Universidade Federal do Vale do São Francisco
Curso de Engenharia de Computação

Computação Gráfica – 04

Prof. Jorge Cavalcanti
jorge.cavalcanti@univasf.edu.br
www.univasf.edu.br/~jorge.cavalcanti
www.twitter.com/jorgecav

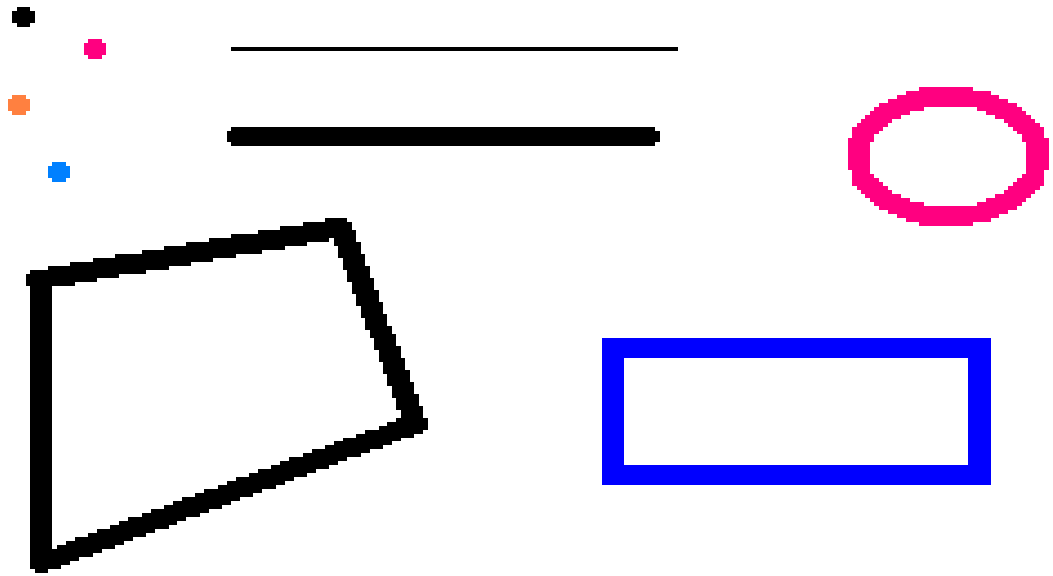
Primitivas Gráficas

Primitivas Gráficas em 2D

- Chamamos de Primitivas Gráficas os elementos básicos que formam um desenho.
  - Exemplos: Ponto, segmento, polilinha, polígono, arco de elipse, etc.
- Primitivas já definidas dão origem a novas primitivas:
  - A polilinha é concatenação de vários segmentos;
  - O polígono é a adequação de uma polilinha;
  - A circunferência é um caso particular de arco de elipse.

