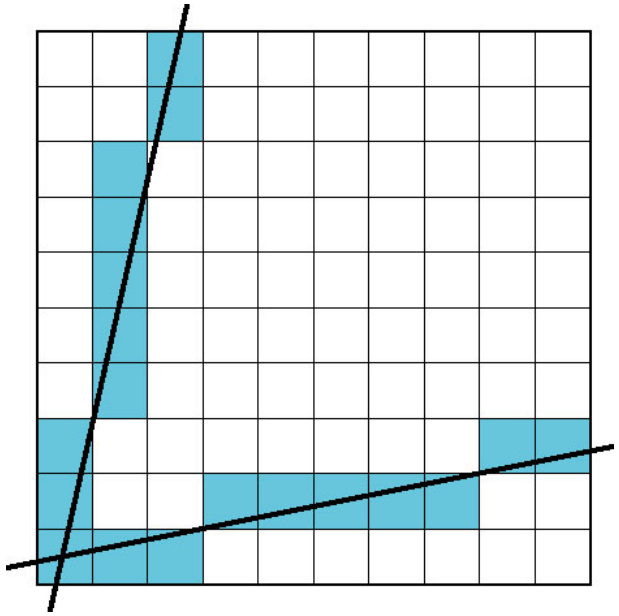
Problemas

- No DDA, para cada unidade em **x**, colore-se o **y** mais próximo
  - Linhas com alta inclinação não são corretamente desenhadas



Simetria

- Para m > 1, troque a função de x e y
  - Para cada y, desenhe o x mais próximo



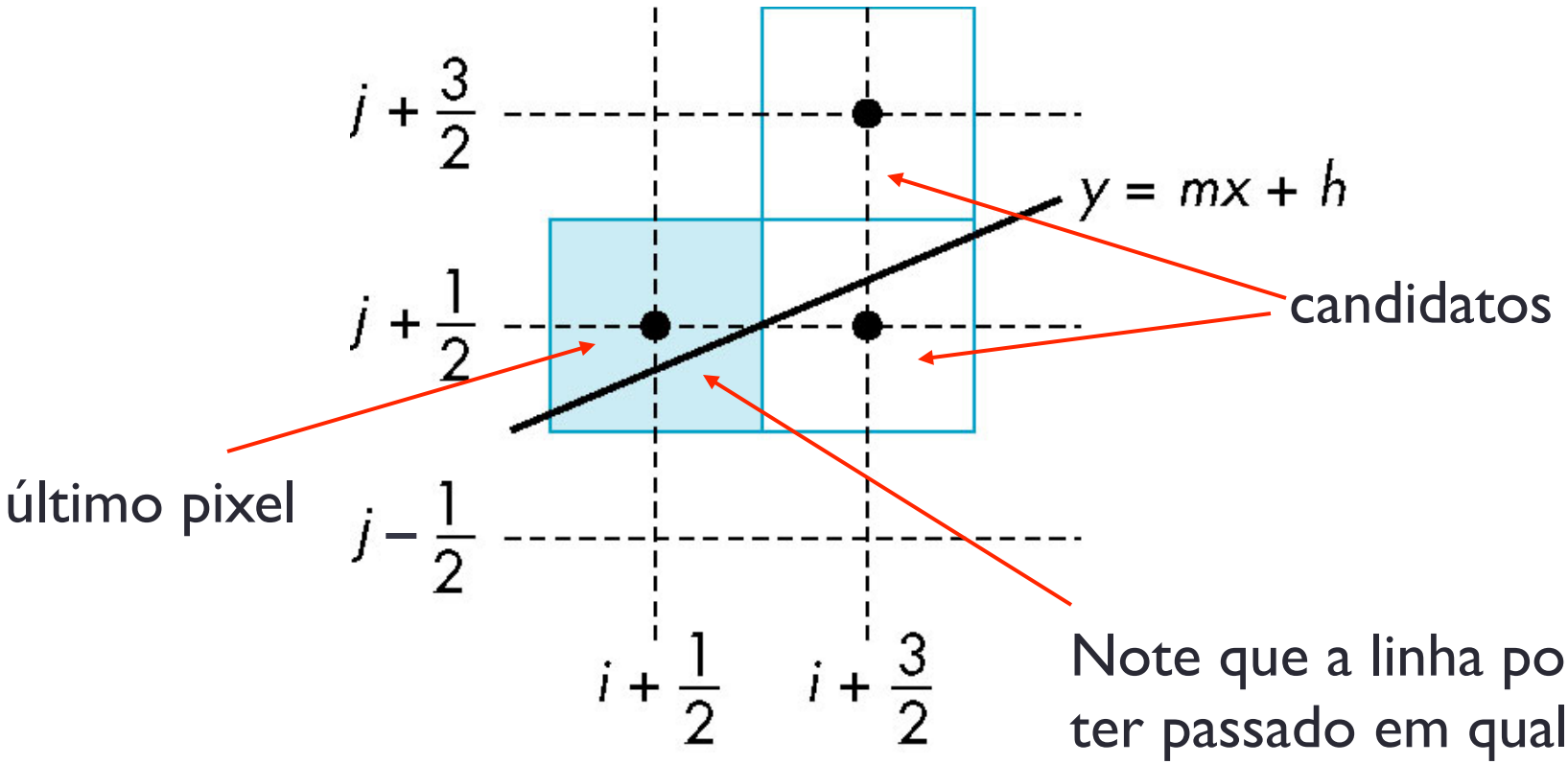
```
for(y=y1; y<=y2, y++) {
    x += 1/m;
  write_pixel(round(x), y, line_color)
}
```

Desenho de retas

- Resumo dos problemas
  - Inclinação das linhas
  - Desempenho
    - Número de operações
    - Operações com números reais x inteiros
    - Multiplicações x adições
- Soluções
  - eliminar ou reduzir operações com números reais
  - aproveitar coerência espacial
    - similaridade de valores referentes a pixels vizinhos

