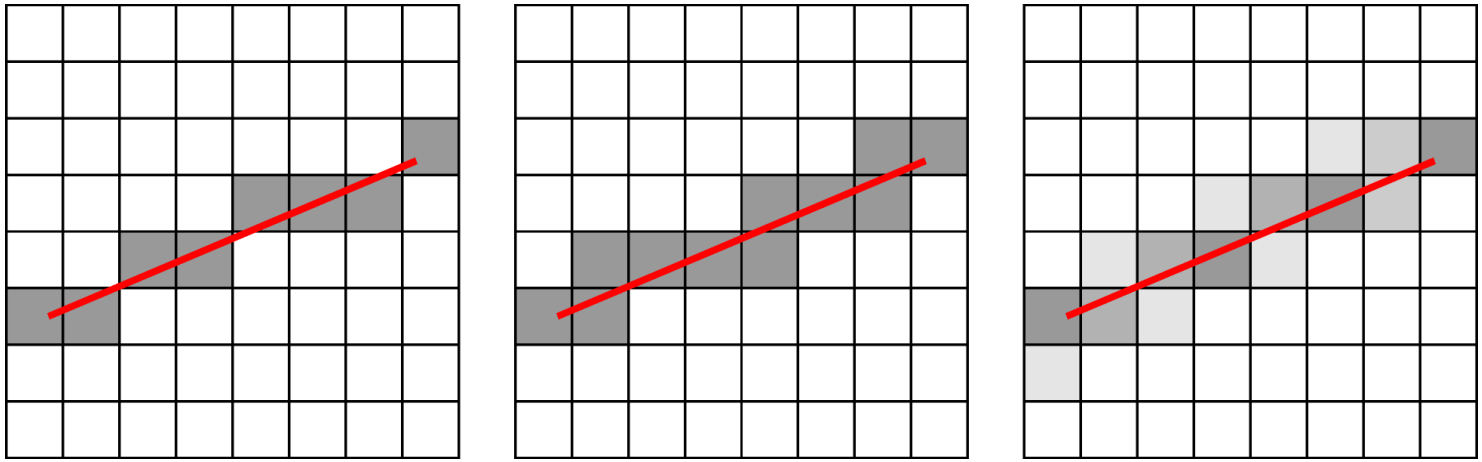
Primitivos gráficos - algoritmos

Prof. Julio Arakaki

Ciência da Computação

Algoritmos de reta

- Reta => *infinitamente fina*, ou seja, área = 0
- Problemas para apresentação num monitor "raster" utilizando-se pixels:
  - deve-se escolher quais pixels melhor representa a reta
  - existem muitos algoritmos que possibilitam diferentes resultados



Algoritmo de reta – equação reduzida

Através da equação de reta: y = mx + b

```
…
for(int x = x1; x <= x2; x++)
{
    int y = (int)round(m*x + b);
    desenharPonto(x, y, cor);
}
…
```

Problemas:

- Linhas verticais. (m = ∞)
- Lento, pois para todo x tem-se:
  uma multiplicação em “floating point”
  uma adição em “floating point”
  uma chamada de função de arrendondamento (“round”)

Algoritmo de reta – Digital Differential Analyser (*DDA*)

Algoritmo simples e também não muito eficiente

– *incremental*

- Incrementa x de 1 ($\Delta x = 1$) e atualiza y de acordo com:

$$
\begin{aligned}
y_1 &= mx_1 + b \\
x_2 &= x_1 + 1 \\
y_2 &= mx_2 + b \\
&= m(x_1 + 1) + b \\
&= (mx_1 + b) + m \\
&= y_1 + m
\end{aligned}
$$

- Desenha o pixel $(x, round(y))$ em cada iteração

Algoritmo de reta – Digital Differential Analyser (*DDA*) - *exemplo*

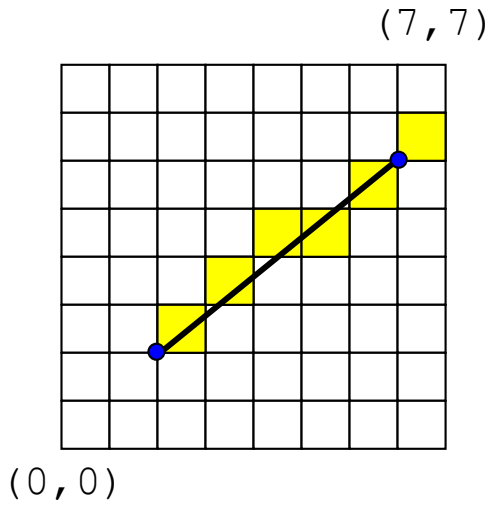
reta: (2,2) → (7,6)