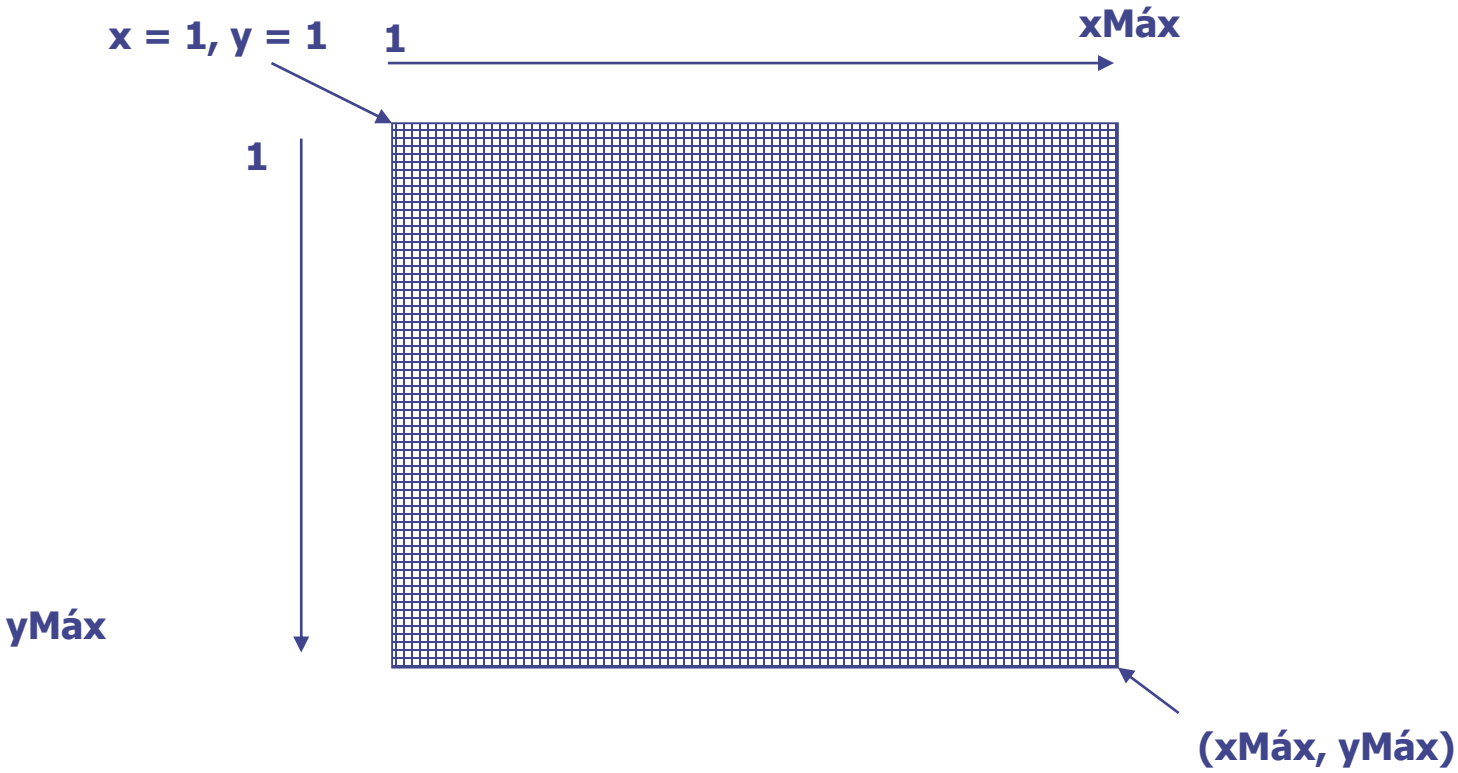
Representação de Imagens

- Atributos podem ser associados às primitivas:
  - O ponto pode ter cor
  - A linha pode ter espessura, cor, traço;



Primitivas Gráficas

A janela gráfica



Primitivas Gráficas

Ponto

- Trataremos o ponto em CG como um pixel, cujas propriedades são:
  - Posição no plano e
  - Cor.
- Como visto, o tamanho do pixel vai depender da resolução gráfica e tamanho físico do dispositivo de exibição da imagem.
- As operações de manipulação de pixels representam uma das essências da CG, pois a partir dessa manipulação, imagens são construídas ou alteradas.
  - Os elementos gráficos mais complexos que o ponto exigem uma sequência de ações para que possam ser construídos.

Primitivas Gráficas

Linhas Retas

- Do ponto de vista matemático, uma reta pode ser descrita como:

$$y = mx + b$$

- O parâmetro **m** é o coeficiente angular, relacionado ao ângulo que a reta faz com o eixo x.
  - Para m <=1, a reta faz um ângulo entre 0º e 45º com o eixo x.
  - Para m >1, a reta faz um ângulo entre 45º e 90º com o eixo x.
- O coeficiente linear **b** dá o valor no eixo y que é cruzado pela reta.
- Dados os pontos no plano P1 e P2, pode-se obter m e b, ou seja, a equação da reta que passa pelos pontos.

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - mx_1$$

Primitivas Gráficas

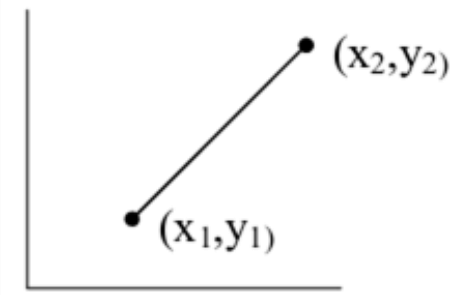
Algoritmos para desenhar retas

- A partir das equações apresentadas para a reta, pode-se definir algoritmos para desenhar um segmento de reta a partir de dois pontos dados.
- **Algoritmo DDA** (*Digital Differential Analyser*) – Baseia-se na aplicação direta das fórmulas que descrevem uma reta no plano.
- Assim, para traçar uma reta que vai do ponto P1 ao P2, podemos pensar no seguinte algoritmo:
  1. Pinta-se o pixel do ponto P1, isto é, o pixel de coordenadas (x_1,y_1), e atribuem-se às variáveis de trabalho (x,y) os valores de (x_1,y_1).
  2. Dá-se o próximo passo: vai ao pixel seguinte (x,y), onde x<- x+1 e y <- y+m e pinta-se esse novo pixel.
  3. Repete o passo 2 até que o ponto P2 seja alcançado.

Primitivas Gráficas

Algoritmo DDA (*Digital Differential Analyser*)

- O DDA é um típico algoritmo incremental. Algoritmos incrementais são aqueles em que uma das variáveis é obtida apenas incrementando o seu valor, por exemplo X = X + 1, e a outra é calculada por alguma regra a partir da primeira.
- Definições para o algoritmo:



$$m = \frac{\Delta y}{\Lambda x} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - m x_1$$

$$\Delta y = m \Lambda x, \quad \Lambda x = \frac{\Lambda y}{m}$$

Primitivas Gráficas

Algoritmo DDA (*Digital Differential Analyser*)

```
void DDA(int X1,Y1,X2,Y2)
{
int step;
float X, Y, Xinc, Yinc;
step = fabs(X2 - X1);
if (fabs(Y2 - Y1)  > step)
     step = fabs(Y2-Y1);
 Xinc = (X2 - X1)/step;
 Yinc = (Y2 - Y1)/step;
 X = X1;
 Y = Y1;
```

```
while(X<X2){
     putpixel(Round(X),Round(Y),9);
      X = X + Xinc;
      Y = Y + Yinc;
  }
}
```

Primitivas Gráficas

Algoritmo DDA (*Digital Differential Analyser*)

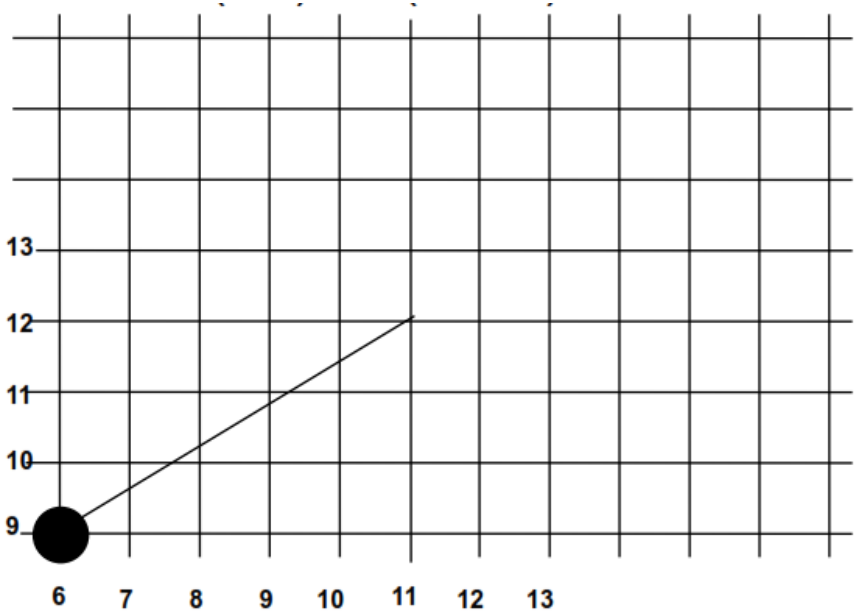
Exemplo: Usando o DDA, compute quais pixels devem ser escolhidos para representar a reta de (6,9) a (11,12)