Distância entre reta e ponto

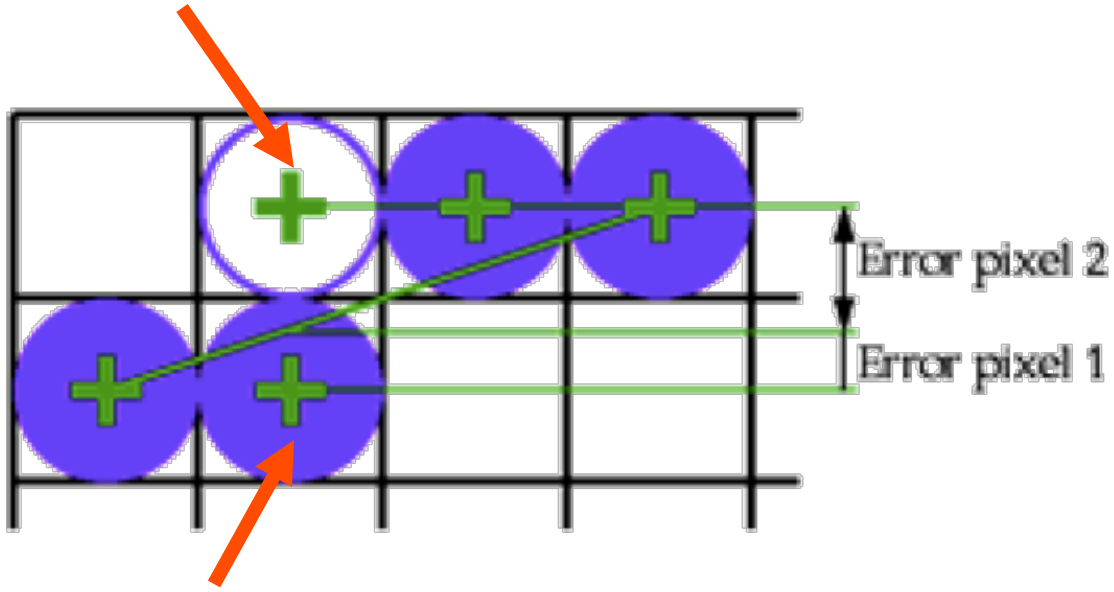
1 >= m > 0



Note que a linha poderia ter passado em qualquer parte deste pixel.

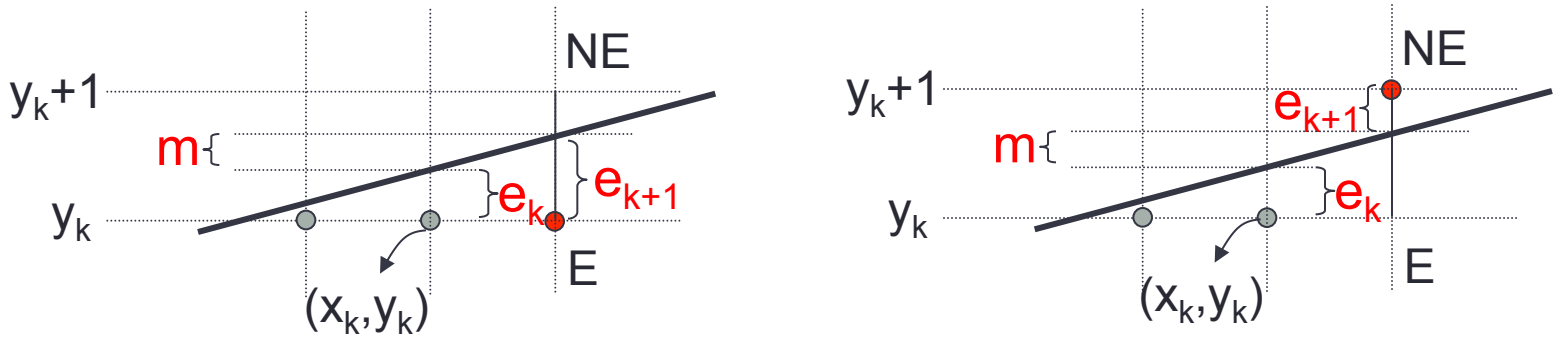
Bresenham

- Que critério usar para escolher entre os dois?
- A distância entre a linha e o centro do pixel
  - Dada pelo erro associado a este pixel



Algoritmo de Bresenham

- Chamemos de e_k, o erro do último pixel desenhado (x_k, y_k).



Algoritmo:

- Estima-se o novo erro (e_{k+1}) caso a escolha seja o pixel E ($y_{k+1} = y_k$)
  - **$e_{k+1} = e_k + m$**

- Se ($e_{k+1} > 0.5$) significa que deveríamos escolher NE ($y_{k+1} = y_k + 1$)
  - Neste caso, o valor correto para e_{k+1} é:
    $e_{k+1} = e_k + m - 1$ (um valor negativo)

- Para facilitar o algoritmo, considere o primeiro pixel com erro = -0.5 e a decisão passa a ser ($e_{k+1} > 0$)

Algoritmo de Bresenham

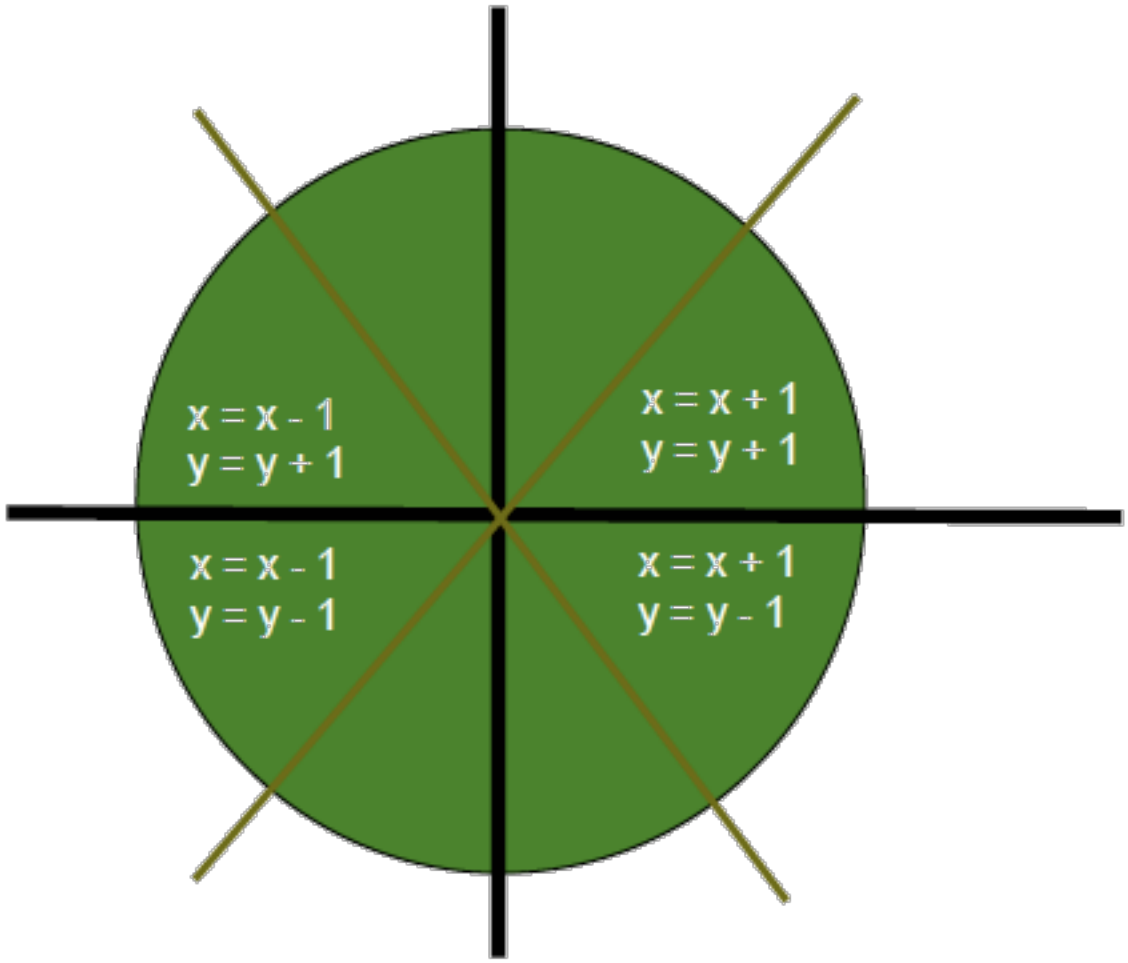
```
void BresLine0(int x1, int y1, int x2, int y2,
               int color)
{
  int   Dx = x2 - x1;
  int   Dy = y2 - y1;
  float m = (float)Dy/Dx;
  float e = -0.5;
  int   x,y;

  write_pixel(x1, y1, color);
  for (x=x1+1; x<=x2; x++)
  {
    e+=m;
    if (e>=0)
    {
      y++;
      e-=1;
    }
    write_pixel(x, y, color);
  }
}
```