$m = 0.8$

$\Delta x = 1$

$y = y + m$

x	y	Round(y)
2.0	2.0	2
3.0	2.8	3
4.0	3.6	4
5.0	4.4	4
6.0	5.2	5
7.0	6.0	6




```
desenharPonto(x, round(y));
```

Algoritmo de reta – Digital Differential Analyser (*DDA*)

```
…
float y;
float m = (y2-y1)/(x2-x1);

desenharPonto(x1,y1, cor);
y = y1;

while( x1 < x2 ){
   x1++;
   y += m;
   desenharPonto(x1, ROUND(y), cor);
}
…
```

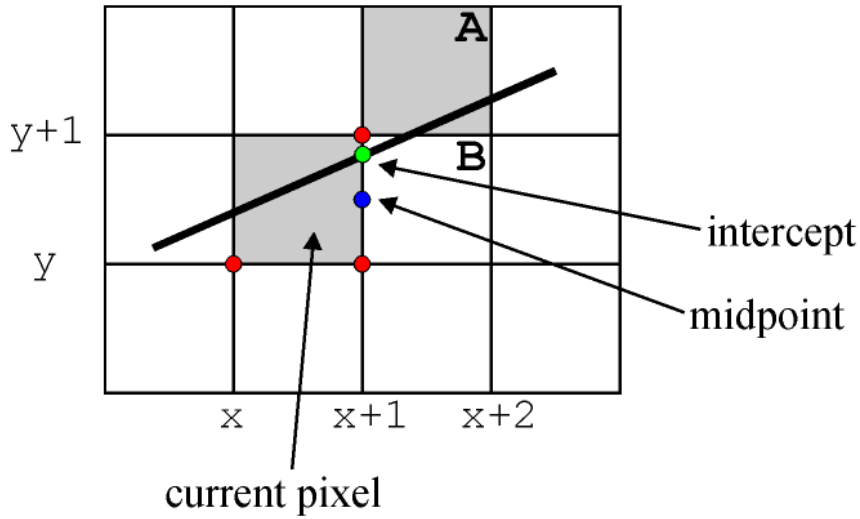
x é um inteiro, mas y é floating point.

custo: incremento, adição em floating point e chamada da função de arredondamento "round(y)" para cada valor de x.

Algoritmo de reta – "Midpoint algoritm" (Bresenham – 1965)

- Mais eficiente que DDA.
  - *incremental* e cálculo baseado em *inteiros*

- Em cada iteração determinamos se a reta (real) intersecciona o próximo pixel acima ("above") ou abaixo ("below") da metade ("*midpoint*") do valor de *y*.
  - se acima então `A`: $y_{i+1} = y_i + 1$
  - caso contrário `B`: $y_{i+1} = y_i$



Algoritmo de reta – "Midpoint algoritm"

- Utiliza a forma implícita da equação de reta:

$ax_i + by_i + c = 0 : (x_i, y_i)$ Na reta
$ax_i + by_i + c < 0 : (x_i, y_i)$ Acima da reta
$ax_i + by_i + c > 0 : (x_i, y_i)$ Abaixo da reta