step = max of fabs(Y2-Y1), fabs(X2-X1) ;

Xinc = (X2 - X1)/step;
Yinc = (Y2 - Y1)/step;

step = Max of (fabs(11-6), fabs(12-9)) = 5

Xinc = (11-6)/5 = 1
Yinc = (12-9)/5 = 0,6



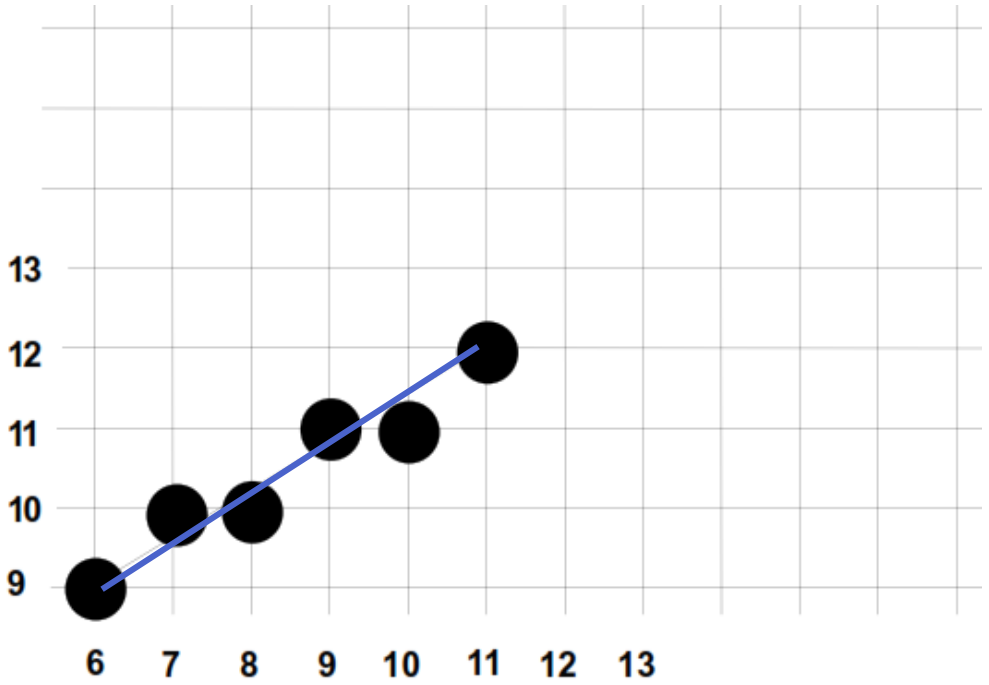
X = X + Xinc
Y = Y + Yinc

Primitivas Gráficas

Algoritmo DDA (*Digital Differential Analyser*)

Exemplo: Usando o DDA, compute quais pixels devem ser escolhidos para representar a reta de (6,9) a (11,12)

Os pontos encontrados são:
(6,9), (7,9.6),
(8,10.2), (9,10.8),
(10,11.4), (11,12)



Primitivas Gráficas

Algoritmo DDA (*Digital Differential Analyser*)

- O DDA cria boas retas, mas consome muito tempo devido as funções de arredondamento.
- Outra desvantagem é que o passo em x é igual a 1 (não existe fração do pixel).
- Há outros comportamentos indesejados para ângulos próximos a 0° e 90°, com a reta quase horizontal ou quase vertical.
- Como o coeficiente **m** não é necessariamente um número inteiro, obriga o hardware a trabalhar com números flutuantes, consumindo tempo de execução.

Primitivas Gráficas

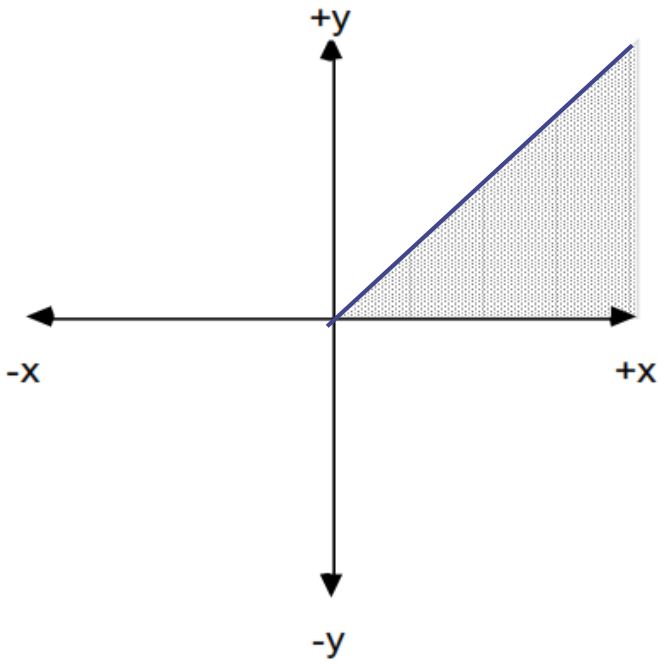
Algoritmo de Bresenham

- O algoritmo de Bresenham (também chamado de **algoritmo do Ponto Médio**) baseia-se no argumento de que um segmento de reta, ao ser plotado, deve ser contínuo, ou melhor, os pixels que compõem um segmento de reta devem ser vizinhos.
  - Será analisado o caso de m <1, pois para m>=1, basta inverter as coordenadas x e y.
- O ponto de partida é a seguinte pergunta: se m<1 e dado um ponto na reta, o próximo ponto é (x+1,y) ou (x+1, y+1)?
  - O algoritmo de Bresenham responde a essa questão calculando um valor (p) para cada pixel, e passando para o pixel seguinte, até alcançar o último pixel do segmento da reta.