Algoritmo de Bresenham

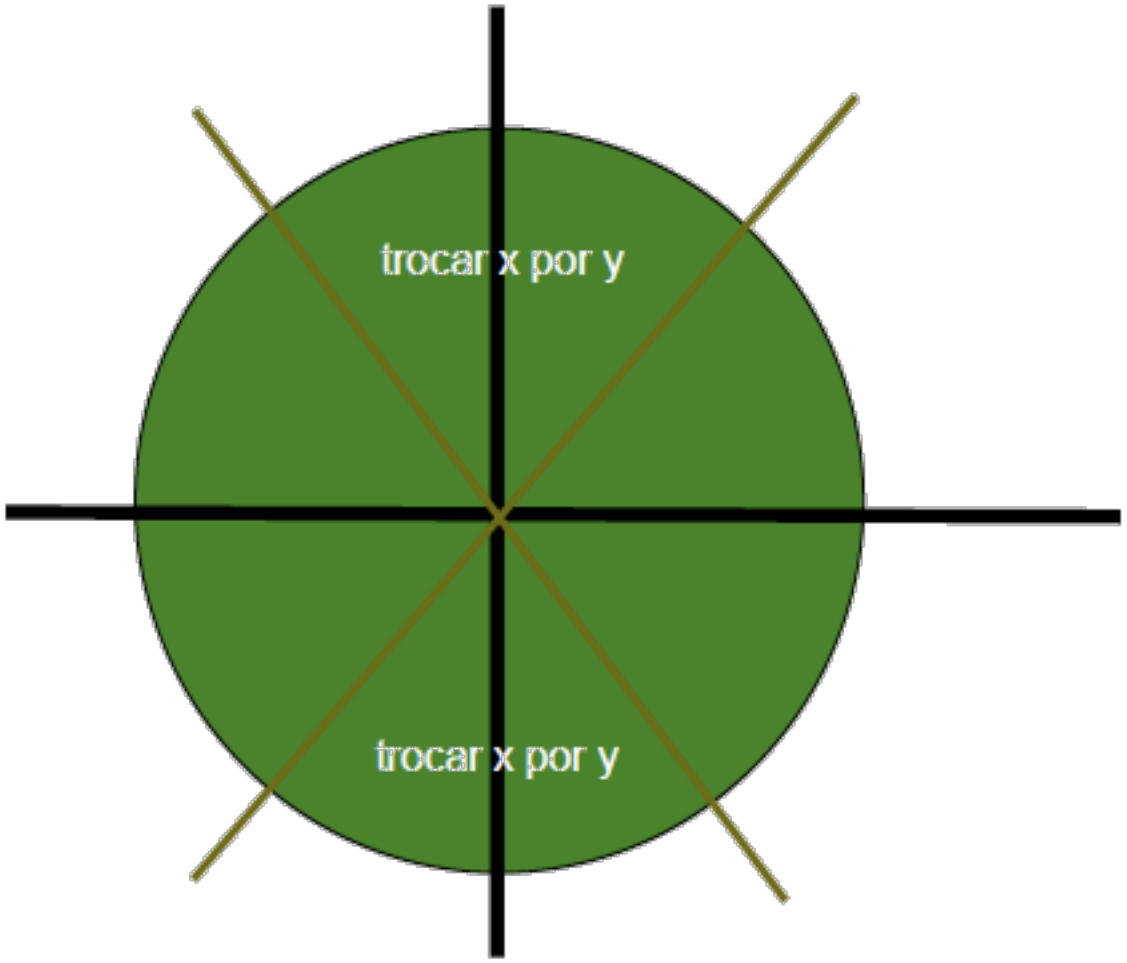
```
void BresLine1(int x1, int y1, int x2, int y2,
               int color)
{
  int DDx, Dx = x2 - x1;
  int DDy, Dy = y2 - y1;
  DDx = Dx << 1;
  DDy = Dy << 1;
  int ei = -Dx;
  int x, y;

  write_pixel(x1, y1, color);
  for (x=x1+1; x<=x2; x++)
  {
    e+=DDy;
    if (e>=0)
    {
      y++;
      e-=DDx;
    }
    write_pixel(x, y, color);
  }
}
```

Para transformar todas as operações em inteiros: multiplicar por (2*Dx)

