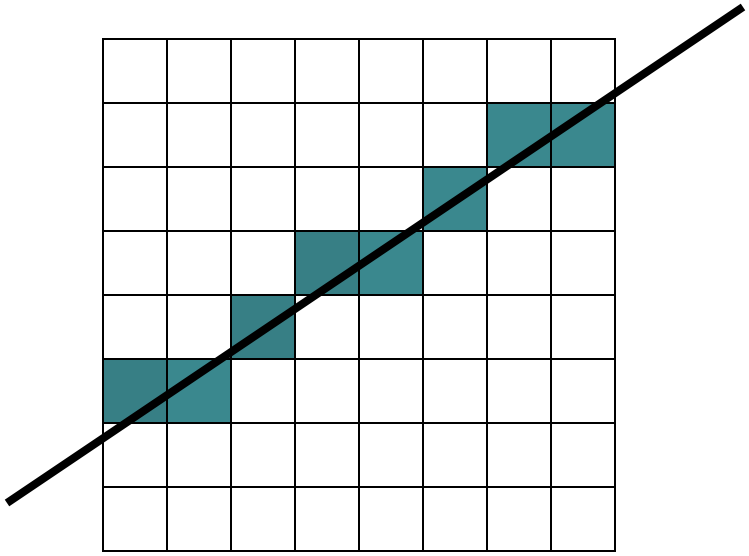
Computação Gráfica

Ana Paula Piovesan Melchiori

Representação Vetorial x Matricial

- Normalmente, gráficos são definidos através de primitivas geométricas como pontos, segmentos de retas, polígonos, etc
  - Representação vetorial
- Dispositivos gráficos podem ser pensados como matrizes de pixels (*rasters*)
  - Representação matricial
- Rasterização é o processo de conversão entre representações vetorial e matricial

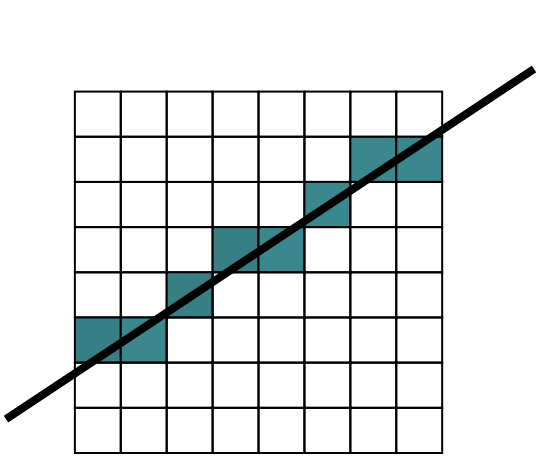


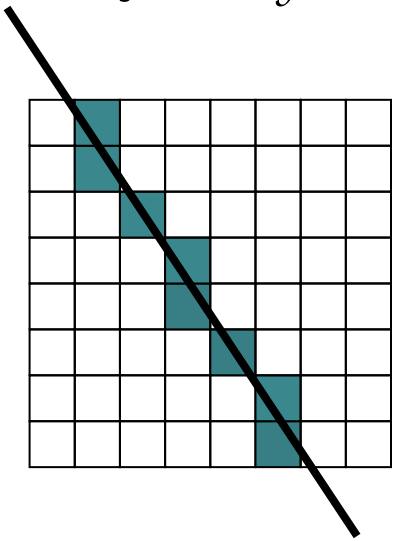
Considerações Gerais

- Rasterização é um processo de amostragem
  - Domínio contínuo → discreto
  - Problemas de *aliasing* são esperados
- Cada primitiva pode gerar um grande número de pixels
  - Rapidez é essencial
- Em geral, rasterização é feita por hardware
- Técnicas de *antialiasing* podem ser empregadas, usualmente extraindo um custo em termos de desempenho