Primitivas Gráficas

Algoritmo de Bresenham

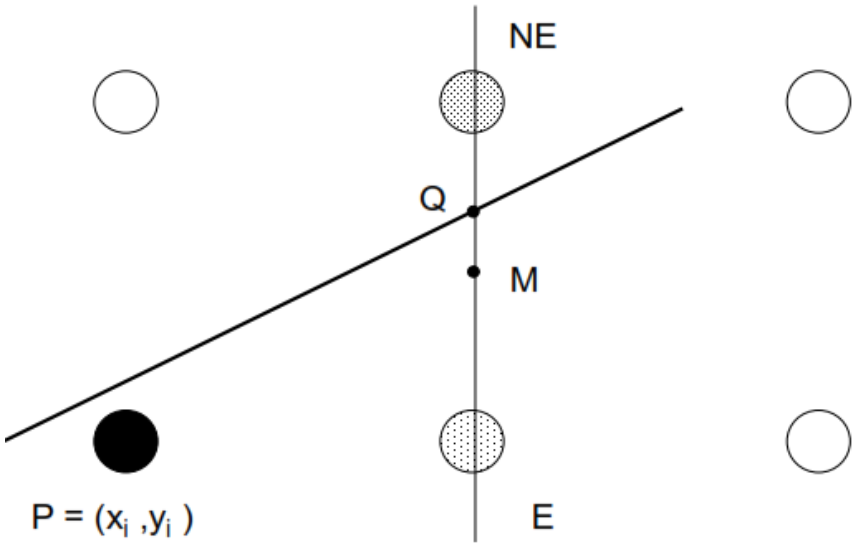
- Ex.: Queremos desenhar uma reta entre os pontos (x1, y1) e (x2, y2), com declive **m** entre 0 e 1 (i.e. reta dentro do 1º octante).



Primitivas Gráficas

Algoritmo de Bresenham

- Para as retas do 1º octante, dado um pixel sobre a reta, o próximo pixel é para direita (E) ou para Direita acima (NE).
- Tendo o pixel (x_i, y_i) o próximo pixel é NE em (x_i+1, y_i+1) ou E em (x_i+1, y_i)?



Primitivas Gráficas

Algoritmo de Bresenham

- Vamos determinar um método para calcular de que lado da reta está do ponto M. Para isso, considere sua função implícita:

$$F(x,y) = ax + by + c = 0.$$

- Se **dy = y2 - y1, e dx = x2 - x1**;
- A equação da reta em termos de sua inclinação pode ser escrita como:

$$y = \frac{dy}{dx}x + B$$

- Na forma implícita: $F(x,y) = dy.x - dx.y + dx.B=0$. Onde **a = dy, b = - dx e c = dx.B**. Assim, verifica-se que:

F(x,y) = 0, ponto sobre a linha
F(x,y) > 0, para pontos abaixo da linha
F(x,y) < 0, para pontos acima da linha

Primitivas Gráficas

Algoritmo de Bresenham

- Para simplificar a definição do próximo ponto em função da posição relativa da reta em relação ao ponto médio, usa-se uma variável de decisão **p**.
  - **p=2dy – dx**
- Se p < 0, o próximo ponto será (x+1, y) e recalcula **p= p+2dy**
- Se p >= 0, o próximo ponto será (x+1, y+1) e recalcula **p= p+2dy – 2dx.**
- O algoritmo é mostrado a seguir. Sua principal vantagem é usar somente variáveis inteiras, dispensando operações de ponto flutuante.
- O cálculo de (x_i+1, y_i+1) é feito incrementalmente usando os cálculos feitos para (x_i, y_i).

Primitivas Gráficas

Algoritmo de Bresenham

- Parâmetros de entrada: (x1,y1) e (x2,y2).

1. Calcula-se **$dx=x_2-x_1$** e **$dy=y_2-y_1$.**
2. Calculam-se as variáveis auxiliares: **2dy** e **2dy-2dx**.
3. Coloca-se nas variáveis de trabalho o ponto inicial: **x<- x1** e **y<- y1**.
4. Plota-se o ponto (x,y).
5. Calcula-se o parâmetro de decisão: **p=2dy – dx**.
6. Se p for negativo, então **x <-x+1**, **p <- p+2dy** e vai para o passo 8.
7. Se p for positivo ou zero, então: **x <- x+1**, **y <- y+1** e **p <- p+2dy – 2dx**.
8. Repetem-se os passos 6 a 7 até que o ponto (x_2,y_2) seja alcançado.