Bresenham: outros octantes

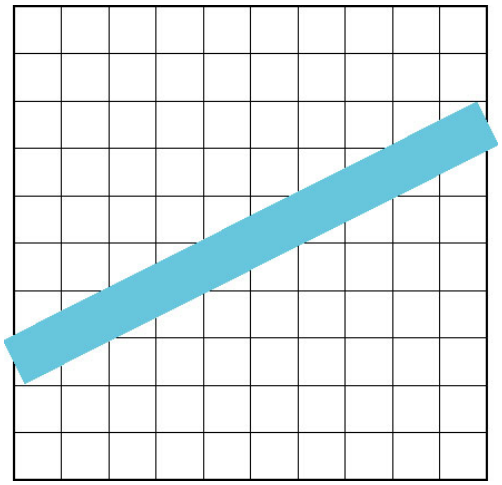


Bresenham: outros octantes



Aliasing

- Uma linha rasterizada de forma ideal deveria ter 1 pixel de largura

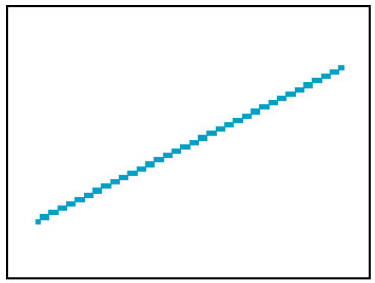


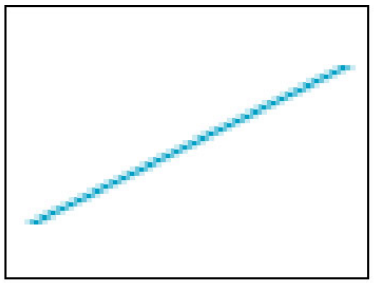
- Escolhendo o melhor y para cada x (ou vice-versa) produz linhas serrilhadas

Antiserrilhamento

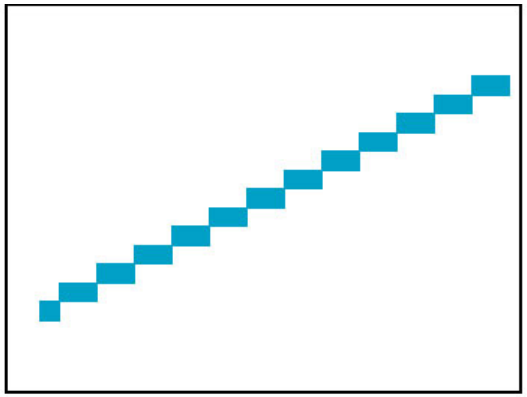
- Colorir múltiplos pixels para cada x dependendo da área de cobertura da reta ideal