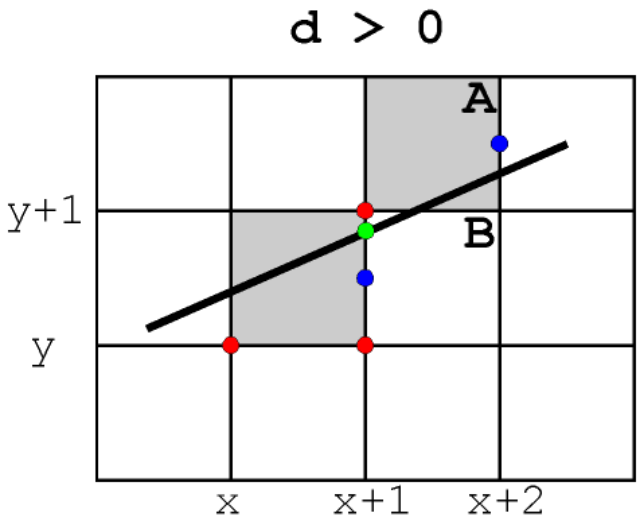
- A cada iteração, verifica se o "midpoint" está acima ou abaixo da reta

  Examina-se o valor de:

$$d_i = a(x_i + 1) + b(y_i + ½) + c$$

d_i é uma variável de decisão no passo i

Algoritmo de reta – "Midpoint algoritm"



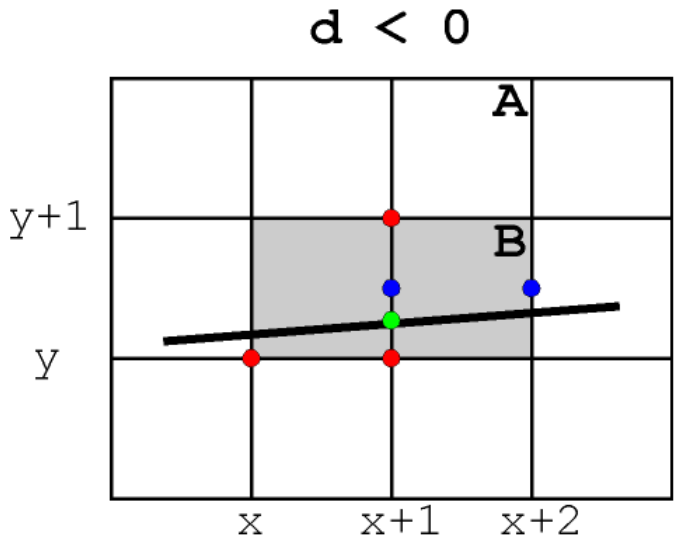
- 1) Se $d_i > 0$ então escolhe-se o pixel A $\Rightarrow$ novo "midpoint" deverá ser checado:

$$(x_i + 2,\ y_i + 3/2) : d_{i+1} = a(x_i + 2) + b(y_i + 3/2) + c$$

$$= [a(x_i + 1) + b(y_i + 1/2) + c] + a + b =$$

$$= d_i + a + b$$

Algoritmo de reta – "Midpoint algoritm"



- 2) Similarmente, se $d_i < 0$ então escolhe-se o pixel B e o novo "midpoint" é:

$$(x_i + 2,\ y_i + 1/2) : d_{i+1} = a(x_i + 2) + b(y_i + 1/2) + c$$

$$= [a(x_i + 1) + b(y_i + 1/2) + c] + a = d_i + a$$

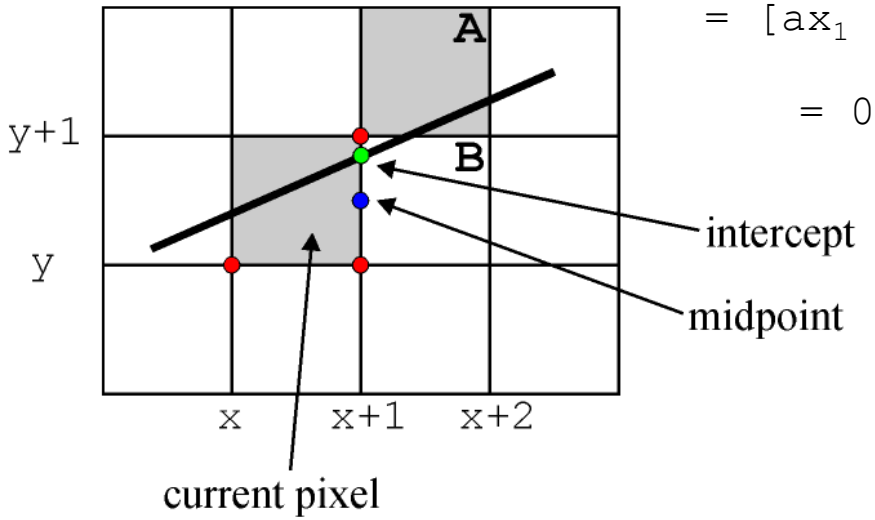
Algoritmo de reta – "Midpoint algoritm"

- No ponto inicial, a variável de decisão é calculada da seguinte forma:

$$(x_1+1,\ y_1+1/2):\ d_1 = a(x_1 + 1) + b(y_1 + 1/2) + C$$

$$= [ax_1 + by_1 + c] + a + b/2$$

$$= 0 + a + b/2$$



Uma parte fracionária é introduzida (*b/2*), mas como interessa-se somente o sinal de d, Pode-se *multiplicar a expressão por 2:*

$$d_i = 2a + b$$

Algoritmo de reta – "Midpoint algoritm"

- Sabendo-se que $y=(dy/dx)*x + C$, temos: $(dy/dx)*x - y + C = 0$

  Multiplicando por dx todos os termos temos:

  $$dy*x - dx*y + dx*c = 0$$

  Então:

  $$a = dy = y_2 - y_1$$

  $$b = -dx = -(x_2 - x_1)$$

  $$c = Cdx$$

Algoritmo de reta – "Midpoint algoritm"

OU SEJA,

Calculo da variável de decisão:

- Inicio: substituindo os valores de a e b obtemos:

$$d_i = 2a + b = 2dy - dx$$

- Para $d_i > 0$:

Multiplicando por 2 e substituindo a e b :

$$d_{i+1} = d_i + 2(a + b) = d_i + 2(dy - dx)$$

- Para $d_i < 0$:

Multiplicando por 2 e substituindo a e b :

$$d_{i+1} = d_i + 2a = d_i + 2(dy)$$

Algoritmo de reta – "Midpoint algoritm"

```
…
dx = x2 - x1;
dy = y2 - y1;
d = 2*dy - dx;
x = x1;
y = y1;
desenharPonto(x, y, cor);
while (x < x2) {
    if (d <= 0){
        d = d + (2*dy);
    }
    else {
        d = d + 2*(dy - dx);
        y = y + 1;
    }
    x = x + 1;
    desenharPonto(x, y, cor);
}
…
```

inicialização

seleciona B

seleciona A

Algoritmo de reta – "Midpoint algoritm"

Exemplo:

Traçar a reta passando por: P1(2,2) e P2(7,6)

```
dx = (7 - 2) = 5
dy = (6 - 2) = 4
Inicial: (d = 2*dy - dx)
Ou seja,  d = 3
```

```
se d > 0 (d = d + 2 * dy)
   ou seja, d = d - 2


se d < 0 (d = d + 2*(dy - dx))
   ou seja, d = d + 8
```