Primitivas Gráficas

Algoritmo de Bresenham

```
void bresenham(int x1, int y1, int x2, int y2)
{
int dx,dy,p,p2,xy2,x,y,xf;

dx = x2-x1;
dy = y2-y1;
p = 2 * dy - dx;
p2 = 2 * dy;
xy2 = 2 * (dy-dx);

if (x1>x2)
   {x = x2; y = y2; xf = x1; }
else
   {x = x1; y = y1; xf = x2; }
putpixel(x,y,9);
```

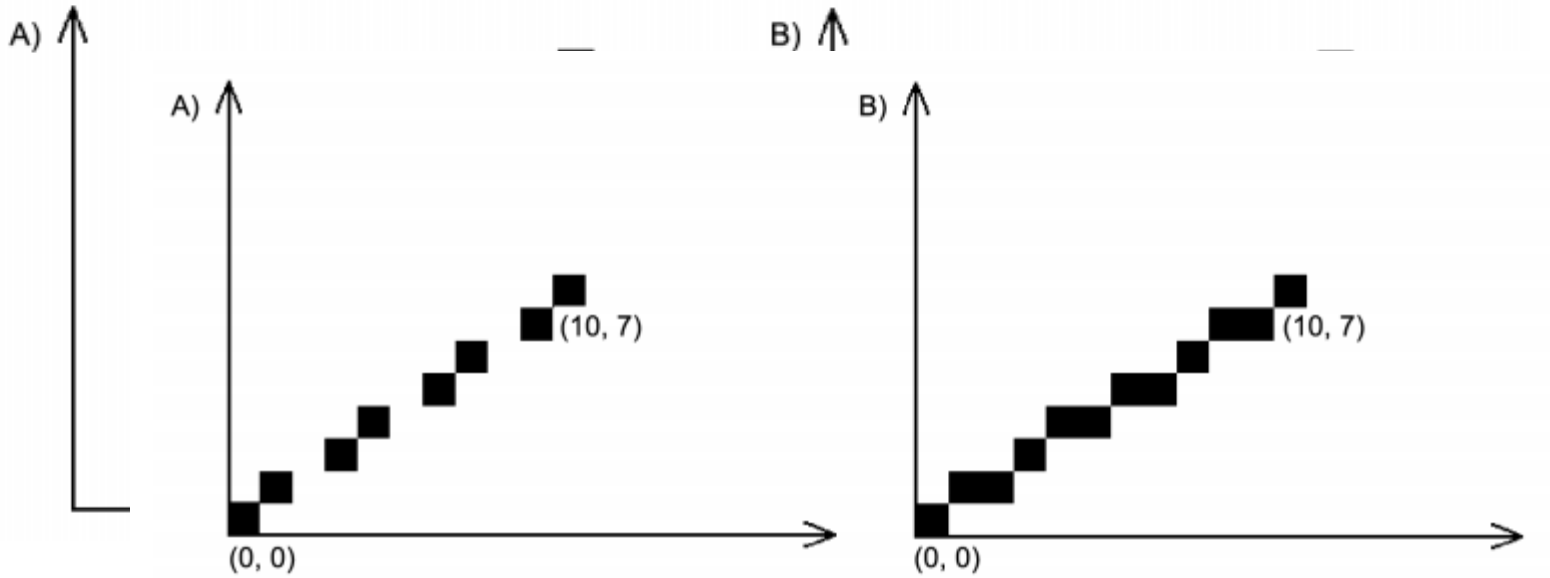
```
while (x<xf) {
        x++;
            if (p<0)
                p += p2;
          else {
                 y++;
                p += xy2;
                }
        putpixel(x, y, 9);
      }
}
```

Primitivas Gráficas

Exercício

Simule a aplicação do algoritmo DDA e do algoritmo de Bresenham para obter a tabela com os pontos de plotagem dos seguintes segmentos de retas.

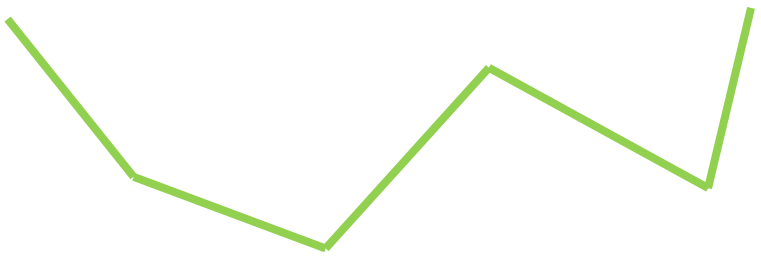
a)(20,10)-(30,18) b)(0,0)-(11,7)



Primitivas Gráficas

Polilinhas

- Linhas retas são a base para uma grande variedade de figuras que são compostas por segmentos de retas.
- A maioria desses objetos pode ser desenhada por algoritmos simples que por sua vez chama o algoritmo de traçar retas.
- Essa sequência de retas são chamada de polilinhas.



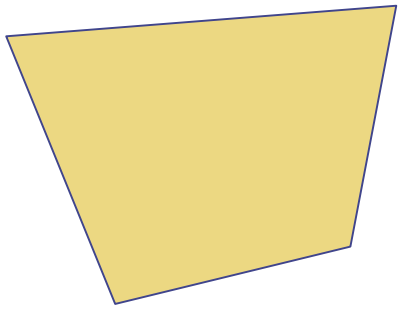
Propriedades

1. Composta por n segmentos de reta, sendo n>=1.
2. É definida por n+1 pontos.

Primitivas Gráficas

Polilinhas

- O algoritmo que desenha uma polilinha deve desenhar um segmento de reta ligando o primeiro ao segundo ponto, o segundo ao terceiro e assim por diante, até o penúltimo e o último ponto.
- Pode-se ainda usar esse mesmo raciocínio para desenhar um **polígono**, que pode ser descrito como uma polilinha fechada.



Propriedades

1. Composto por n segmentos de reta, sendo n>=2.
2. É definida por n pontos.

Primitivas Gráficas

Polilinhas

- O algoritmo que desenha um polígono deve desenhar um segmento de reta ligando o primeiro ao segundo ponto, o segundo ao terceiro e assim por diante, até o que se desenhe uma reta ligando o último ao primeiro ponto, fechando o polígono.
- Caso seja definido apenas dois pontos, haverá dois segmentos de reta, um indo do primeiro ponto ao segundo e o outro voltando. Não será desenhado o polígono, mas n=2 satisfaz o algoritmo.