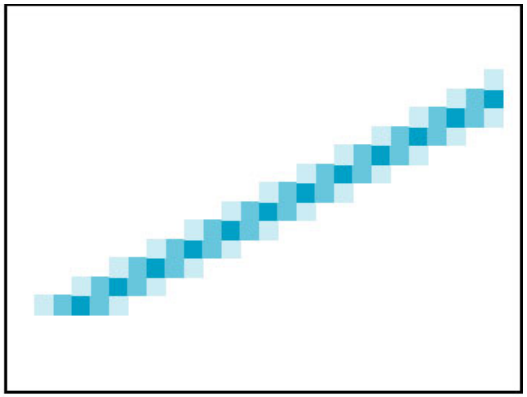
original





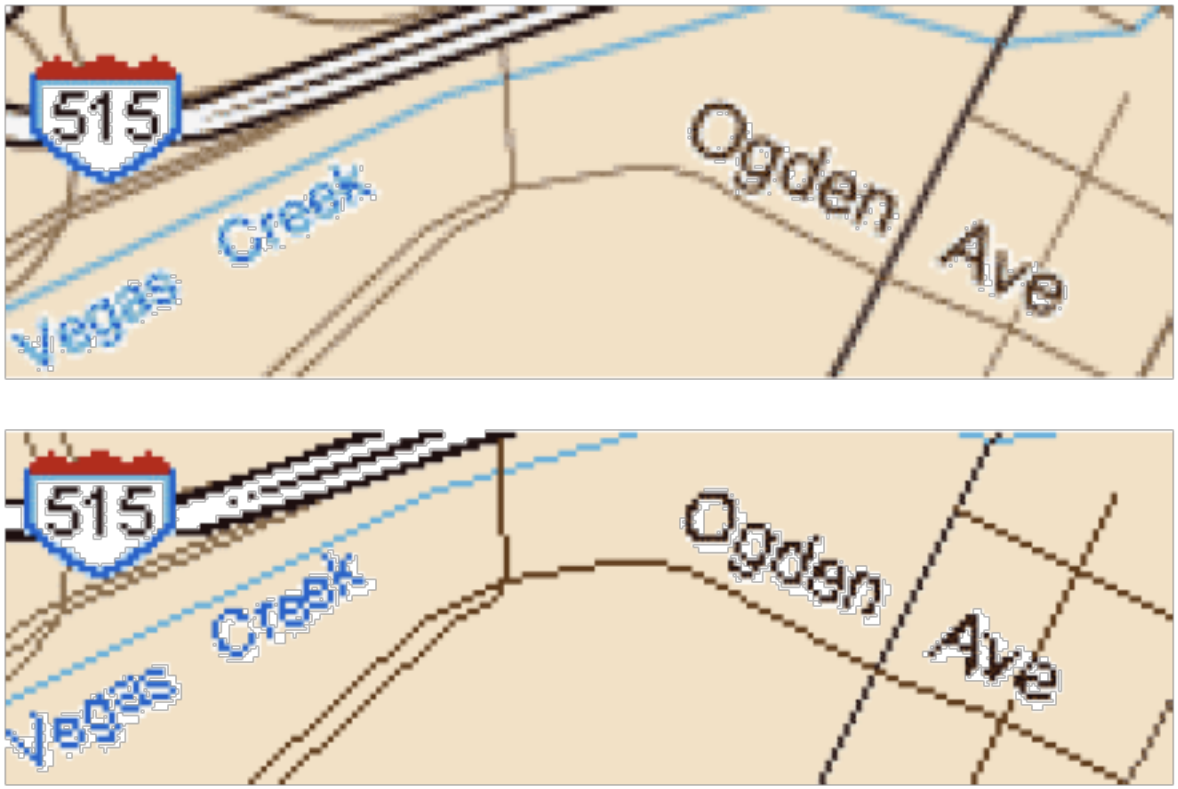
antiserrilhada



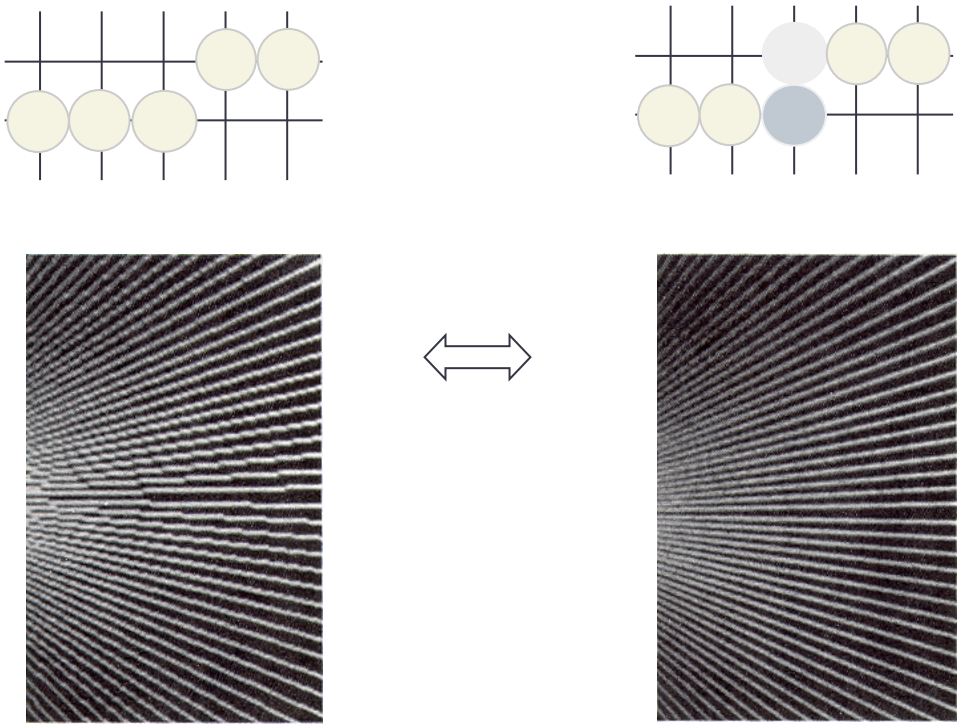


magnificação

Antiserrilhamento



Exemplo



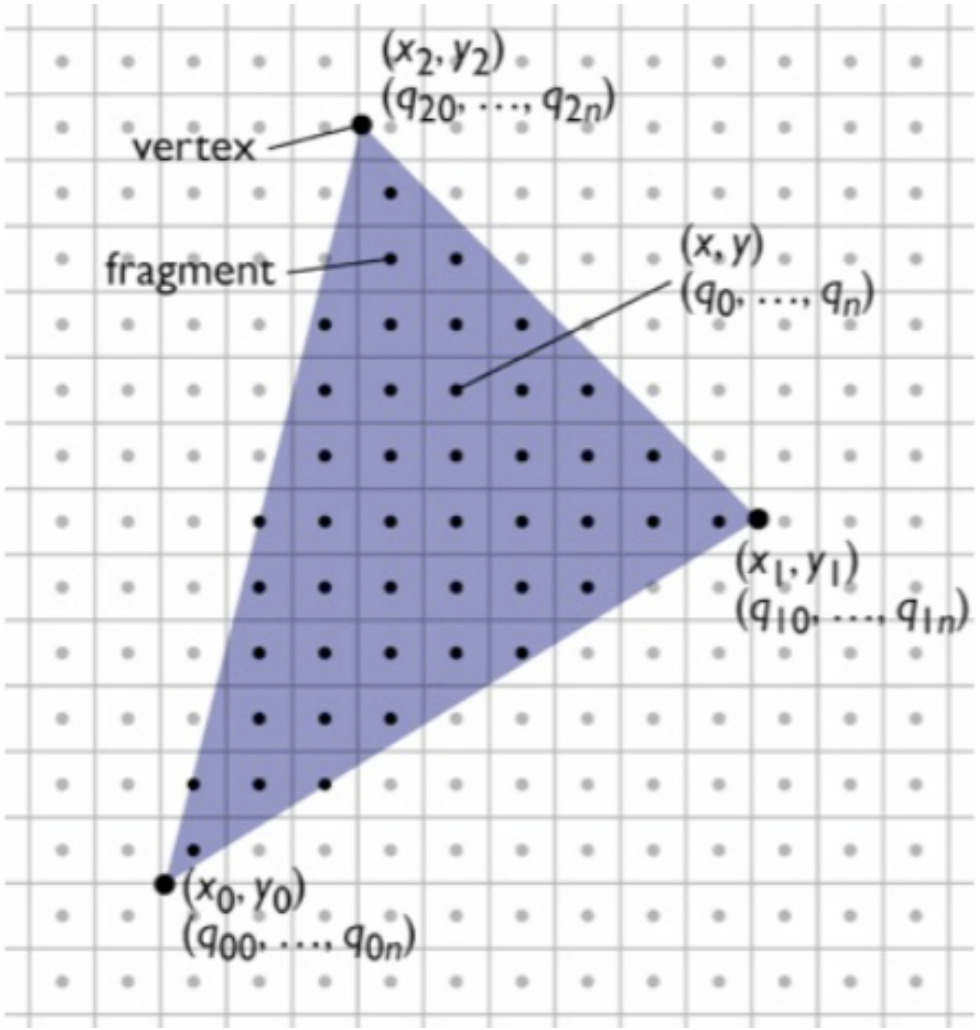
- Observe que quando a cor de fundo não é preto, o anti-aliasing deve fazer uma composição da intensidade com a cor de fundo.
- Anti-aliasing é necessário não só para retas, mas também para polígonos e texturas (o que já é mais complicado)

Rasterização de triângulos

- Input:
  - three 2D points (the triangle's vertices in pixel space)
    - $(x_0, y_0); (x_1, y_1); (x_2, y_2)$
  - parameter values at each vertex
    - $q_{00}, \ldots, q_{0n}; q_{10}, \ldots, q_{1n}; q_{20}, \ldots, q_{2n}$
- Output: a list of fragments, each with
  - the integer pixel coordinates (x, y)
  - interpolated parameter values $q_0, \ldots, q_n$