Rasterização de Segmentos de Reta

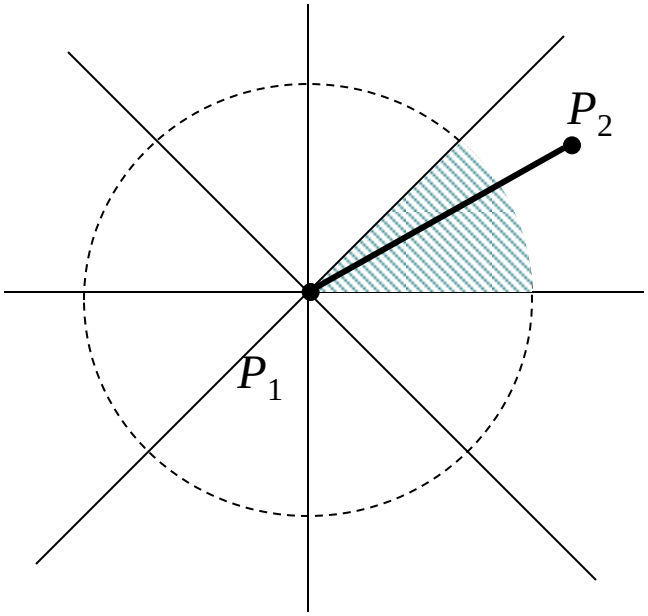
- Segmento de reta entre $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$
  - Já foi recortado com relação ao *viewport*
- Objetivo é pintar os pixels atravessados pelo segmento de reta
  - Na verdade, nem todos, apenas os mais próximos
- Reta de suporte dada por $a\,x + b\,y + c = 0$
- Queremos distinguir os casos
  - Linhas ~ horizontais → computar y como função de x
  - Linhas ~ verticais → computar x como função de y





Algoritmo Simples

- Assumimos segmentos de reta no primeiro octante, com
  - Demais casos resolvidos de forma simétrica
- Inclinação (entre 0 e 1) dada por $m = (y_2 - y_1) \,/\, (x_2 - x_1)$
- Algoritmo:
  - Para x desde x_1 até x_2 fazer:
    - $y \leftarrow y_1 + \lfloor m * (x - x_1) + 0.5 \rfloor$
    - Pintar pixel (x, y)



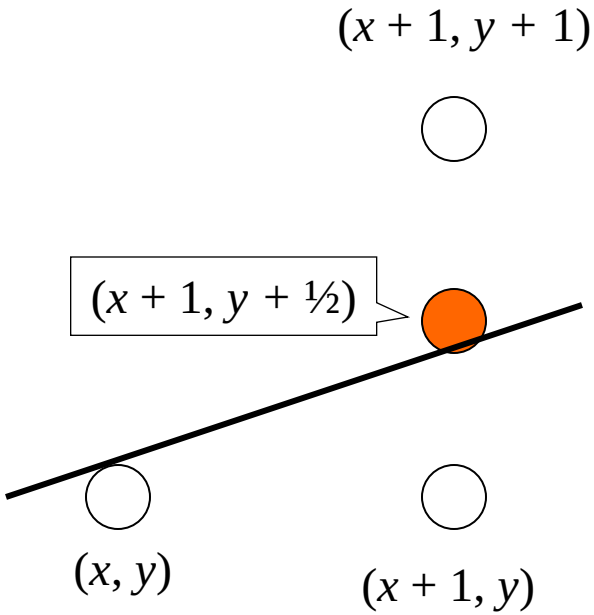
Algoritmo Incremental

- Algoritmo simples tem vários problemas:
  - Utiliza aritmética de ponto-flutuante
  - Sujeito a erros de arredondamento
  - Usa multiplicação
  - *Lento*
- Se observarmos que m é a variação em y para um incremento unitário de x, podemos fazer ligeiramente melhor:

  $x \leftarrow x_1$; $y \leftarrow y_1$
  Enquanto $x \leq x_2$ fazer:
  $x \leftarrow x + 1$
  $y \leftarrow y + m$
  Pintar pixel $(x, \lfloor y + 0.5 \rfloor)$
- Ainda usa ponto-flutuante