Primitivas Gráficas

Círculos e Elipses.

- Os círculos e os arcos são elementos fundamentais para a computação gráfica, para geração de curvas e superfícies.
  - Além da aplicação gráfica, os círculos são essenciais para o aprendizado dos detalhes mais complexos do traçado de curvas através dos computadores.

Traçado de círculos

- O círculo é definido como um conjunto de pontos que estão a uma mesma distâncias de um ponto.
  - A distância é o raio e o ponto equidistante é o centro do círculo.

Primitivas Gráficas

Traçado de círculos

- Matematicamente:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- Essa definição não tem como ser usada em CG, pois deseja-se uma definição do tipo y=f(x) ou x=f(y).
  - Essas definições podem ser obtidas isolando-se as variáveis:

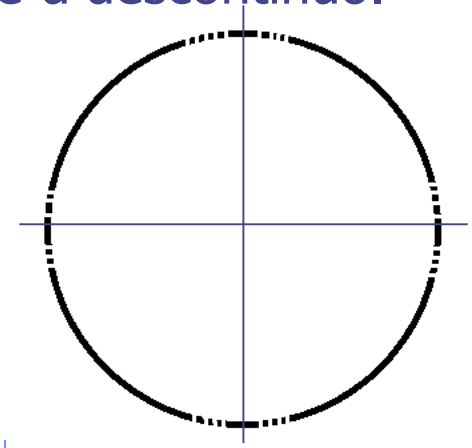
$$x = x_c \pm \sqrt{r^2 - (y - y_c)^2} \qquad y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

  - A operação ± é necessária pois a raiz pode ser positiva ou negativa.
  - Para cada valor de x (1ª expr), são obtidos dois valores de y: um para metade superior do círculo, outro para a inferior.

Primitivas Gráficas

Traçado de círculos

- Essas expressões, apesar de corretas, apresentam dois problemas evidentes para serem usadas em computação gráfica:
  - Exigem muitos cálculos (quadrados e raiz quadrada).
  - Geram imprecisão no traçado, pois quando o segmento do círculo fica quase horizontal ou vertical, um pequeno incremento em *x <- x+1* (ou *y <- y+1*) leva a um salto e o arco apresentar-se-á descontínuo.



Primitivas Gráficas

Traçado de círculos

- Pode-se chegar a uma outra definição de círculos usando o sistema de coordenadas polares.
  - As novas expressões apresentam pontos de um círculo como função do raio e de um ângulo.

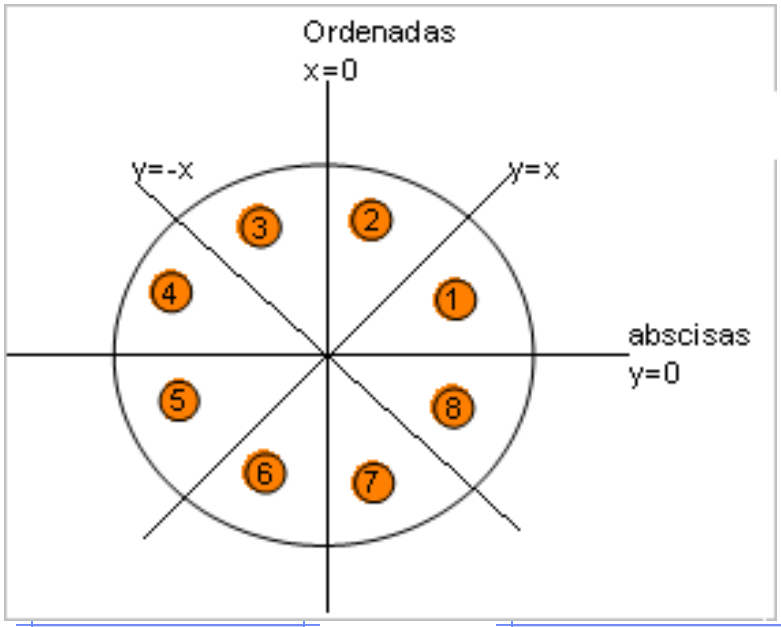
$$x = x_c + r\cos\theta \qquad y = y_c + r\,\mathrm{sen}\,\theta$$

  - Onde θ é um ângulo que varia entre 0 e 2π (em radianos).
- Um algoritmo trivial de traçado de círculos pode ser idealizado a partir da escolha de um passo para θ (que chamaremos de $\Delta\theta$ e um loop que calcula os valores de x e y para cada valor de θ.
  - Cada ponto calculado é inserido numa tabela que mais tarde é fornecida como parâmetro para traçado de uma polilinha fechada.

Primitivas Gráficas

Traçado de círculos

- Esse algoritmo pode ser otimizado, se levarmos em conta as simetrias num círculo.
- O cálculo de um octante (*1/8* do círculo, com θ indo de 0 a π/4 apenas) pode ser copiado para os outros octantes, conforme a figura e tabela abaixo:



Oct	Xn	Yn
1	x	y
2	y	x
3	y	-x
4	-x	y
5	-x	-y
6	-y	-x
7	-y	x
8	x	-y

Primitivas Gráficas

Traçado de círculos

- Os octantes podem se calculados e desenhados independentemente (8 polilinhas) ou calculados todos em conjunto (1 polilinha).
- Apesar de simples e eficaz, esse algoritmo apresenta dois defeitos:
  - A precisão é dependente do raio do círculo e o cálculo de funções trigonométricas, mesmo para poucos pontos, implica em perda de tempo e agilidade.