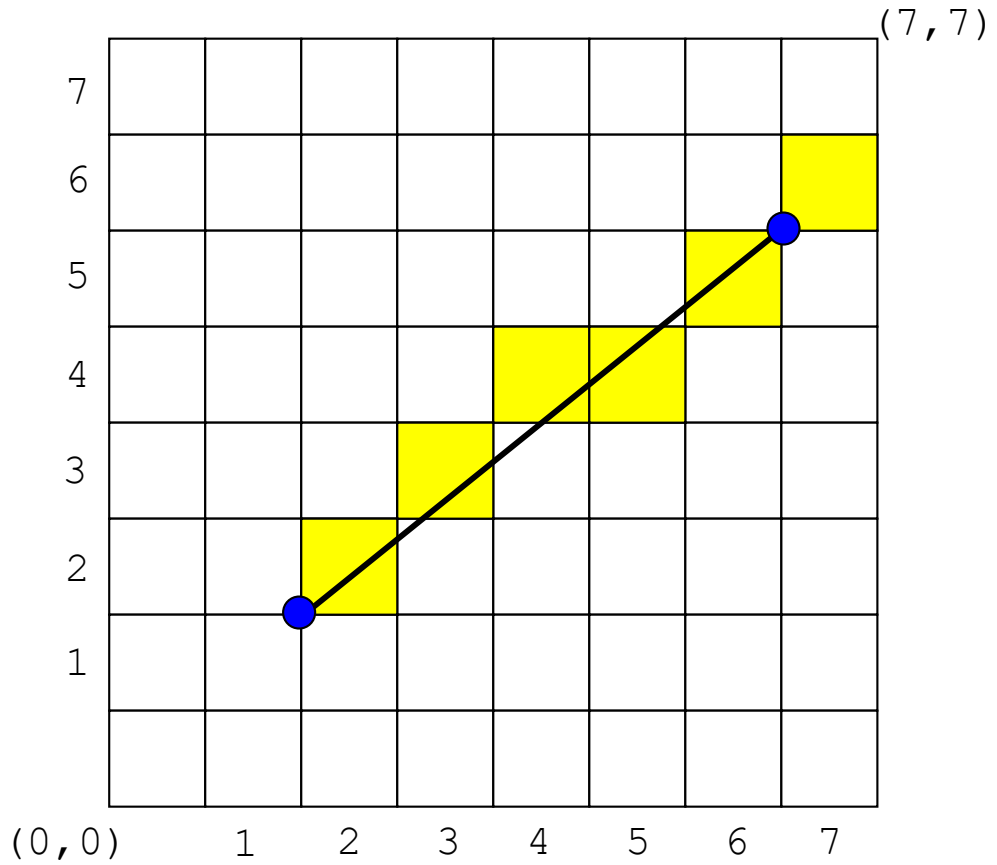
Algoritmo de reta – "Midpoint algoritm"

reta: (2,2) → (7,6)

dx = 5
dy = 4
d = 3

```
if d>0 then d=d-2, y=y+1
if d<0 then d=d+8
x=x+1
```

x	y	d
2	2	3
3	3	1
4	4	-1
5	4	7
6	5	5
7	6	3



Algoritmo de círculo – "Midpoint algoritm"

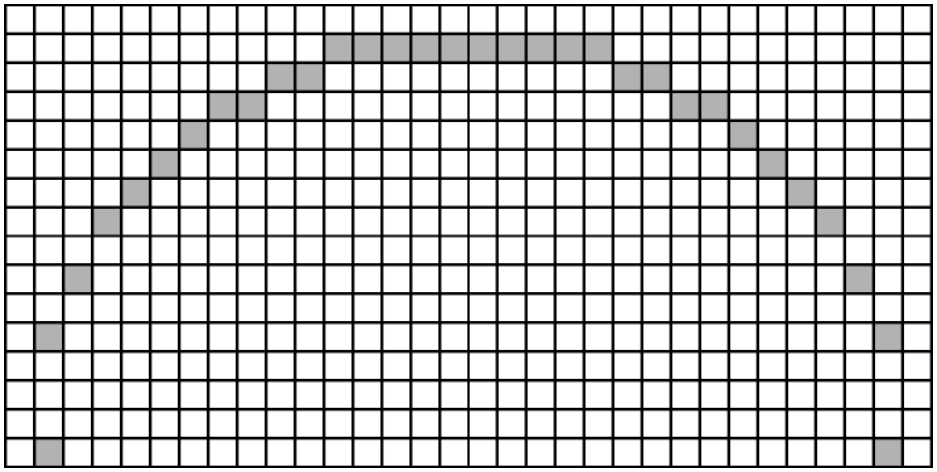
- Círculo com raio r e centro (x_c, y_c) é definido parametricamente como:

$$x = x_c + r\cos\theta$$

$$y = y_c + r\,\mathrm{sen}\,\theta$$

$\therefore$ Variando θ de 0 até 2π plotando-se as coordenadas:

  – Dificuldade para efetivamente controlar a dimensão do passo, para eliminar os espaços entre os pixels



Algoritmo de círculo – "Midpoint algoritm"

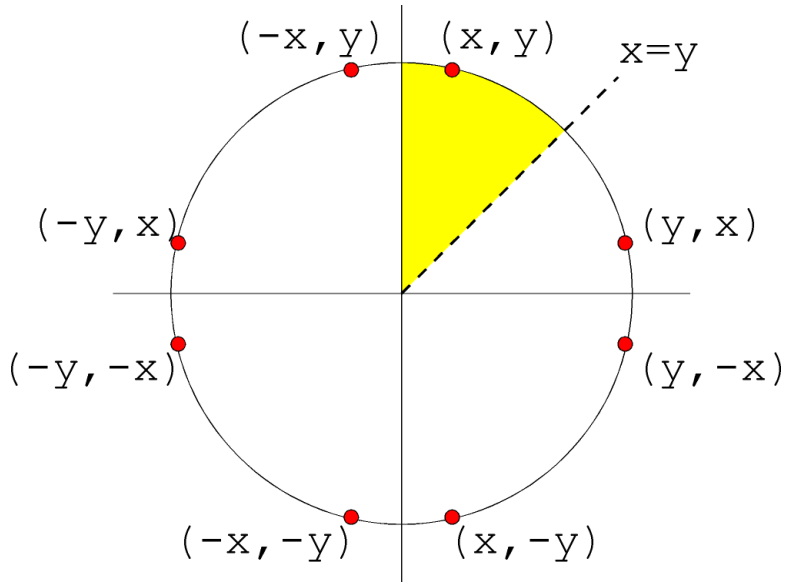
- Forma implícita do círculo: $(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$
- Utiliza um esquema similar ao algoritmo de reta ("midpoint"):
  - Necessário a determinação dos pixels de um *octante*, os pixels de outros octantes são determinados pela simetria
  - A variável de decisão é calculada da seguinte forma:

    d_i:

    < 0 se (x_i, y_i) dentro do círculo

    $= 0$ se (x_i, y_i) no círculo

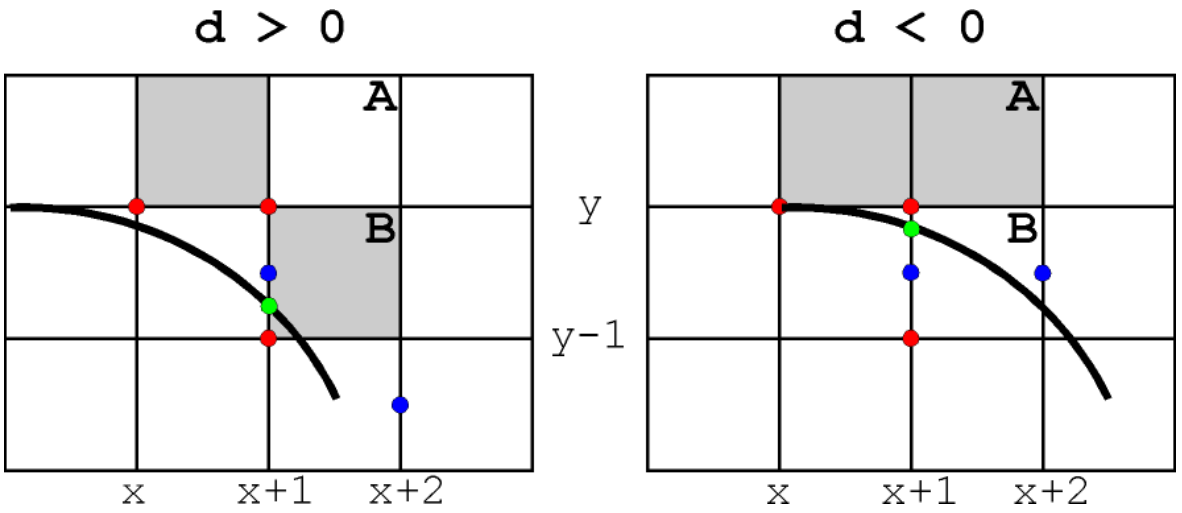
    > 0 se (x_i, y_i) fora do círculo



Algoritmo de círculo – "Midpoint algoritm"

- Como na reta, determina-se o valor da variável de decisão pela substituição do "midpoint" do próximo pixel pela forma implícita do círculo:

$$d_i = (x_i + 1)^2 + (y_i - 1/2)^2 - r^2$$



  - Se $d_i < 0$ escolhe-se pixel `A` senão escolhe-se pixel `B`

Algoritmo de círculo – "Midpoint algoritm"