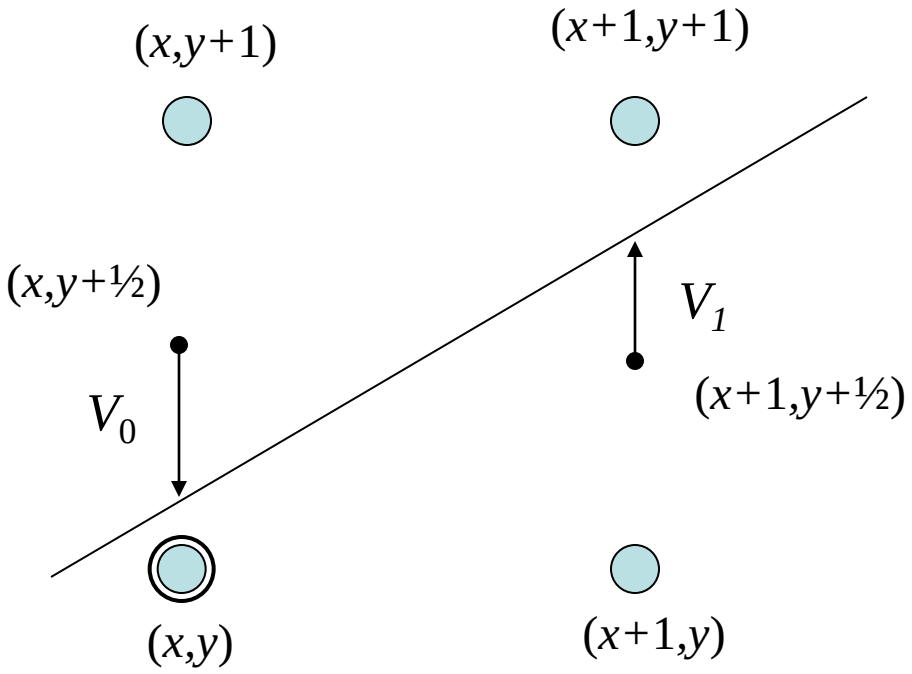
Algoritmo de Bresenham

- Algoritmo clássico da computação gráfica
- Algoritmo incremental que utiliza apenas soma e subtração de inteiros
- Idéia básica:
  - Em vez de computar o valor do próximo y em ponto flutuante, decidir se o próximo pixel vai ter coordenadas $(x + 1, y)$ ou $(x + 1, y + 1)$
  - Decisão requer que se avalie se a linha passa acima ou abaixo do ponto médio $(x + 1, y + ½)$



Algoritmo de Bresenham

- Variável de decisão *V* é dada pela classificação do ponto médio com relação ao semi-espaço definido pela reta
- Caso 1: Linha passou abaixo do ponto médio



$$ax + by + c = V$$

$$\text{onde} \begin{cases} V = 0 \rightarrow (x, y) \text{ sobre a reta} \\ V < 0 \rightarrow (x, y) \text{ abaixo da reta} \\ V > 0 \rightarrow (x, y) \text{ acima da reta} \end{cases}$$