Primitivas Gráficas

Algoritmo do ponto médio para círculos

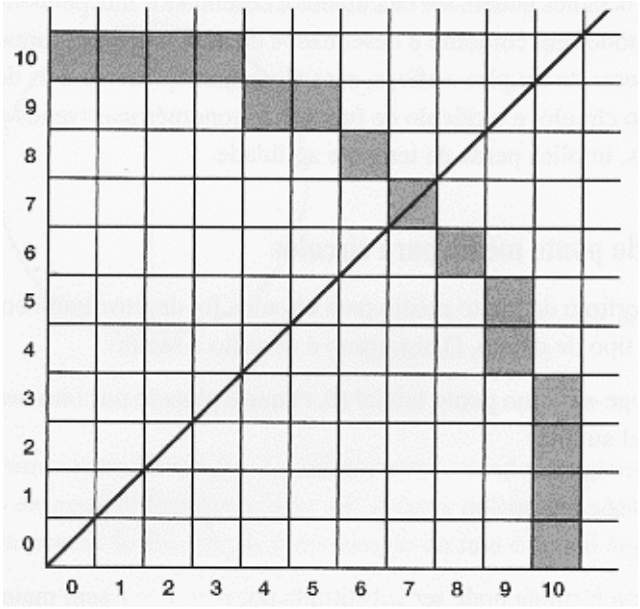
- O algoritmo do ponto médio para círculos foi desenvolvido com o objetivo de otimizar o traçado desse tipo de objeto.
  1. Elege-se como ponto inicial (0,r) que é plotado nos oito octantes, e ainda calcula-se uma variável auxiliar p = 5/4 – r (ou simplesmente *p=1-r*).
  2. Incrementa-se x.
  3. Se p for negativo, então calcula-se um novo *p=p+2x+1*; caso contrário, decrementa-se y e calcula-se *p=p+2x+1–2y*.
  4. Plota-se o novo ponto (x,y) nos oito octantes.
  5. Os passos 2 e 4 são repetidos enquanto x<y.
- A seguir é apresentada uma simulação da execução desse algoritmo para um círculo de raio 10 pixels, com centro em (0,0).

Primitivas Gráficas

Algoritmo do ponto médio para círculos

- A variável auxiliar p valerá inicialmente 1 - r = 1-10 = -9. Os valores de (x,y) estão inicialmente com (0,10). Os valores obtidos no loop estão na tabela abaixo e o resultado da aplicação do algoritmo na figura.

Loop	p	(x, y)	$2x$	$2y$
0	−9	(0, 10)	0	20
1	−9	(1, 10)	2	20
2	−6	(2, 10)	4	20
3	−1	(3, 10)	6	20
4	6	(4, 9)	8	18
5	−3	(5, 9)	10	18
6	8	(6, 8)	12	16
7	5	(7, 7)	14	14



Primitivas Gráficas

Exercícios

1. Escreva um programa em C que receba como parâmetros o centro, o raio e uma cor e desenhe o círculo usando o algoritmo simples de simetria de octantes.

```
void CircleSimple(int xc, int yc, int r,int c)
void PlotPoint(int xc, int yc, int x, int y,int c)
```

2. Altere a função **`CircleSimple`** para a função **`CircleMidPoint`** implementando o algoritmo do ponto médio para círculos.

```
CircleMidPoint(int xc, int yc, int r, int c)
```

Enviar por e-mail até o início da próxima aula.

Primitivas Gráficas

Preenchimento de áreas

- O resultado de qualquer atividade gráfica tem que levar em conta as cores dos objetos que são visualizados.
- A propriedade mais simples que se pode para definir um objeto 2D é uma cor sólida, única.
  - Isso é feito de forma trivial: todos os pixels que se encontram dentro do contorno do objetos são carregados com o mesmo valor (mesma cor).
- O desenho de um objeto 2D sólido passa a ter duas etapas:
  1. O traçado do contorno, que mantém as propriedades geométricas do objeto.
  2. A fase do preenchimento do seu interior, sua área.

Primitivas Gráficas

Preenchimento de áreas

- É interessante manter as duas atividades separadas, pois cada objeto tem suas propriedades (algoritmos) e a cor pode ser um genérico.
  - A seguir são mostrados 2 algoritmos de preenchimento, partindo do princípio que o contorno do objeto já está desenhado.