- Semelhante ao algoritmo de reta, a escolha de A ou B pode ser utilizado para determinar o novo valor de d_{i+1}
- Se A for escolhido, então o próximo "midpoint" tem o seguinte variável de decisão:

$$(x_i + 2,\ y_i - 1/2) : d_{i+1} = (x_i + 2)^2 + (y_i - 1/2)^2 - r^2$$

$$= d_i + 2x_i + 3$$

- Por outro lado, se B for escolhido, então a próxima variável de decisão é:

$$(x_i + 2,\ y_i - 3/2) : d_{i+1} = (x_i + 2)^2 + (y_i - 3/2)^2 - r^2$$

$$= d_i + 2x_i - 2y_i + 5$$

Algoritmo de círculo – "Midpoint algoritm"

- Assumindo-se que o raio é um valor *integral*, então o primeiro pixel a ser desenhado é *(0, r)* e o valor inicial da vaariável de decisão é dado por:

$$(1,\ r-1/2)\ :\ d_0 = 1 + (r^2 - r + 1/4) - r^2$$

$$= 5/4 - r$$

- Neste caso o valor é fracionário, e todos os outros são inteiros.

  Pode-se arredondar para:

$$d_0 = 1 - r$$

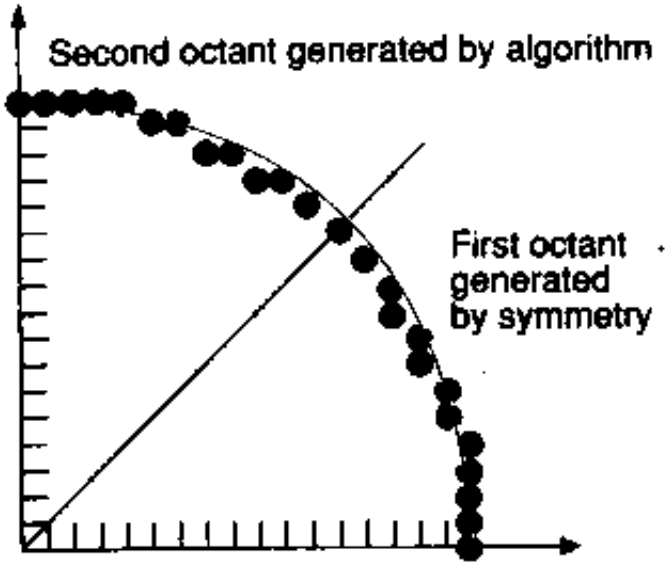
Algoritmo de círculo – "Midpoint algoritm"

```
…
d = 1-r;
x = 0;
y = r;
desenharPonto(cx+x,cy+y);
while (x < y) {
    if (d < 0) {
        d = d + 2*x + 3;
    }
    else{
        d = d + 2*(x - y) + 5;
        y = y - 1;
    }
 x = x + 1;
    desenharPonto(cx+x,cy+y);
}
…
```

