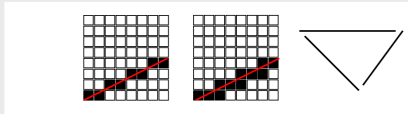
Desenho de Linhas

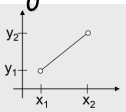
Desenho de Segmentos de Reta

Requisitos

- O algoritmo tem de obter coordenadas inteiras, porque só pode endereçar coordenadas (x,y) inteiras no *raster display.*
- Os algoritmos têm de ser eficientes: execução ao nível do *pixel* é chamada centenas ou milhares de vezes.
- Os algoritmos devem criar linhas com aspeto visual satisfatório:
  - Devem parecer "retas"
  - Terminar com precisão
  - Apresentar brilho constante



Algoritmo



Equação da reta: $y = m.x + b$

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

Podemos observar que:
Se m<1 então x avança sempre uma unidade; y mantém valor anterior ou é incrementado.
Se m>1 então y avança sempre uma unidade; x mantém valor anterior ou é incrementado.

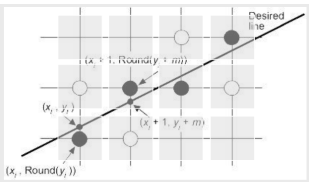
A equação pode ser simplificada para:
$y_{i+1} = m.x_{i+1} + b = m(x_i + \Delta x) + b = y_i + m.\Delta x$ Fazendo $\Delta x = 1$, $\mathbf{y_{i+1} = y_i + m}$

Algoritmo Básico para desenhar o segmento de reta (m<1)
1. Incrementar *x* de 1 em cada passo, partindo do ponto mais à esquerda.
2. $y_{i+1} = y_i + m$
3. O ponto a desenhar será: $(x_{i+1}$, round$(y_{i+1}))$ { O pixel mais próximo da reta real, i.e. cuja distância é a menor.

Digital Differential Analyser (DDA)

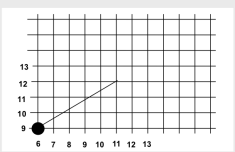
```
void DDA(int X1, int Y1, int X2, int Y2)
{   //considerando -1 <=m <=1  e  X1<X2
    int  x;
    float dy, dx, y, m;

    dy = Y2 - Y1;
    dx = X2 - X1;
    m = dy/dx;
    y = Y1;
    for (x=X1; x<=X2; x++) {
       WritePixel(x, (int)(y + 0.5));
       y += m;
    }
}
```



Problemas do algoritmo:
1. Operações em vírgula flutuante -> menor eficiência do que com inteiros
2. O valor de ***y*** evolui pelo incremento sucessivo de ***m*** (valor real); variáveis reais têm precisão limitada -> soma acumulada de um valor inexato pode originar um desvio do valor real pretendido *round*(y_i).

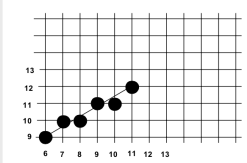
Exercício:
Quais os pontos que serão desenhados? Segmento de reta entre (6,9) e (11,12)



m = ?

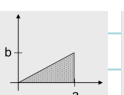
Os pontos calculados são:

```
( 6,  9.0),
( 7,  9.6),
( 8, 10.2),
( 9, 10.8),
(10, 11.4),
(11, 12.0)
```



Algoritmo Midpoint

- Supor que se pretende desenhar um segmento de reta entre os pontos (0,0) e (a,b) tal que:
  - 0 <= m <= 1
- A equação da reta fica:
  $y = m.x$ sendo $m = b/a$
- Se a reta passa na origem:
  $y = (b/a).x + 0 \rightarrow f(x,y) = bx - ay = 0$
  é também uma equação da reta.



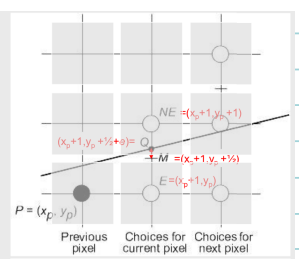
Para retas no primeiro octante, o ponto seguinte a ***P*** será ***E*** ou ***NE***.

Escolher o ponto mais próximo da reta real:

$f(x,y) = bx - ay = 0$

Estratégia do algoritmo MidPoint:
1. Verificar de que lado fica *M*
2. Se ***M*** acima da reta → escolhe ***E***
3. Se ***M*** abaixo da reta → escolhe ***NE***

O erro será sempre inferior a ½.

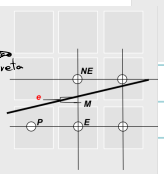


e=Q-M

Algoritmo Midpoint

O ponto médio entre ***E*** e ***NE*** é $(x_p + 1, y_p + ½)$.
Façamos **e** a distância entre o ponto médio e o ponto onde a reta intersecta entre ***E*** e ***NE***. e=M−I
Se **e** for positivo -> escolhe-se ***NE*** → Acima do M;
Se **e** for negativo -> escolhe-se ***E*** → Abaixo do M;

I → Interseção NE,E com a reta



Conclui-se que, para escolher o ponto correcto. apenas é necessário saber o sinal de **e**.

$f(x_p + 1, y_p + ½ + e) = 0 \Leftrightarrow$ (ponto pertence à reta)
$b(x_p + 1) - a(y_p + ½ + e) = 0 \Leftrightarrow$
$b(x_p + 1) - a(y_p + ½) - a.e = 0 \Leftrightarrow$
$f(x_p + 1, y_p + ½) - a.e = 0 \Leftrightarrow$
$f(x_p + 1, y_p + ½) = a.e$

Designemos uma **variável de decisão** d_p como:
$d_p = f(x_p + 1, y_p + ½)$

Sendo a > 0

sign(e) =
sign(a.e) =
sign($f(x_p + 1, y_p + ½)$) =
sign(d_p)

→ apenas é necessário calcular o sinal de d_p para escolher o próximo ponto.

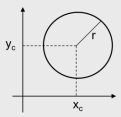
Otimizações nos slides 11 – 14
– reduzir complexidade de operações.

Desenho de Circunferências

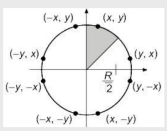
Algoritmo Midpoint para desenho de circunferências

Algumas propriedades das circunferências:

1. Calcular a circunferência pela sua equação $(x-x_c)^2+(y-y_c)^2 = r^2$ não é eficiente.



2. A simetria da circunferência pode ser explorada:
Obtendo ***(x,y)*** obtém-se também:
(-x,y) (-x,-y) (x, -y)
(y,x) (-y,x) (-y,-x) (y, -x)
→ Calcula-se apenas o segundo octante desde x=0 até x=y=R/sqrt(2)

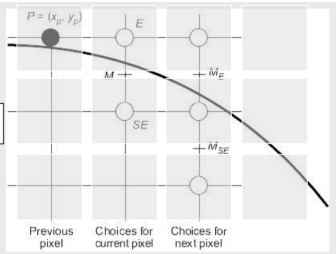


3. Se centro em (0,0) → $f(x,y)=x^2+y^2-r^2$
f(x,y) < 0 então (x,y) está **dentro** da circunferência
= 0 então (x,y) está **sobre** a circunferência
> 0 então (x,y) está **fora** da circunferência

Da mesma forma que foi feito para a reta define-se a variável de decisão ***d***:

$d_p = f(x_p + 1, y_p - ½) =$

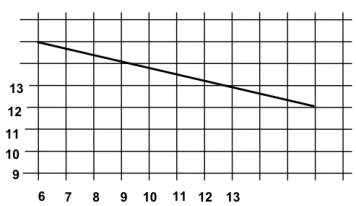
$(x_p+1)^2+(y_p-½)^2 - r^2$ → Subtração



otimização da variável de decisão nos slides 18 - 20.

Exercício

1. Generalize o algoritmo para funcionar com qualquer declive positivo 0 <= ***m*** < *infin.*
2. Generalize o algoritmo para funcionar com qualquer octante
3. Implemente o código correspondente e teste...



4. Utilize o algoritmo de Midpoint para obter a tabela de pontos e o valor de d_i em cada etapa para o caso da figura.