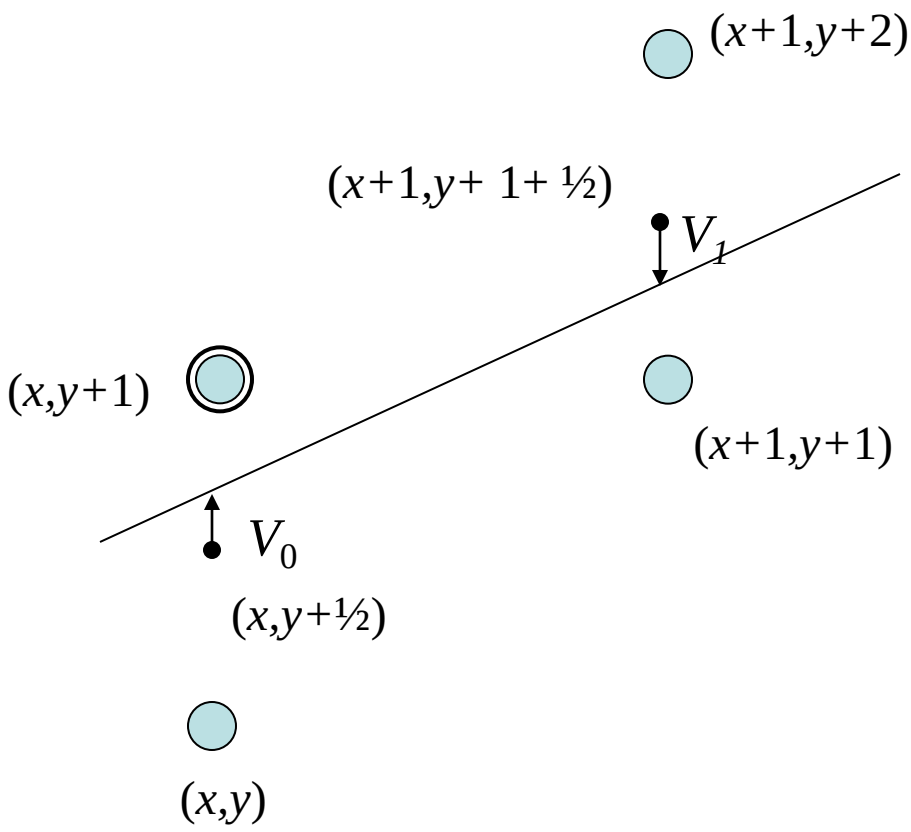
$$V_1 = a(x+1) + b(y + ½) + c$$

$$V_0 = ax + b(y + ½) + c$$

$$\therefore V_1 = V_0 + a$$

Algoritmo de Bresenham

- Caso 2: Linha passou acima do ponto médio



$$V_1 = a(x+1) + b(y+1+½) + c$$

$$V_0 = ax + b(y+½) + c$$

$$\therefore V_1 = V_0 + a + b$$

Algoritmo de Bresenham

- Coeficientes da reta
  - $a = y_2 - y_1$
  - $b = x_1 - x_2$
  - $c = x_2\, y_1 - x_1\, y_2$
- Para iniciar o algoritmo, precisamos saber o valor de V em $(x_1 + 1, y_1 + ½)$
  - $V = a\,(x_1 + 1) + b\,(y_1 + ½) + c$
    $= \underbrace{a\,x_1 + b\,y_1 + c}_{0} + a + b/2 = a + b/2$
- Podemos evitar a divisão por 2 multiplicando a, b e c por 2 (não altera a equação da reta)

Algoritmo de Bresenham - Resumo

$a \leftarrow y_2 - y_1$

$b \leftarrow x_1 - x_2$

$V \leftarrow 2 * a + b$

$x \leftarrow x_1$

$y \leftarrow y_1$

Enquanto $x \leq x_2$ fazer:

    Pintar pixel (x, y)

    $x \leftarrow x + 1$

    Se $V \leq 0$

        Então $V \leftarrow V + 2 * a$

        Senão $V \leftarrow V + 2 * (a + b)$; $y \leftarrow y + 1$

Extensão para demais Octantes

- Se $x_2 < x_1$
  - Trocar P_1 com P_2
- Se $y_2 < y_1$
  - $y_1 \leftarrow - y_1$
  - $y_2 \leftarrow - y_2$
  - Pintar pixel $(x, -y)$
- Se $|y_2 - y_1| > |x_2 - x_1|$
  - Repetir o algoritmo trocando "y" com "x"

Parte Prática

Implementação

Pacote Gráfico 2D

Pacote gráfico

- Um pacote gráfico é um conjunto de rotinas gráficas básicas.
- Estas rotinas permitem o desenvolvimento de programas aplicativos sofisticados sem a necessidade de se preocupar com detalhes particulares dos dispositivos gráficos, como, por exemplo, a forma de se gerar uma reta em um determinado terminal de vídeo.
- A maior dificuldade na construção de pacotes gráficos é dar a importância adequada aos vários procedimentos que afetam a utilidade do pacote.

Característica do Pacote

- **Simplicidade**: características que são muito complexas para o programador de aplicativos entender não serão usadas.
- **Consistência**: um sistema gráfico consistente é aquele que geralmente se comporta de uma maneira previsível. Nomes de funções, seqüências de chamadas, gerenciamento dos erros e sistemas de coordenadas devem seguir um padrão simples e consistente.
- **Completeza**: não deve haver omissões no conjunto de funções do pacote. O pacote deve ser projetado com um pequeno conjunto de funções que possam gerenciar convenientemente um grande número de aplicações.
- **Robustez**: erros triviais de omissão ou repetição podem ser corrigidos sem nenhum "comentário" do sistema. Somente em circunstâncias extremas é que os erros podem causar a finalização da execução.
- **Desempenho**: o desempenho de um pacote gráfico é muito dependente do hardware e do sistema operacional da máquina, porém, quando este é projetado deve-se evitar a utilização de funções gráficas dinâmicas que exijam uma resposta muito rápida do hardware.
- **Economia**: os pacotes gráficos devem ser pequenos e econômicos, de maneira que a inclusão de novas rotinas possa sempre ser considerada.

Padronização e Portabilidade

- A padronização é indiscutivelmente útil e necessária aos pacotes gráficos, pois traz muitos benefícios, destacando-se o grande ganho em termos de portabilidade. Na verdade, na área de Computação Gráfica a padronização está muito ligada à portabilidade.
- A portabilidade pode ser definida como sendo a facilidade de se adaptarem programas já existentes a ambientes diferentes daqueles em que foram desenvolvidos. Um programa portável, ou independente da máquina, pode ser levado de uma instalação para outra com equipamentos diferentes, sem necessitar de muitas alterações ou até mesmo de nenhuma.

Exercício “para aula”

- Simule o processo de rasterização de uma reta
  - Por meio da equação de uma reta qualquer, calcule a matriz dos pontos que representam esta reta.
  - Utilizando qualquer pacote gráfico 2D, plote os pontos na tela.

Exercício “para casa”

- O primeiro exercício da nossa parte prática será desenvolver uma aplicação que simule um Protetor de Tela para o seu micro, onde se utiliza figuras geométrica básicas (linhas, cículos, quadrados....) da biblioteca gráfica que vocês escolherem.

- Exemplos: