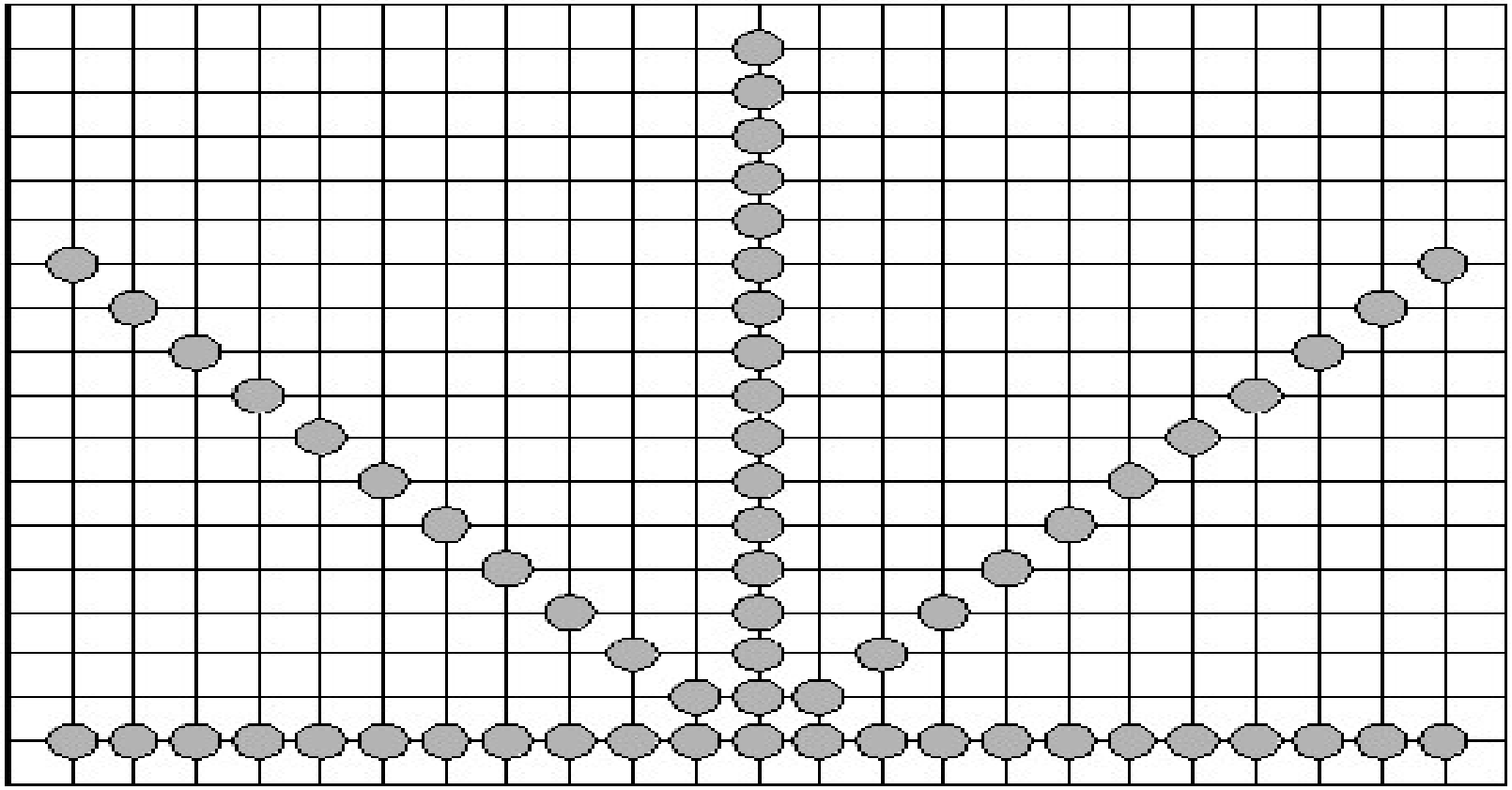
Trazado de curvas en dispositivos gráficos matriciales

Algoritmos de conversión matricial

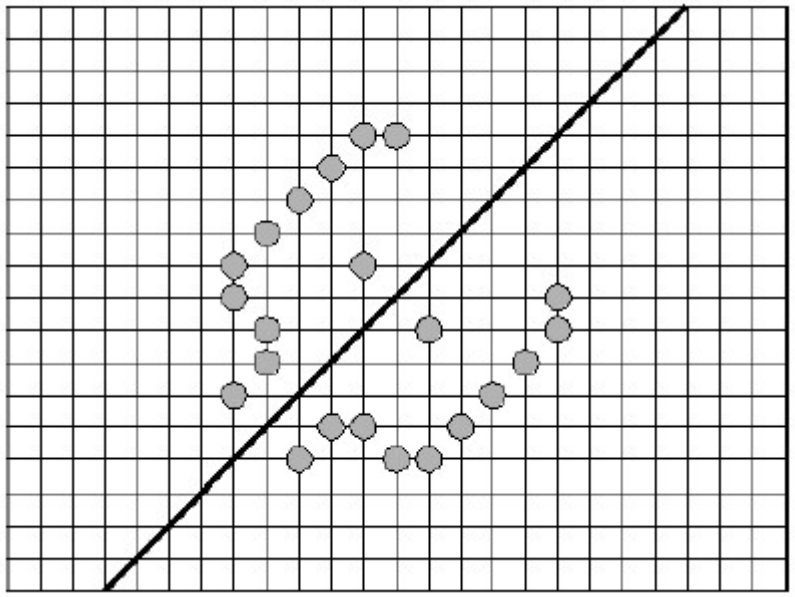
- El trazado de primitivas básicas elementales como segmentos de recta o arcos de circunferencia requieren de algoritmos capaces de determinar en la matriz de píxeles de exhibición cuales píxeles deben ser alterados para de forma de simular la apariencia grafica del elemento grafico deseado.

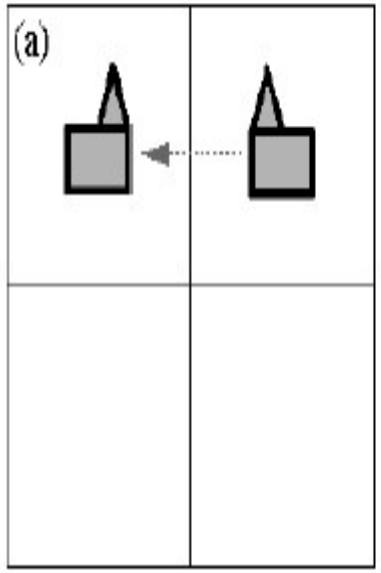
Representaciones de recta

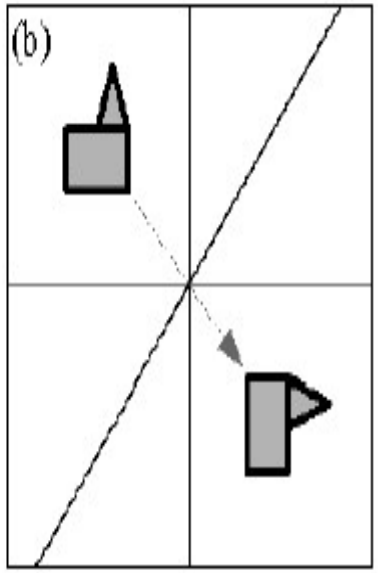


Simetría y reflexión

- Las líneas que pasan con coordenadas exactas forman ejes de simetría en el espacio matricial
- Cualquier imagen en el espacio matricial puede ser reflejada en relación a cualquier recta horizontal, vertical o diagonal sin presentar ninguna deformación



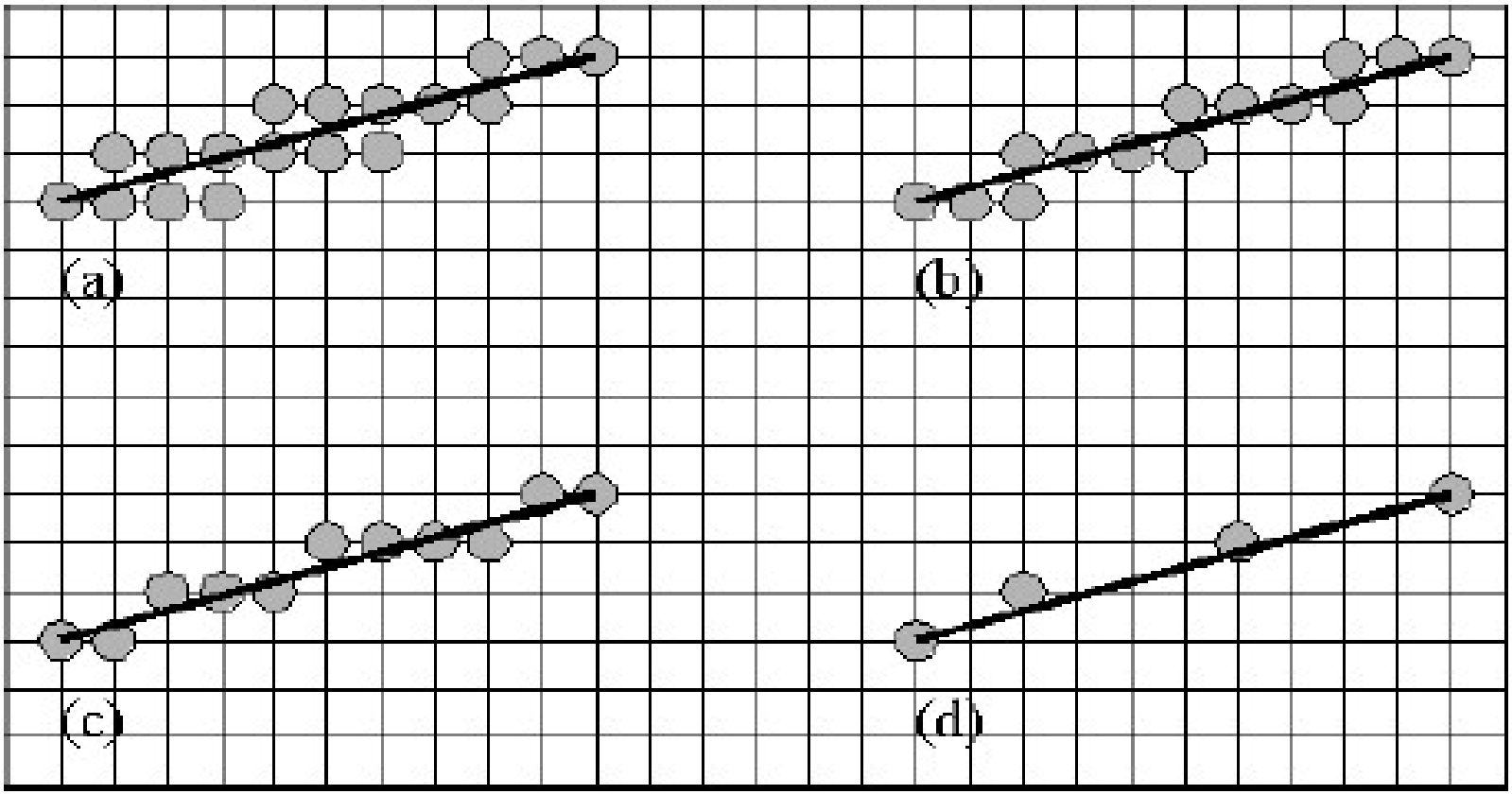




Conversión matricial de segmentos de recta

- Seleccionar 2 píxeles inmediatamente encima y debajo del punto de intersección del segmento
- Todos los píxeles cuyos coordenadas son obtenidas por el redondeo de las coordenadas de algún punto del segmento
- Seleccionar en cada vertical el píxel mas próximo del punto de intersección del segmento con la recta vertical (continuidad)
- Seleccionar en cada horizontal el píxel mas próximo del punto de intersección del segmento con la recta horizontal

Conversión matricial de segmentos de recta



Características para algoritmos conversión matricial

- Linealidad
  - dar apariencia de estar sobre una recta
- Precisión
  - Empezar y terminar en los puntos especificados
- Espesura uniforme
  - El segmento no varia de intensidad a lo largo de su extensión
- Intensidad independiente de la inclinación
- Continuidad
  - No aparentar interrupciones indeseables
- Rapidez en el trazado de los segmentos

Algoritmo incremental básico

- Sean $x1$, $y1$ y $x2$, $y2$, además el segmento se encuentra en el 1er octante

$$0 < x1 < x2$$

$$0 < y1 < y2$$

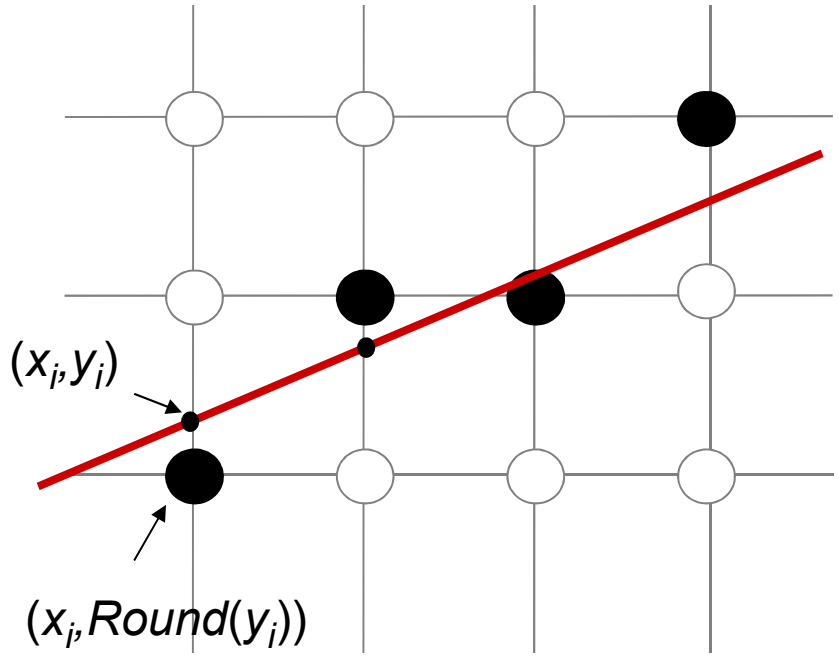
$$y_2 - y_1 < x_2 - x_1$$

- Si esto sucede entonces en cada vertical de la malla con abscisa entre $x1$ y $x2$ apenas el píxel mas próximo dela intersección del segmento con la vertical forma parte de su imagen

Algoritmo incremental básico

$y_i = mx_i + B \quad \forall i$

y luego pintar el pixel $(x_i, round(y_i))$

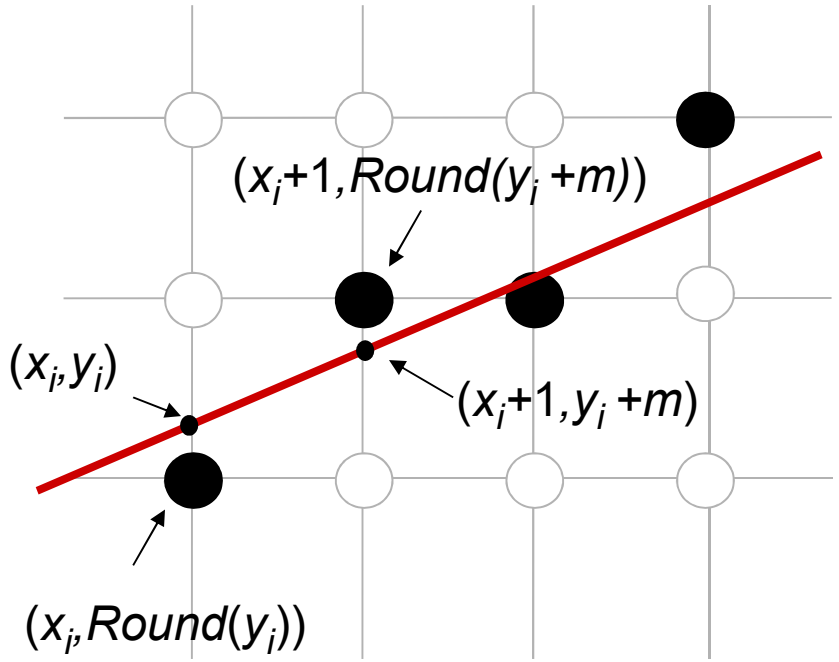


Algoritmo incremental básico

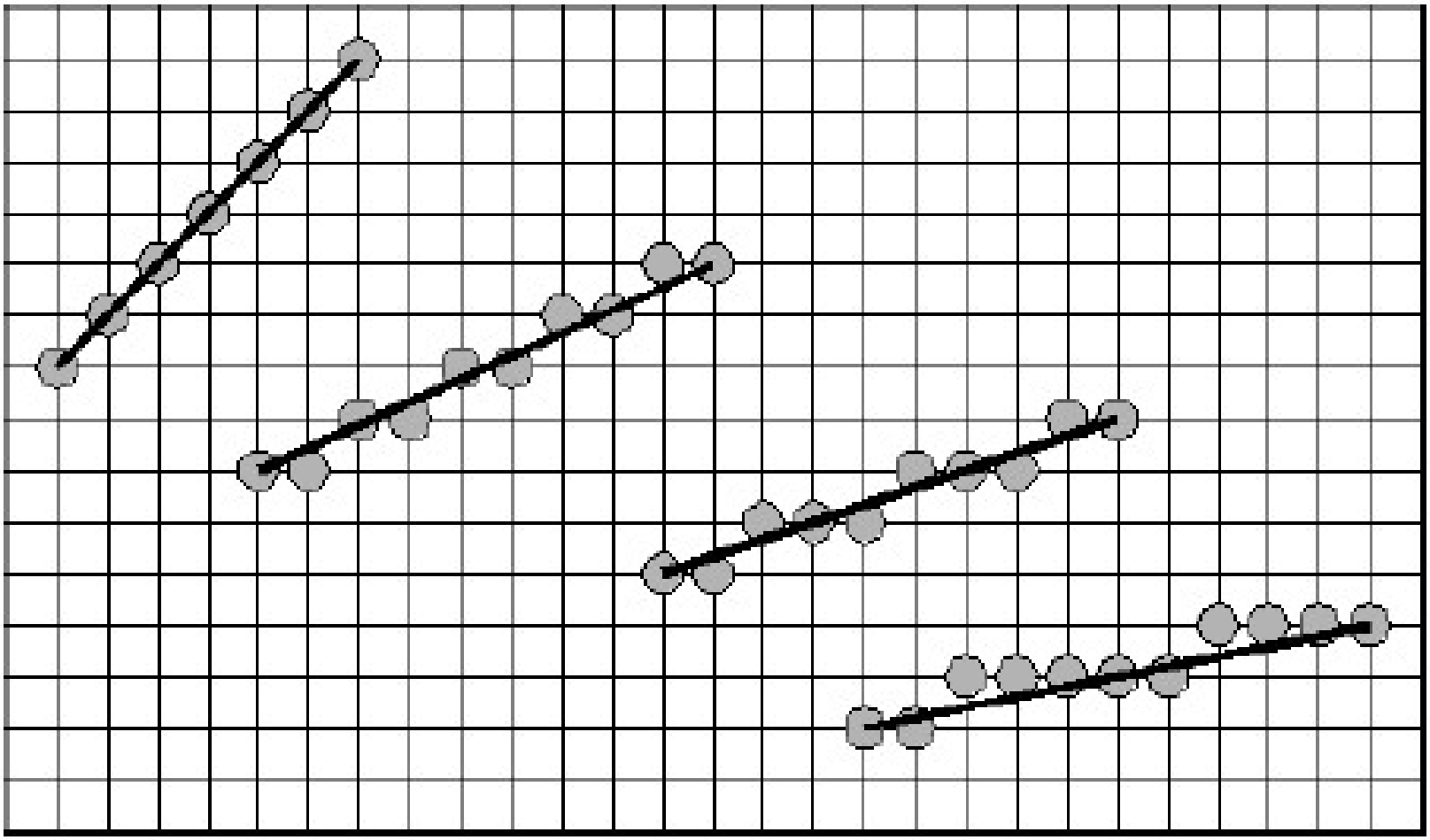
$y_i = m x_i + B \qquad \forall i$

$y_{i+1} = m x_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$

Si $\Delta x = 1 \Rightarrow y_{i+1} = y_i + m$



Algoritmo incremental básico

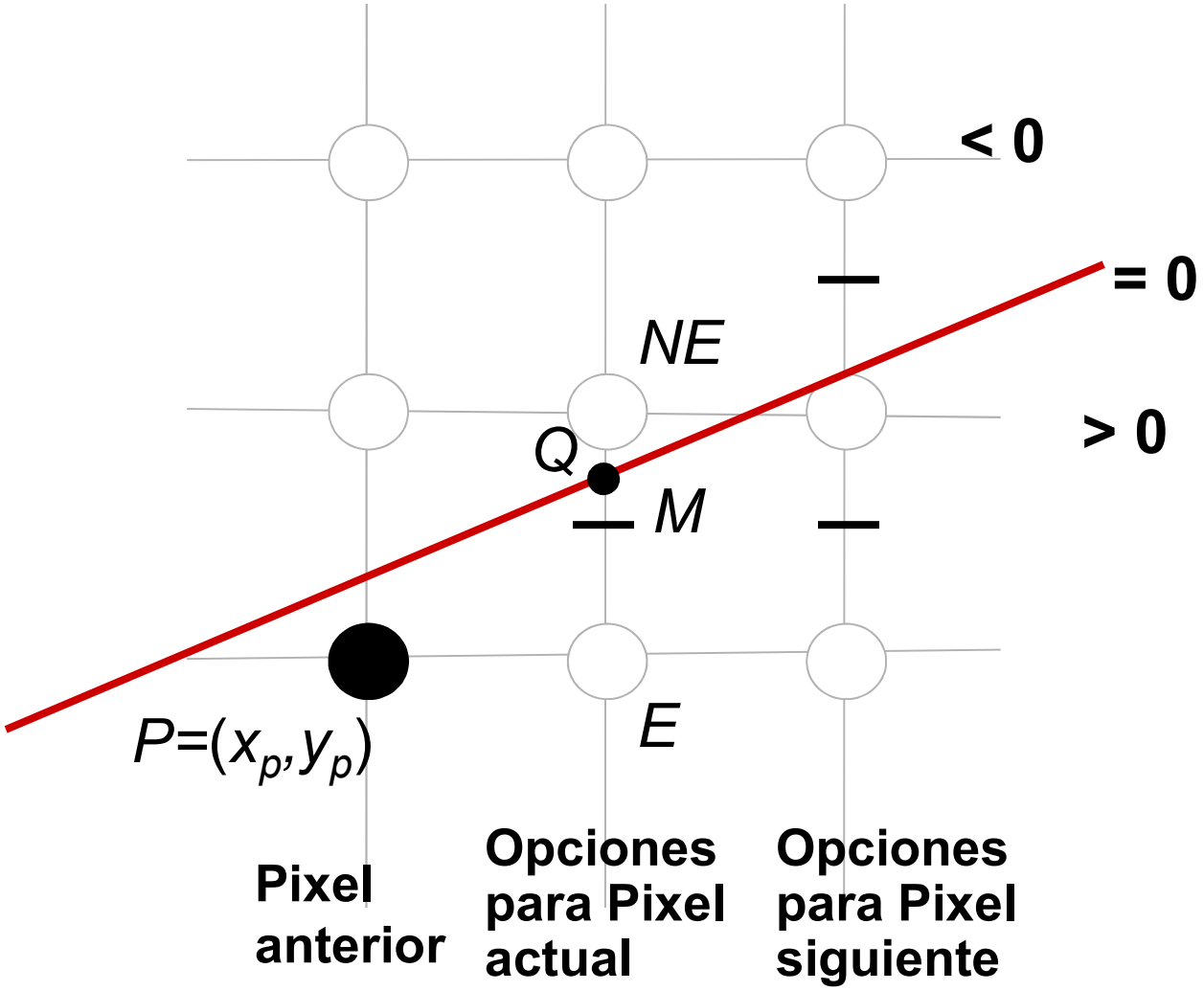


Algoritmo incremental básico

```
void line(int x1, int y1, int x2, int y2, int color){
   int x, y;
   float a;
   int valor;

   a=(y2-y1)/(x2-x1);
   for (x=x1;x<=x2;x++){
      /* arredonda y */
      y = (y1 + a * (x - x1));
      write_pixel(x, y, color);
   }/* end for */
}/*end line */
```