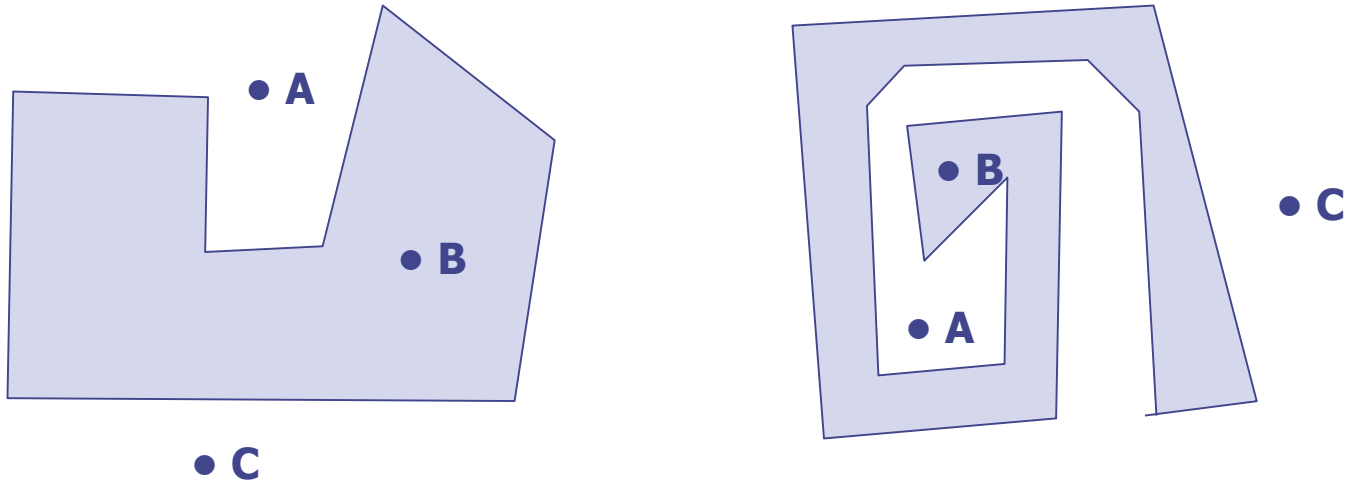
Algoritmo ponto dentro-ponto fora

- Baseia-se na seguinte proposição: alguns pontos da tela (ou frame-buffer) estão dentro do contorno e outros não. Basta saber como identificá-los.
- Apesar de parecer trivial, a identificação não é tão simples como parece ser.

Primitivas Gráficas

Algoritmo ponto dentro-ponto fora

- Considere as figuras abaixo:

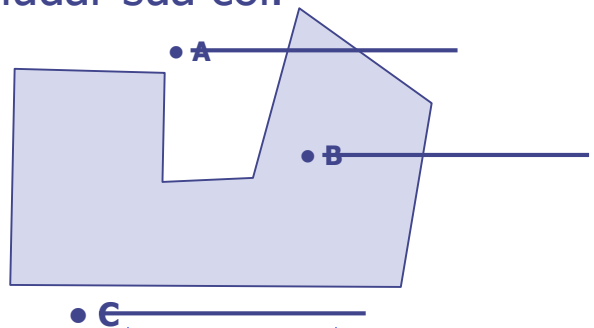


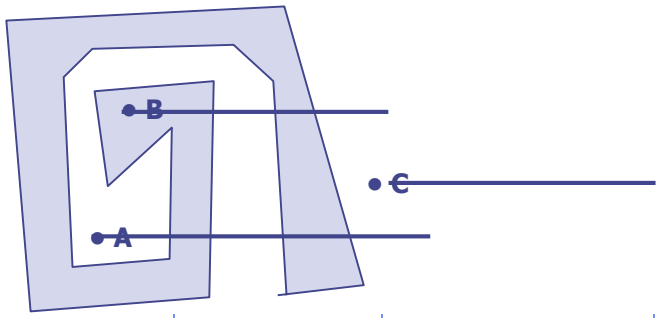
- É claro que os pontos A e C estão fora do polígono e que o ponto B está dentro. Nossa capacidade de interpretação de imagens permite essa conclusão óbvia.
  - O problema é definir um algoritmo que chegue ao mesmo resultado.

Primitivas Gráficas

Algoritmo ponto dentro-ponto fora

- Uma solução muito utilizada, baseada nas teorias da topologia prevê o seguinte:
  1. Traça-se um segmento de reta do ponto até o limite da tela.
  2. Conta-se quantas vezes essa reta intercepta os lados do polígono.
  3. Se o número de lados interceptados for ímpar, o ponto está dentro do polígono, caso contrário, o ponto está fora.
  4. Uma simplificação nos testes pode ser obtida se forem feitos apenas retas horizontais, num único sentido, por exemplo, a borda direita.
  5. Uma vez estabelecido que o pixel está dento do polígono, basta mudar sua cor.



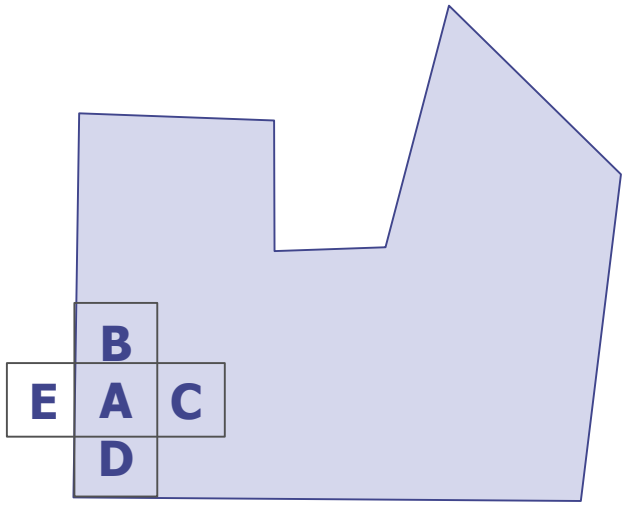


Primitivas Gráficas

Algoritmo de preenchimento recursivo

- Mais adequado para linguagens de alto nível (que admita recursão), sem necessidade de economia de memória.
  - A ideia básica é que um pixel vizinho de outro pixel que está dentro do polígono, também está dentro.

1. Conforme a figura abaixo, partindo de um ponto A, que se sabe estar dentro do interior do polígono, testam-se os pontos B, C, D e E.
2. Se um deles também estiver dentro do polígono, repete-se o teste para este ponto.
3. Um risco para o algoritmo é que se o ponto B tem como vizinho o ponto A e, se fosse aplicado o mesmo procedimento no ponto A, teríamos um *loop* no programa e o polígono não seria pintado em tempo finito.



Primitivas Gráficas

Algoritmo de preenchimento recursivo

- Uma maneira de evitar o problema é marcando os pontos que já foram visitados, para que o procedimento não seja aplicado mais de uma vez para o mesmo ponto.
  - Fazer isso é simples, pois uma vez que se deseja é pintar o interior do polígono, basta pintar (ligar) o pixel cada vez que este for considerado como vizinho.
  - Somente os pixels não pintados serão considerados como vizinhos válidos, para os quais pode ser aplicado o procedimento.