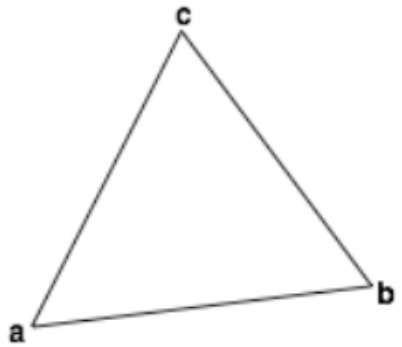
Rasterização de triângulos

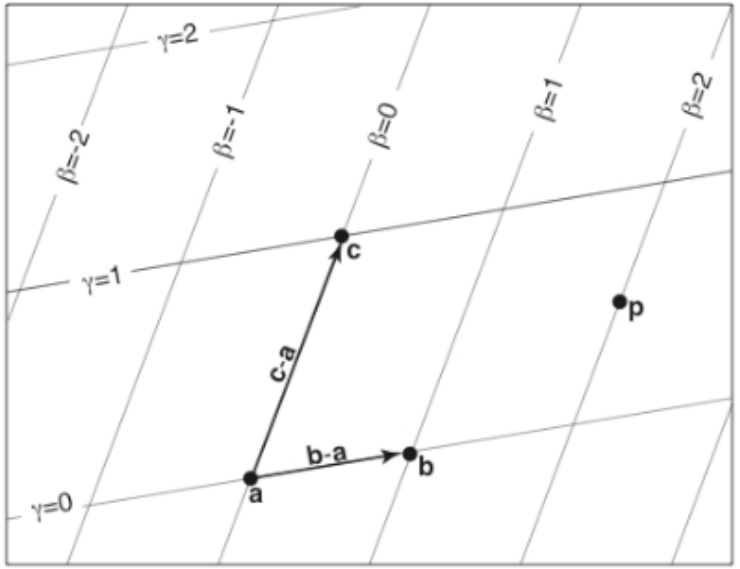
- Summary
  1. evaluation of linear functions on pixel grid
  2. functions defined by parameter values at vertices
  3. using extra parameters to determine fragment set



Rasterização de triângulos

- A triangle is defined by 2D points, **a,b,c**
  - Defines non-orthogonal coordinate system



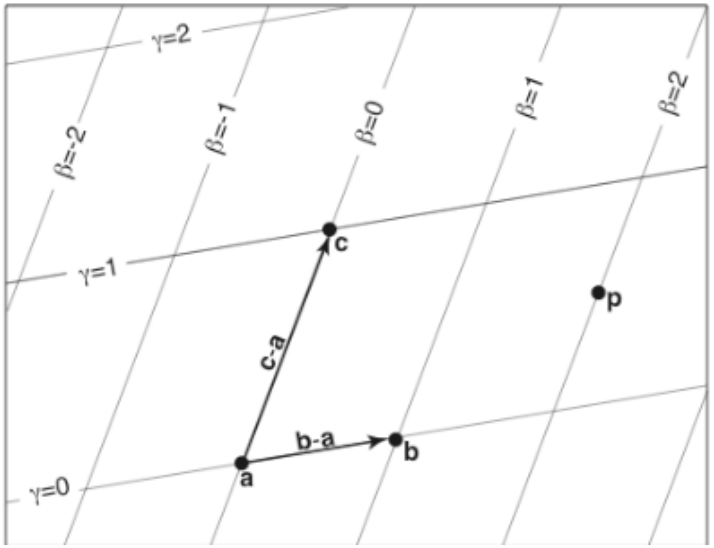


Coordenadas baricêntricas

- Points on triangle satisfy equation

$$\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$\beta, \gamma \in [0,1]$ and $\beta + \gamma <= 1$



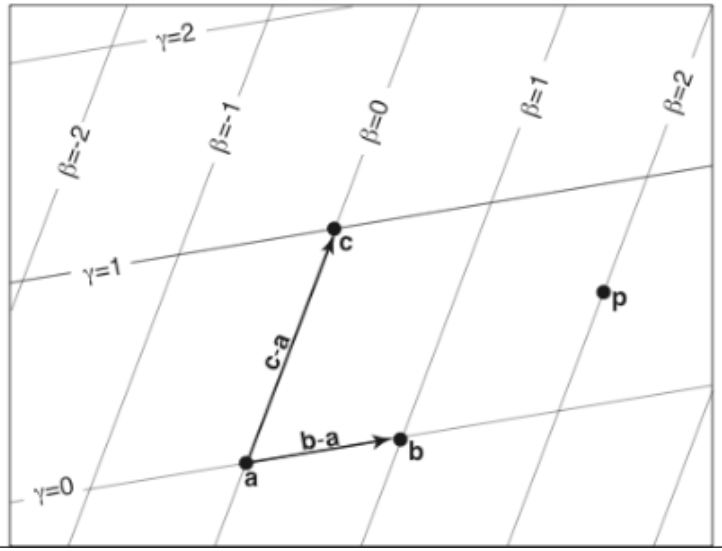
Coordenadas baricêntricas

- Equivalently,

$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\alpha, \beta, \gamma \in [0,1]$$

$$\alpha = 1 - \beta - \gamma$$



Coordenadas baricêntricas

- Given a point P, how to find α, β, γ?
  - Solving linear system

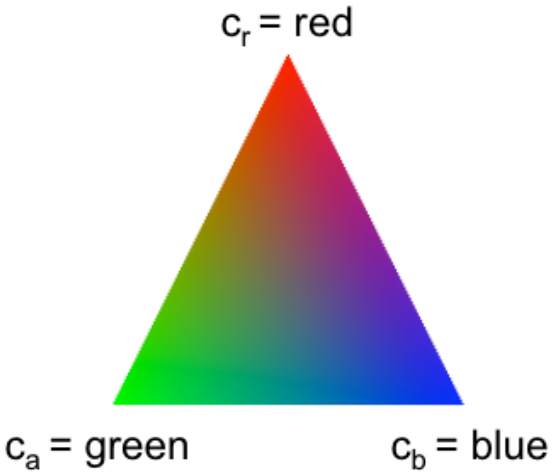
$$\begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x_p - x_a \\ y_p - y_a \end{bmatrix}$$

  - Cramer's Rule

$$\beta = \frac{\begin{vmatrix} x_p - x_a & x_c - x_a \\ y_p - y_a & y_c - y_a \end{vmatrix}}{\begin{vmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{vmatrix}} \qquad \gamma = \frac{\begin{vmatrix} x_b - x_a & x_p - x_a \\ y_b - y_a & y_p - y_a \end{vmatrix}}{\begin{vmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \end{vmatrix}}$$

$$\alpha = 1 - \beta - \gamma$$

Algoritmo de rasterização



$$\mathbf{c} = \alpha\mathbf{c}_a + \beta\mathbf{c}_b + \gamma\mathbf{c}_c$$

For all x do
    For all y do
        Compute (α, β, γ) for (x,y)
            If (all α, β, $\gamma \in [0,1]$)
                $\mathbf{c} = \alpha\mathbf{c}_a + \beta\mathbf{c}_b + \gamma\mathbf{c}_c$
                DrawPixel(x,y,c)