Algoritmo del punto medio



Algoritmo del punto medio

La representación explícita de la recta:

$$dy = y2 - y1;\ dx = x2 - x1$$

$$y=(dy/dx)\cdot x + B$$

Se transforma en otra implícita:

$$F(x,y)= a\cdot x + b\cdot y + c = dy\cdot x - dx\cdot y + B\cdot dx = 0$$

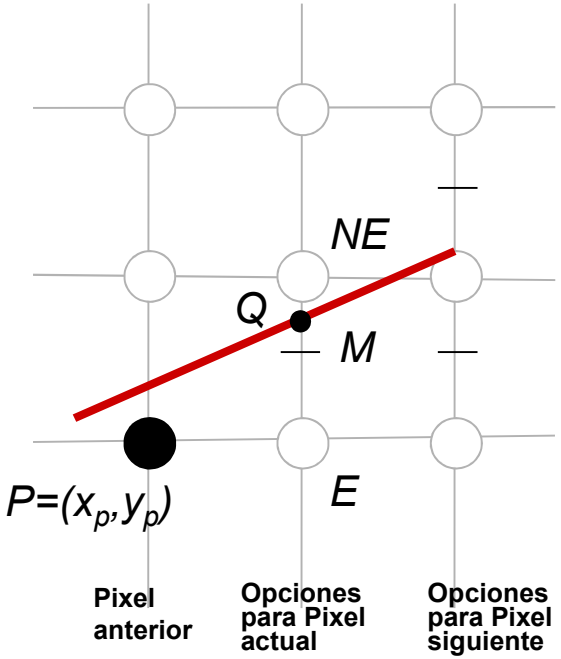
Donde:

$$a=dy\ ;\ b=-dx\ ;\ c= B\cdot dx$$

Algoritmo del punto medio

Propiedades de F(x,y)

F(x,y)=0 en la línea
F(x,y)>0 en los puntos debajo de la línea
F(x,y)<0 en los puntos sobre la línea



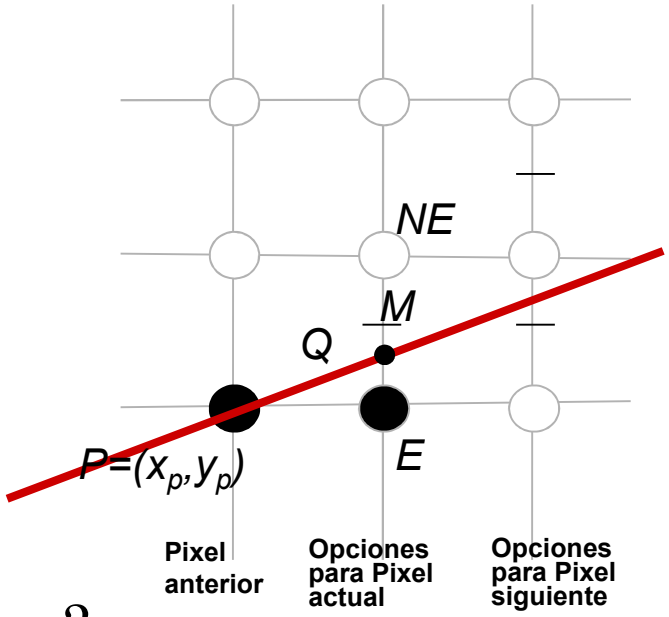
$\Rightarrow$ Aplicamos F al punto M ; d = F(M)=F(x_p+1,y_p+½)

Si $d > 0$ $\Rightarrow$ se elige el pixel NE
Si $d <= 0$ $\Rightarrow$ se elige el pixel E

Algoritmo del punto medio

$d = a(x_p + 1) + b(y_p + ½) + c$

Llamemos d_{viejo} a este d



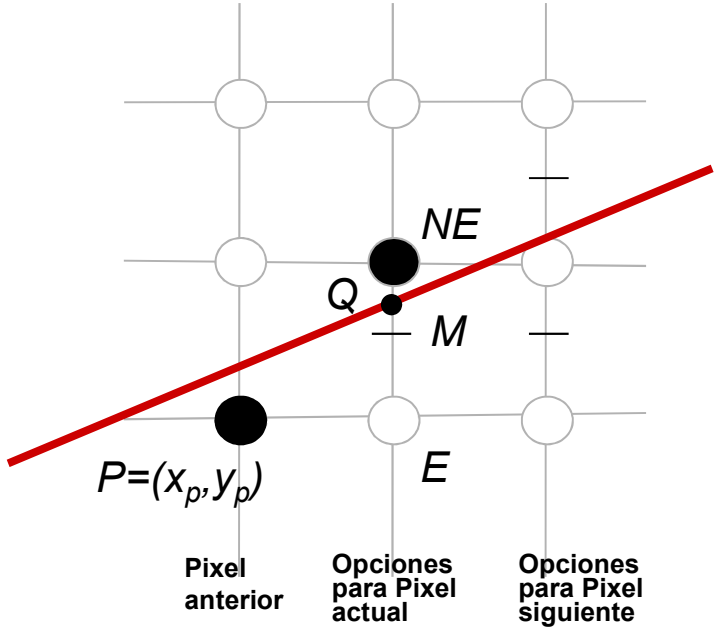
Si se eligió el pixel *E,* ¿cuál es el valor d_{nuevo}?

$d_{nuevo} = a(x_p + 2) + b(y_p + ½) + c$

$$\boldsymbol{d_{nuevo} = d_{viejo} + a = d_{viejo} + dy = d_{viejo} + \Delta_E}$$

Algoritmo del punto medio

$d_{viejo} = a(x_p + 1) + b(y_p + ½) + c$



Si se eligió el pixel *NE*, ¿cuál es el valor d_{nuevo}?

$d_{nuevo} = a(x_p + 2) + b(y_p + 3/2) + c$

$$\boldsymbol{d_{nuevo} = d_{viejo} + (a+b) = d_{viejo} + (dy-dx) = d_{viejo} + \Delta_{NE}}$$

Repaso Algoritmo del punto medio

Quiero dibujar el segmento entre (x_0,y_0) y (x_{fin},y_{fin}) que tienen coordenadas enteras => la recta tiene coeficientes enteros.

El primer punto medio está en

$F(x_0+1,y_0+½)=F(x_0,y_0)+a+b/2= a+b/2$

Dibujo (x_0,y_0)

Defino $d_{nuevo}= a + b/2$

$p=0$

Mientras no llegué a x_{fin}

Según el signo de d_{nuevo}

elijo y dibujo $(x_{p+1},y_{p+1}) \Rightarrow M \Rightarrow \Delta \Rightarrow d_{nuevo}$

$p=p+1$

Fin

Repaso Algoritmo del punto medio

El problema es que d_{inicio}= a + b/2, por lo que hay una fracción inicial que perjudica todos los cálculos posteriores.

¿Solución?: Multiplicar la función F por 2

F(x,y)=2(ax + by + c)

Conclusiones:

Para cada paso, d_{nuevo} se calcula a partir de una suma entera.

El algoritmo se puede generalizar para líneas con pendientes que no estén entre 0 y 1.

```
void inc_line(int x1, int y1, int x2, int y2, int color){
   int dx, dy, incE, incNE, d, x, y;

   dx = x2 - x1;
   dy = y2 - y1;
   d = 2 * dy - dx; /* Valor inicial de d */
   incE = 2 * dy; /* Incremento de E */
   incNE = 2 * (dy - dx); /* Incremento de NE */
   x = x1;
   y = y1;
   write_pixel(x, y, color);
   while (x < x2){
      if (d <= 0){
         /* Escolhe E */
         d = d + incE;
         x = x + 1;
      }else{
       /* Escolhe NE */
       d = d + incNE;
       x = x + 1;
       y = y + 1;
      }/* end if */
      write_pixel(x, y, color);
   }/* end while */
}/* end inc_line */
```

Conversión matricial para circunferencias

- Existen muchos abordajes para el trazado de circunferencias
  - En algoritmos simples no incrementales un polígono de *n* lados es usado como aproximación para una circunferencia
  - Cuanto mayor sea el valor de *n* mas lento será el proceso

Conversión matricial para circunferencias

Ecuación del círculo centrado en el origen:

$$X^2 + y^2 = R^2$$

Función explícita:

y = f(x) entonces para la circunferencia

$$y=\pm sqrt(R^2-x^2)$$

y al discretizar queda:

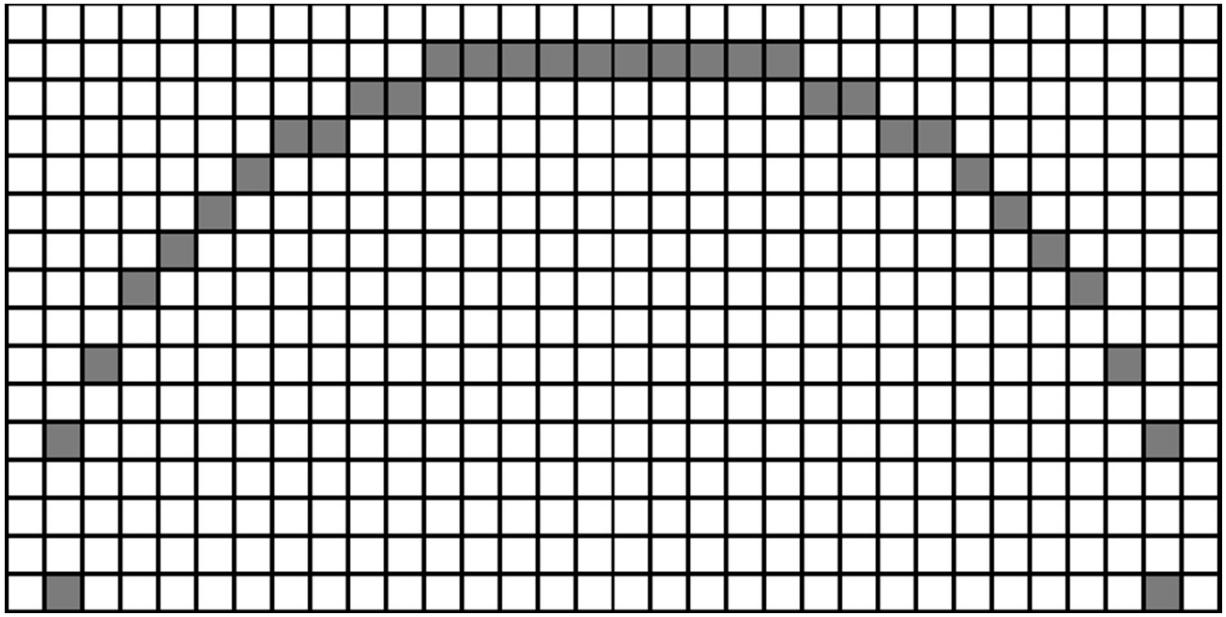
$$y=round(\pm sqrt(R^2-x^2))$$

Problemas en la forma tradicional de discretización de círcunferencias

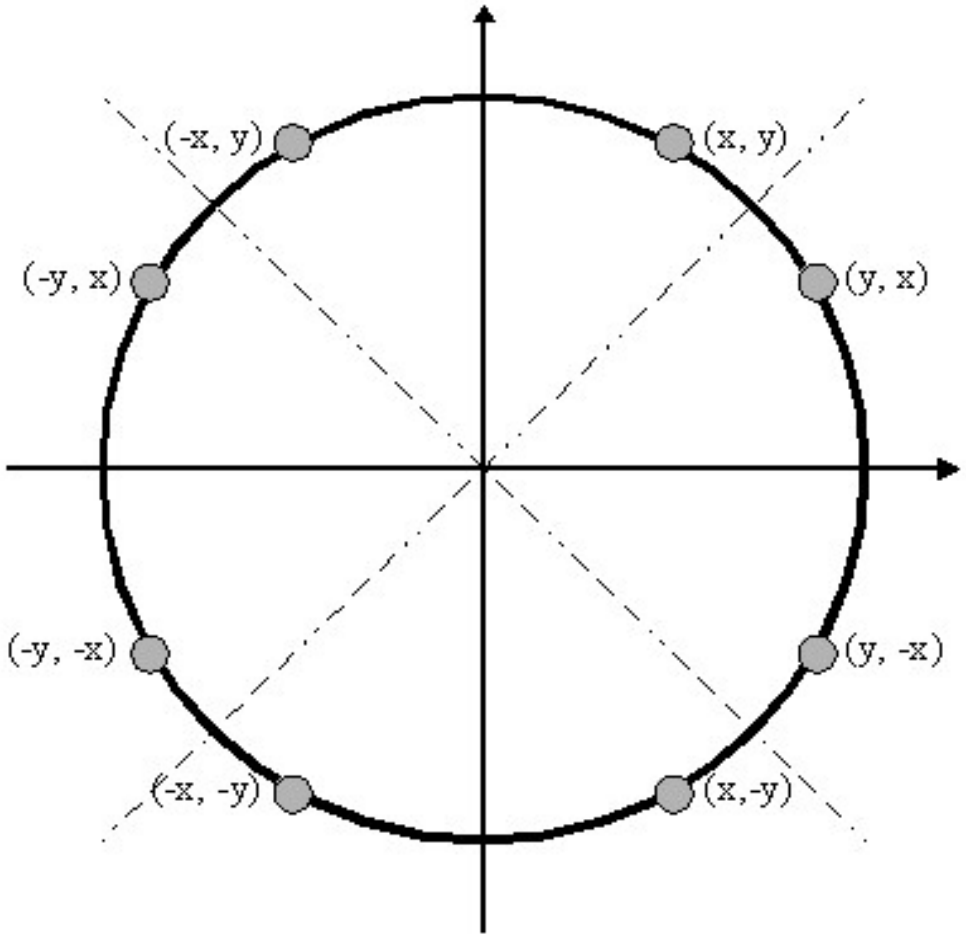
Precisa utilizar raiz cuadrada y redondeo

Uso de punto flotante

Dibujo de la circunferencia con huecos



Simetría de 8 lados

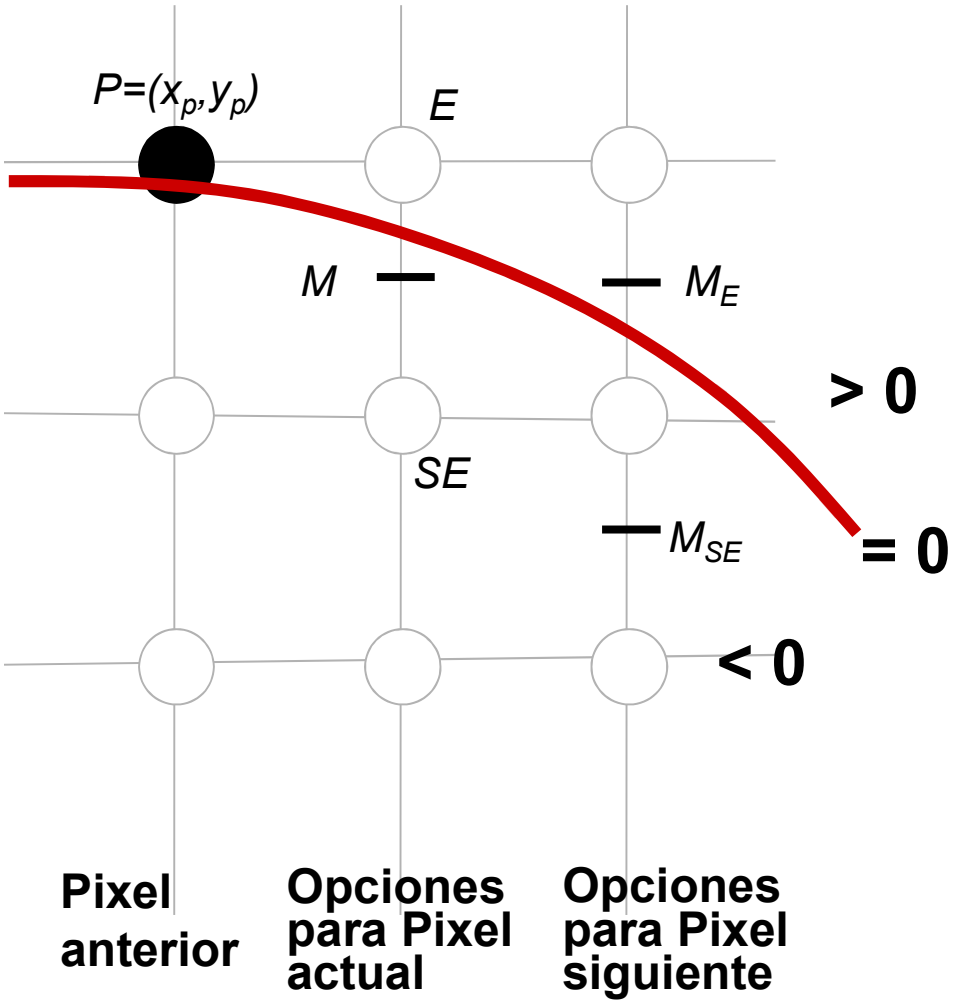


Algoritmo de simetría de orden 8

```
void CirclePoints(int x, int y, int color){

   write_pixel( x,  y, color);
   write_pixel( x, -y, color);
   write_pixel(-x,  y, color);
   write_pixel(-x, -y, color);
   write_pixel( y,  x, color);
   write_pixel( y, -x, color);
   write_pixel(-y,  x, color);
   write_pixel(-y, -x, color);
}/* end CirclePoints */
```

Algoritmo del circulo de punto medio para circunferencias



Algoritmo del circulo de punto medio para circunferencias

Sea la ecuación de la circunferencia una función implicita

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

Con valor 0 *si el punto* ME *corresponde a un valor en la circunferencia y positivo si esta fuera de ella y negativo si esta dentro de ella*

Algoritmo del circulo de punto medio

Se considera solo 45° de un círculo, de x=0 a x=y=R/sqrt(2) .

$$d_{viejo} = F(x_p+1, y_p-½) = (x_p+1)^2 + (y_p-½)^2 - R^2 = 0$$

Si $d_{viejo} < 0$ se escoge E y luego el punto medio es M_E

$$d_{nuevo} = F(x_p+2, y_p-½) = (x_p+2)^2 + (y_p-½)^2 - R^2$$

$$\boldsymbol{d_{nuevo} = d_{viejo} + (2x_p + 3) \Rightarrow \Delta_E = (2x_p + 3)}$$

Si $d_{viejo} > 0$ se escoge SE y luego el punto medio es M_{SE}

$$d_{nuevo} = F(x_p+2, y_p-3/2) = (x_p+2)^2 + (y_p- 3/2)^2 - R^2$$

$$\boldsymbol{d_{nuevo} = d_{viejo} + (2x_p - 2y_p + 5) \Rightarrow \Delta_{SE} = (2x_p - 2y_p + 5)}$$

Algoritmo del circulo de punto medio

Δ_E y Δ_{SE} varían en cada paso (en el caso lineal son ctes.).

Las operaciones para el cálculo de los d_{nuevo} son enteras.

Falta ver cómo comienza el algoritmo.

Si se parte del punto (0,*R*), con R entero

el siguiente punto medio es (1,*R*-½), o sea:

$F(1,R-½)=1+(R^2-R+¼)-R^2= {}^5/_4 - R = d_{viejo}$

Solución: pasar a $h = d - ¼ \Rightarrow h_{viejo} = 1-R$

Esto funciona porque h comienza y continúa con valores enteros

```
void MidPointCircle(int r, int color){
    int x, int y;
    float d;

    /* Valores iniciais */
    x = 0;
    y = r;
    d = 5/4 - r;

    CirclePoints(x, y, color);
    while (y > x){
        if (d < 0){
            /* Selecione E */
            d = d + 2 * x + 3;
            x++;
        }else{
            /* Selecione SE */
            d = d + 2 * (x - y) + 5;
            x++;
            y--;
        }/*end if*/
        CirclePoints(x, y, color);
    }/* end while */

}/* end MidpointCircle */
```

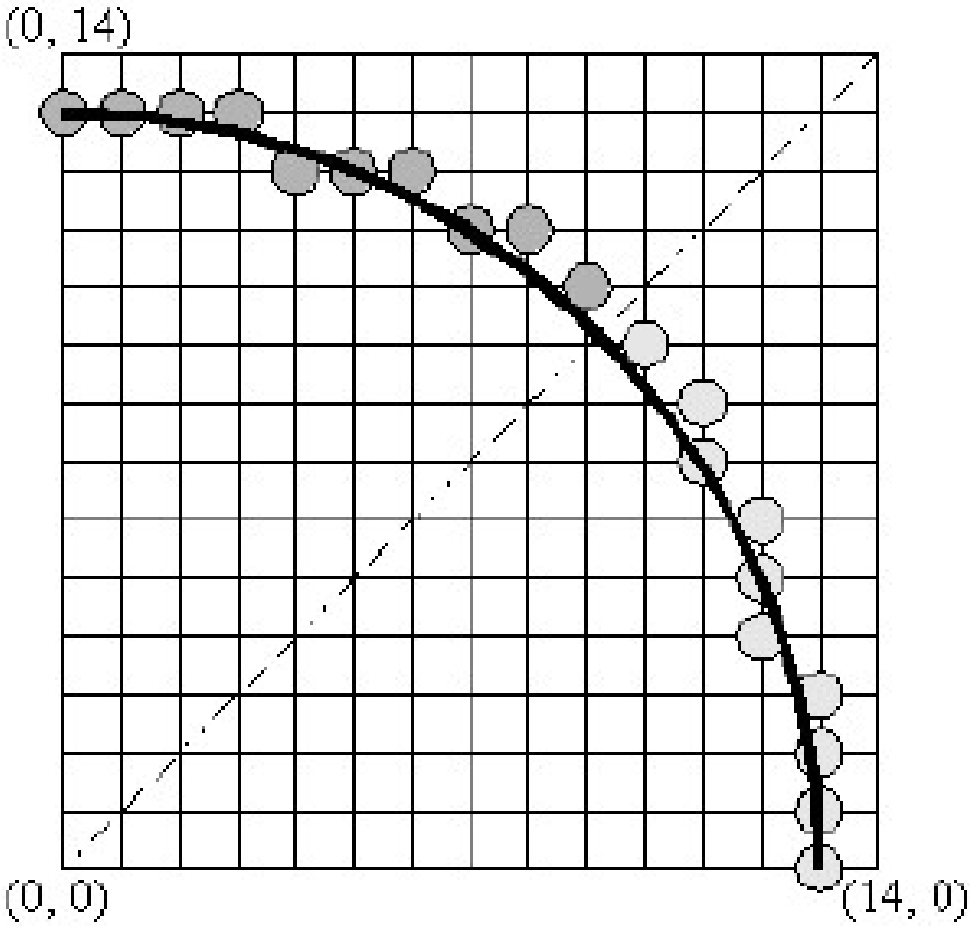
```
void MidPointCircleInt(int r, int color){
   int x, int y, d;

   /* Valores iniciais */
   x = 0;
   y = r;
   d = 1 - r;

   CirclePoints(x, y, color);
   while (y > x){
      if (d < 0){
         /* Selecione E */
         d = d + 2 * x + 3;
         x++;
      }else{
         /* Selecione SE */
         d = d + 2 * (x - y) + 5;
         x++;
         y--;
      }/*end if*/
      CirclePoints(x, y, color);
   }/* end while */

}/* end MidpointCircleInt */
```

Algoritmo del circulo de punto medio



Diferencias de segundo orden

Si escogemos E el punto de evaluacion es (x_p+1 , y_p)

en la diferencia de orden 1 tenemos:

$\Delta_{E(old)}$ *en* $(x_p, y_p) = 2x_p + 3$

En $\Delta_{E(new)}$ *en* $(x_p+1, y_p) = 2(x_p+1) + 3$

en la diferencia de orden 2 tenemos:

$\Delta_{E(new)} - \Delta_{E(old)} = 2$

Analogamente para $\Delta_{SE(old)}$ *en* $(x_p, y_p) = 2x_p - 2y_p + 5$

En $\Delta_{E(new)}$ *en* $(x_p+1, y_p) = 2(x_p+1) - 2(y_p) + 5$

en la diferencia de orden 2 tenemos:

$\Delta_{E(new)} - \Delta_{E(old)} = 2$

Diferencias de segundo orden

Si escogemos SE el punto de evaluacion es (x_p+1, y_p-1)

en la diferencia de orden 1 tenemos:

$\Delta_{E(old)}$ *en* $(x_p, y_p) = 2x_p + 3$

En $\Delta_{E(new)}$ *en* $(x_p+1, y_p) = 2(x_p+1) + 3$

en la diferencia de orden 2 tenemos:

$\Delta_{E(new)} - \Delta_{E(old)} = 2$

Analogamente para $\Delta_{SE(old)}$ *en* $(x_p, y_p) = 2x_p - 2y_p + 5$

En $\Delta_{E(new)}$ *en* $(x_p+1, y_p) = 2(x_p+1) - 2(y_p - 1) + 5$

en la diferencia de orden 2 tenemos:

$\Delta_{E(new)} - \Delta_{E(old)} = 4$

```
void MidPointCircleInt(int r, int color){
   // Função de utiliza diferenças parciais
   // decisão d. Circunferência centrada na

   /* Valores iniciais */
   int x = 0;
   int y = r;
   int d = 1 - r;
   int deltaE=3
   inte deltaSE= -2*r+5

   CirclePoints(x, y, color);
   while (y > x){
      if (d < 0){ /* Selecione E */
         d +=deltaE;
         deltaE += 2;
         deltaSE += 2;
      }else{  /* Selecione SE */
         d +=deltaSE;
         deltaE += 2;
         deltaSE += 4;
         y--;
      }/*end if*/
      x++;
      CirclePoints(x, y, color);
   }/* end while */

}/* end MidpointCircleInt */
```