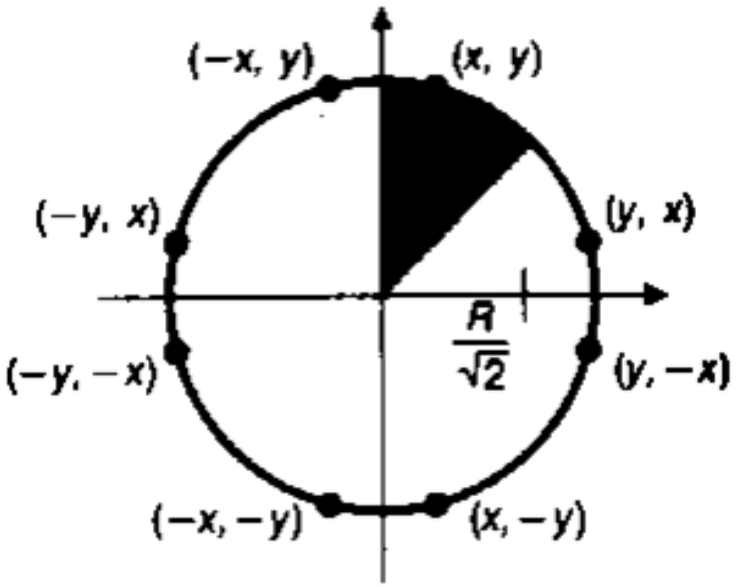
inicialização

seleciona A

seleciona B

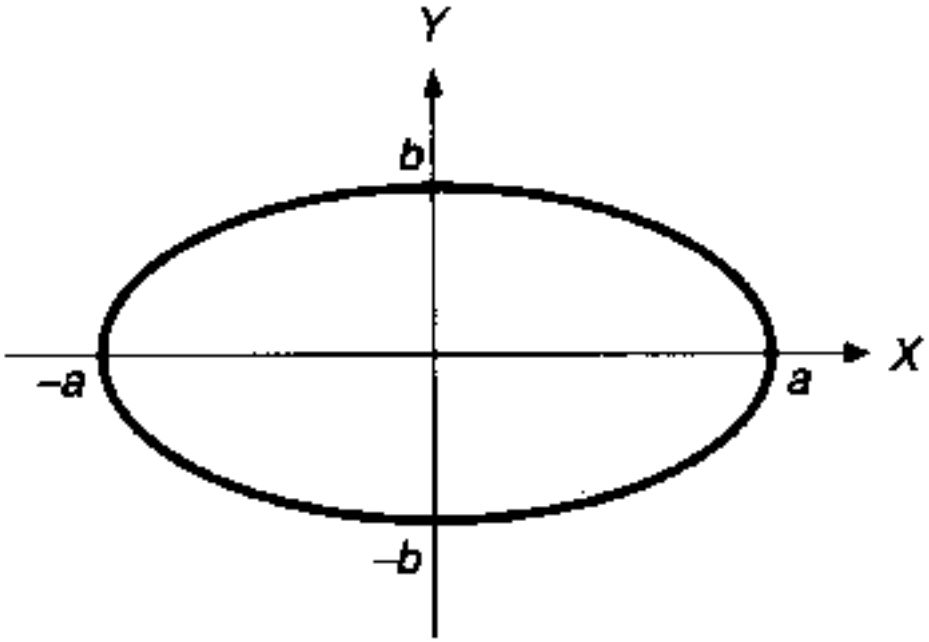


Onde (c_x, c_y) é a coordenada do centro




```
void CirclePoints (int x, int y, int value)
{
    WritePixel (x, y, value);
    WritePixel (y, x, value);
    WritePixel (y, −x, value);
    WritePixel (x, −y, value);
    WritePixel (−x, −y, value);
    WritePixel (−y, −x, value);
    WritePixel (−y, x, value);
    WritePixel (−x, y, value);
} /* CirclePoints */
```

Algoritmo de elipse – "Midpoint algoritm"



$$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

Algoritmo de elipse – "Midpoint algoritm"

```
void MidpointEllipse (int a, int b, int value)
/* Assumes center of ellipse is at the origin. Note that overflow may occur */
/* for 16-bit integers because of the squares. */
{
    double d2;

    int x = 0;
    int y = b;
    double d1 = b^2 - (a^2 b) + (0.25 a^2);
    EllipsePoints (x, y, value);                    /* The 4-way symmetrical WritePixel */

    /* Test gradient if still in region 1 */
    while ( a^2 (y - 0.5) > b^2 (x + 1) ) {         /* Region 1 */
        if (d1 < 0)                                 /* Select E */
            d1 += b^2 (2x + 3);
        else {                                      /* Select SE */
            d1 += b^2 (2x + 3) + a^2 (-2y + 2);
            y--;
        }
        x++;
        EllipsePoints (x, y, value);
    }  /* Region 1 */
```

Algoritmo de elipse – "Midpoint algoritm"

```
d2 = b²(x + 0.5)² + a²(y − 1)² − a²b²;
while (y > 0) {                          /* Region 2 */
    if (d2 < 0) {                        /* Select SE */
        d2 += b²(2x + 2) + a²(−2y + 3);
        x++;
    } else
        d2 += a²(−2y + 3);               /* Select S */
    y−−;
    EllipsePoints (x, y, value);
} /* Region 2 */
} /* MidpointEllipse */
```