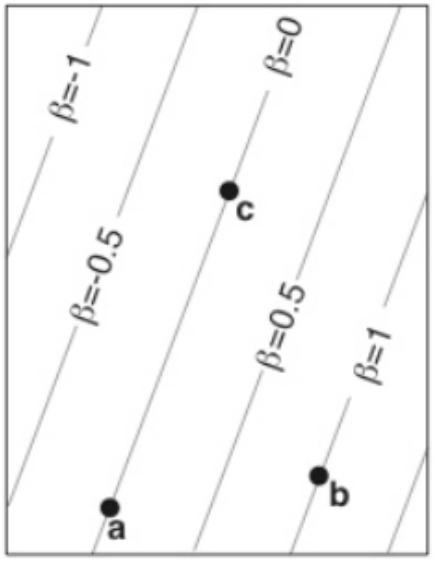
Coordenadas baricêntricas

- Given a point P, how to find α, β, γ?
  - Use the geometric interpretation
    - Let $f_{ac}(x,y)$ be the implicit equation for line ac

$$\beta(x,y) \propto f_{ac}(x,y)$$

$$\beta(x_b,y_b) = 1$$

$$\beta(x,y) = \frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)}$$



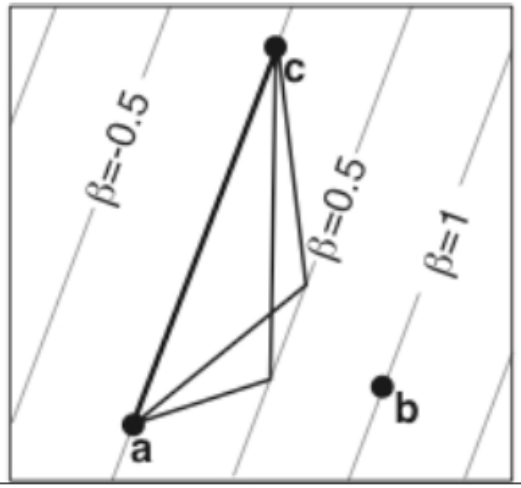
    - Can do similar reasoning for α, γ

Algoritmo de rasterização

For $y = y_{min}$ to y_{max} do
  For $x = x_{min}$ to x_{max} do
    $\beta = f_{ac}(x,y)/f_{ac}(x_b, y_b)$
    $\gamma = f_{ab}(x,y)/f_{ab}(x_c, y_c)$
    $\alpha = 1 - \beta - \gamma$
    If ($\alpha >= 0$ and $\beta >= 0$ and $\gamma >= 0$) then
      $\mathbf{c} = \alpha\mathbf{c}_a + \beta\mathbf{c}_b + \gamma\mathbf{c}_c$
      DrawPixel(x,y,c)

Coordenadas baricêntricas

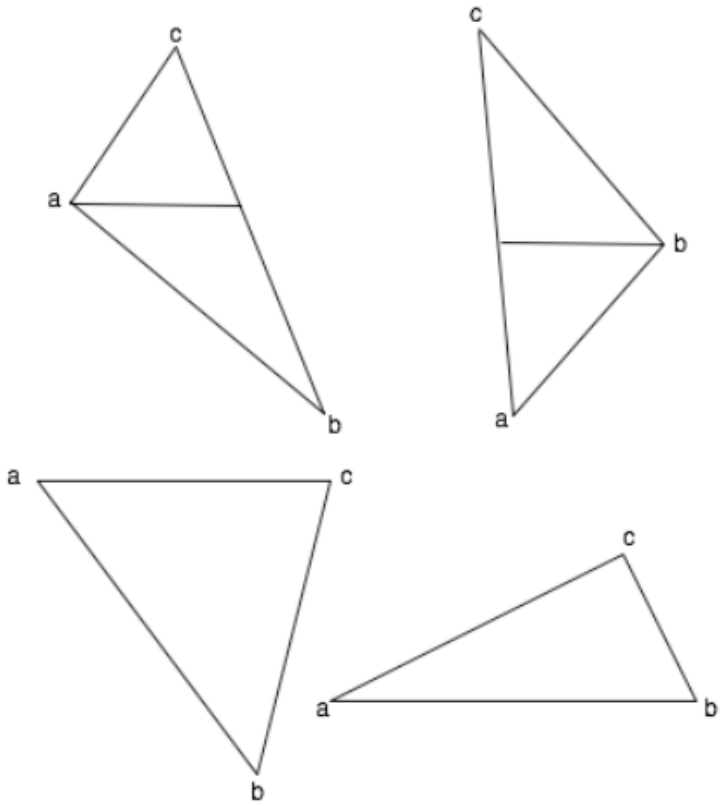
- Geometric Interpretation
  - Ratio of area
    - Eg. β = Area of Δapc / Area of Δabc
    - 1 at corresponding vertex
    - 0 on opposite edge



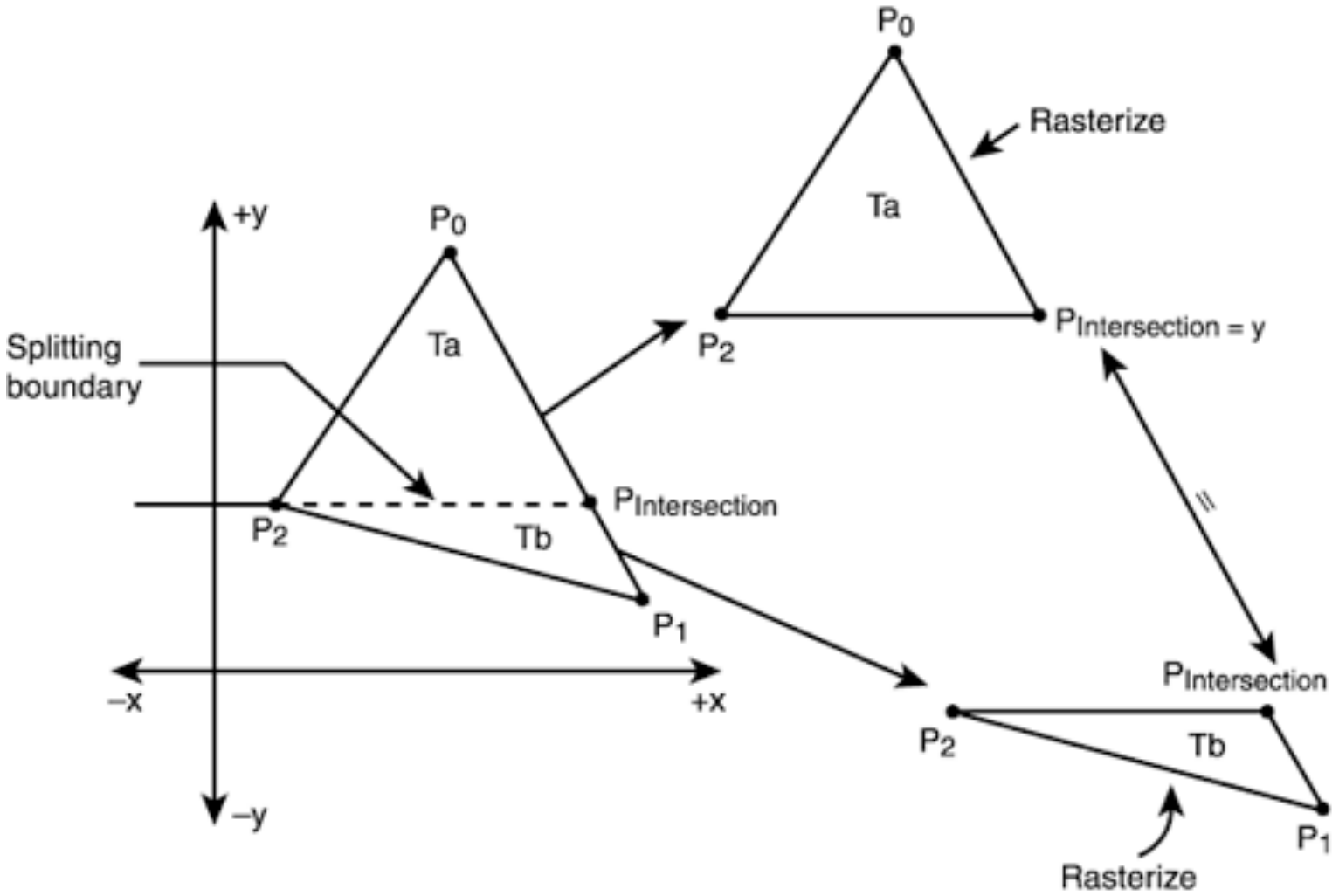
$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