Conversión matricial de elipses

Ecuación del círculo centrado en el origen:

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

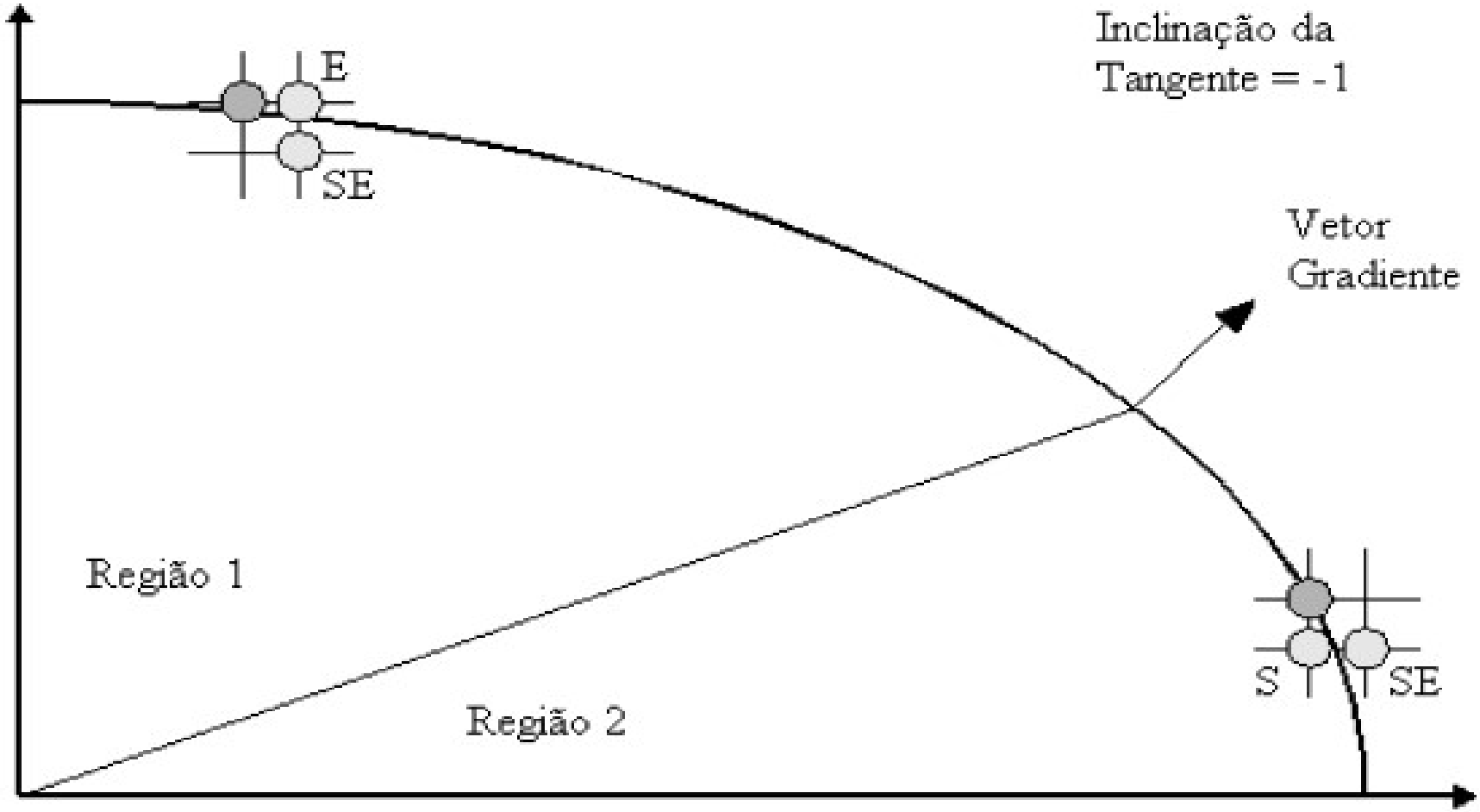
Con 2a como el eje mayor

Y 2b la media del eje menor

Para graficar una elipse el primer cuadrante se dividira en dos regiones, en el punto de curva cuya tanjente tiene inclinacion –1 el cual estara dado por la gradiente

$$GradienteF(x,y) = \frac{\partial F}{\partial x}\vec{i} + \frac{\partial F}{\partial x}\vec{j} = 2b^2x\vec{i} + 2a^2y\vec{j}$$

Conversión matricial de elipses



Conversión matricial de elipses

- Tendrá que ser usado la variable d para saber si el punto evaluado esta dentro de la elipse o no.
- Repitiendo el proceso de derivación igual que en la circunferencia tenemos que en la región 1 tenemos la variable d1, y el punto medio es $F(x_p+1, y_p- 1/2)$:

  $\Delta_E = b^2(2x_p+3)$

  $\Delta_{SE} = b^2(2x_p+3) + a^2(-2y_p-1)$
- En la región 2, tenemos d_2 y el punto medio es $F(x_p+1/2, y_p-1)$
- El primer punto medio será en (0,b) y el primer punto medio será (1, b-1/2):

$$F(1, b - \frac{1}{2}) = b^2 + a^2(b - \frac{1}{2})^2 - a^2b^2 = b^2 + a^2(-b + \frac{1}{4})$$

```
void MidpointElipse(int a, int b, int color){
   int x, int y;
   float d1, d2;

   /* Valores iniciais */
   x = 0;
   y = b;
   d1 = b * b - a * a * b + a * a / 4.0;

   ElipsePoints(x, y, color); /* Simetria de ordem 4 */
   while(a * a * (y - 0.5) > b * b * (x + 1)){
      /* Região 1 */
      if (d1 < 0)
         d1 = d1 + b * b * (2 * x + 3);
         x++
      }else{
         d1 = d1 + b * b * (2 * x + 3) + a * a * (-2 * y + 2);
         x++;
         y--;
      }/*end if*/
      ElipsePoints(x, y, color);
   }/* end while */
```

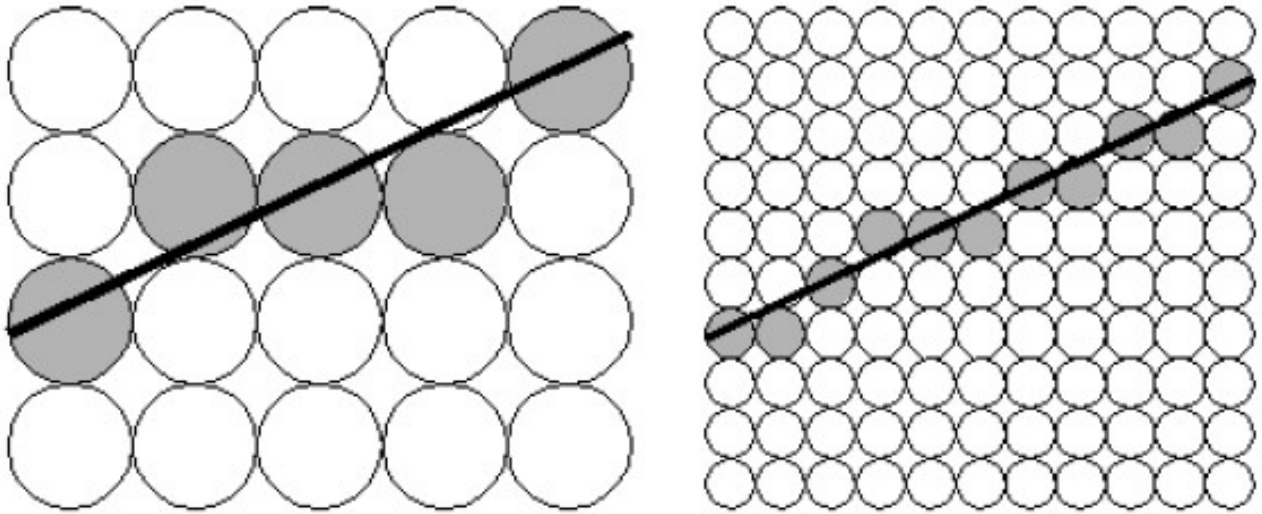
```
    d2 = b * b * (x + 0.5) * (x + 0.5) + a * a * (y - 1) * (y - 1) - a * a * b * b;
    while(y > 0){
       /* Região 2 */
       if (d2 < 0){
          d2 = d2 + b * b * (2 * x + 2) + a * a * (-2 * y + 3);
          x++;
          y--;
       }else{
          d2 = d2 + a * a * (-2 * y + 3);
          y--;
       }/*end if*/
       ElipsePoints(x, y, color);
    }/* end while */
}/*end MidpointElipse*/
```

Corrección en el trazado

- Los dispositivos gráficos generalmente no son cuadrados y necesita salvar esta necesidad de las coordenadas de usuario para las del dispositivo.
- Las diferencias de densidades representan el hecho de una escala no homogénea realizada por le dispositivo
- Otra cosa es eliminar el efecto de cerrillado, generalmente para la corrección de esos errores se usan técnicas de no solo pintar los píxeles calculados por el algoritmo de conversión matricial si no también los vecinos

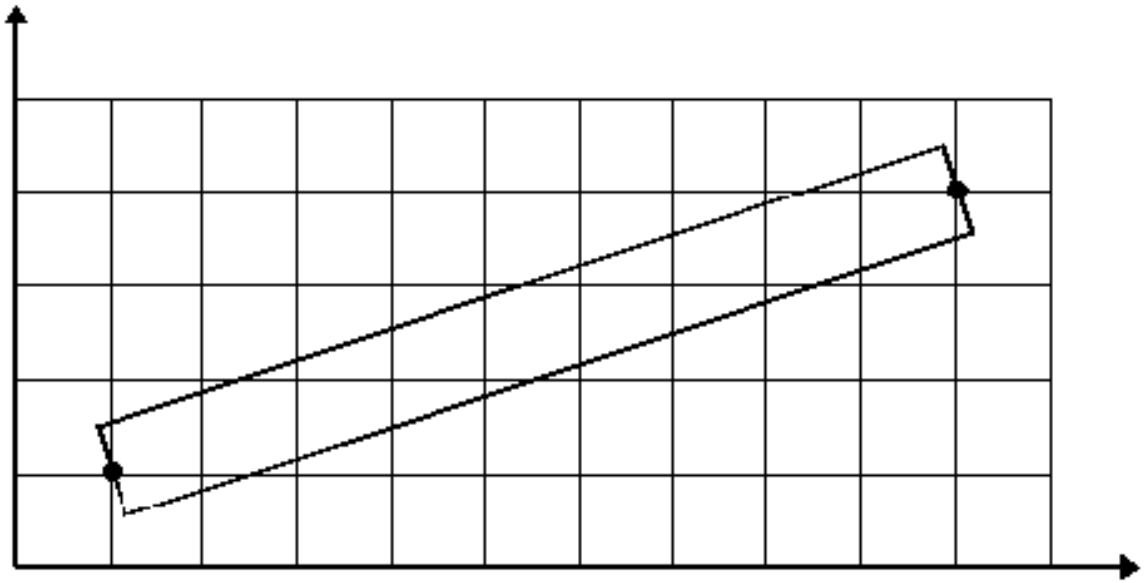
Antialiasing

- Una de las cosas mas simples para resolver este problema es aumentar la resolución del dispositivo



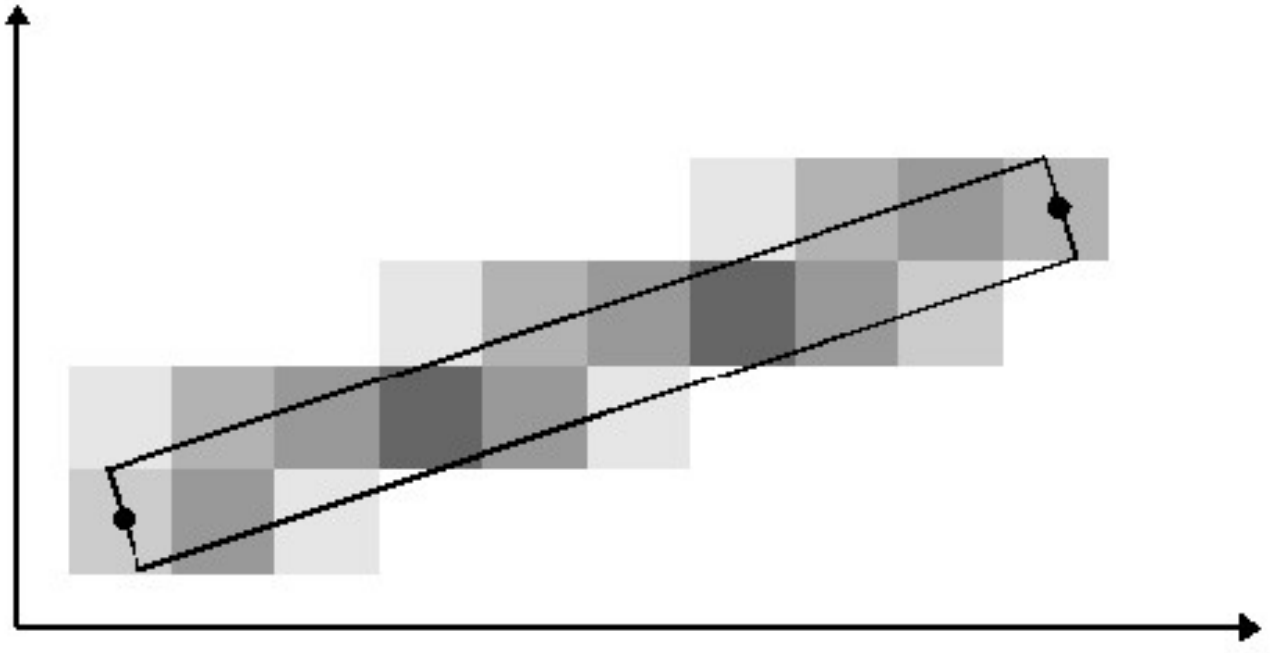
Corrección por áreas no ponderadas

- Un recta es considera como si fuese un rectángulo



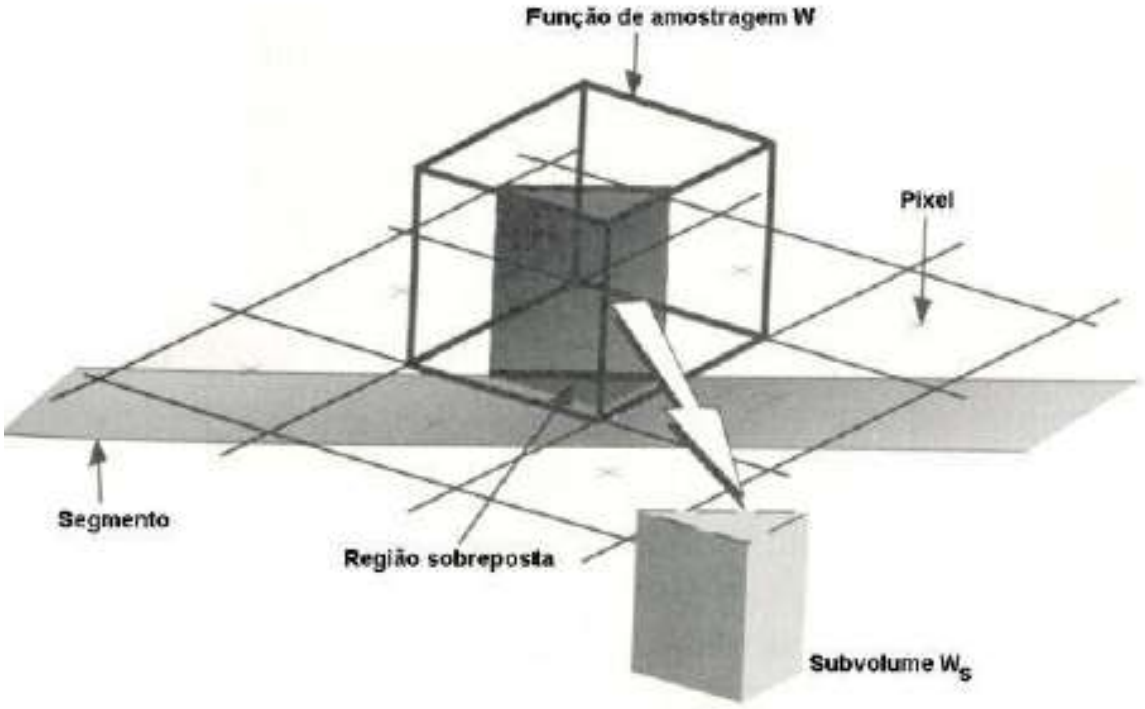
Corrección por áreas no ponderadas

- Un recta es considera como si fuese un rectángulo



Corrección por áreas ponderadas

- Un recta es considera como si fuese un rectángulo



Corrección por áreas ponderadas