Rasterização de triângulos

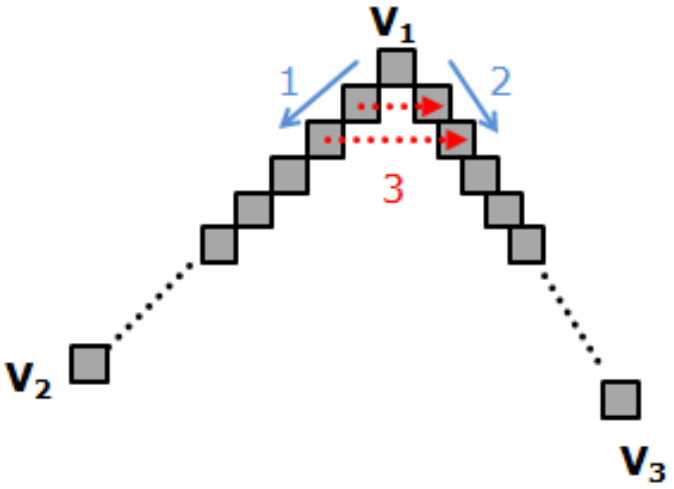
- Fast Algorithm
  - Split triangle into 2 pieces
    - Each piece involves only 2 edges
  - Start with top 2 edges
    - For each y,
      - Compute span
      - ceil for min, floor for max
      - Draw horizontal line
      - Linearly interpolate barycoord
      - Until an edge runs out
    - If not done yet,
      - Continue with another edge



Subdivisão de triângulos



Bresenham



1. Draw line V_1V_2 using the *Bresenham*, but stop if the algorithm moves one pixel in y-direction
2. Draw also line V_1V_3 using the Bresenham, but stop if the algorithm moves one pixel in y-direction
3. At this point we are on the same y-coordinate for line V_1V_2 as well as for line V_1V_3
4. Draw the horizontal lines between both current line points
5. Repeat above steps until you triangle is completely rasterized

Resultado da rasterização

- Interpolate three barycentric coordinates across the plane
  - recall each barycentric coord is 1 at one vert. and 0 at the other two
- Output fragments only when all three are > 0.

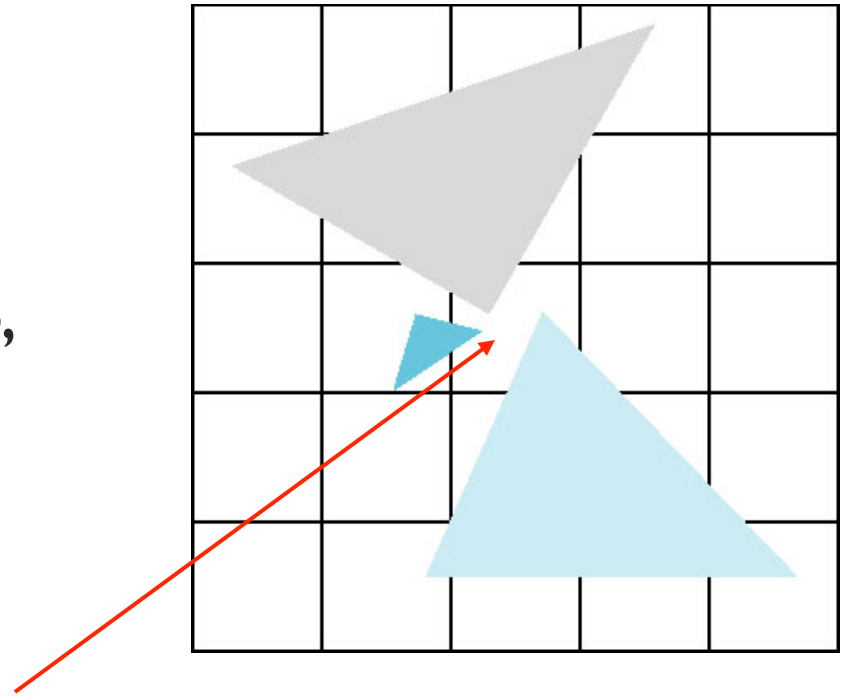
Preenchimento *flood fill*

- Preenchimento pode ser recursivo caso consigamos localizar um ponto no interior do polígono (e.g. WHITE)
- Converte pixels com a color desejada (e.g. BLACK)

```
flood_fill(int x, int y) {
     if(read_pixel(x,y) == WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
}    }
```

Aliasing de polígonos

- Problemas de serrilhamento podem ser sérios:
  - Arestas serrilhadas
  - Desaparecimento de pequenos polígonos
  - Utiliza-se de composição, assim a cor de um único polígono não determina a cor do pixel



Todos os três polígonos devem contribuir para a cor final

Tarefa de casa

- Leitura livro-texto
  - Shirley and Marschner. Fundamentals of Computer Graphics, CRC Press, 3rd Ed. 2010
  - Capítulo 8