COMPUTACION VISUAL

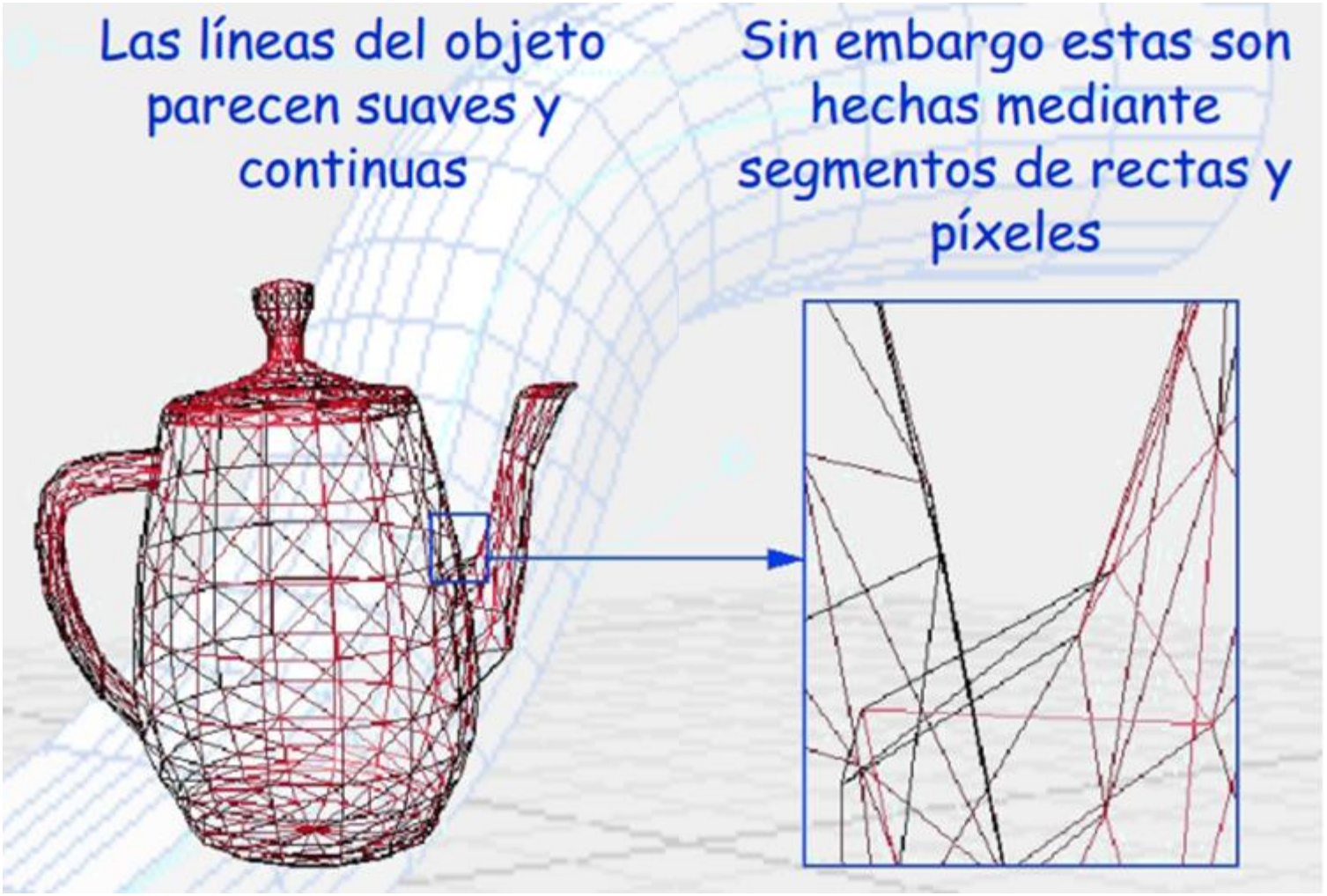
CONSTRUCCIÓN DE PRIMITIVAS GRAFICAS

Prof. John Ledgard Trujillo Trejo

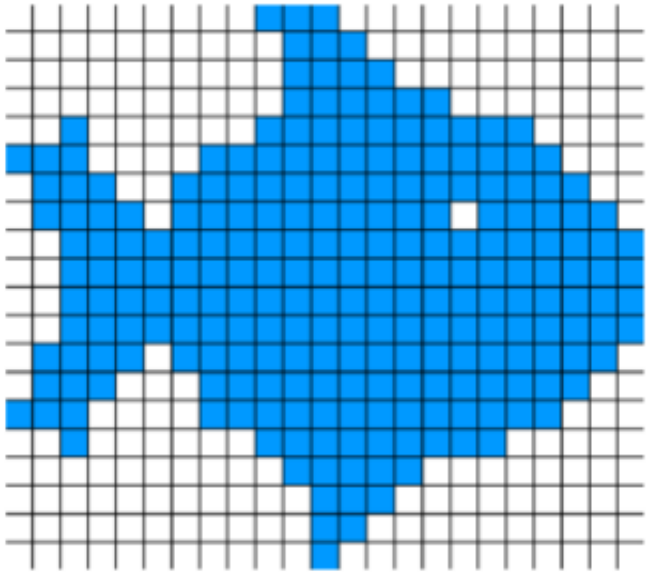
Facultad de Ingeniería de Ingeniería de Sistemas
Departamento de Ciencias de la Computación
UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Motivación

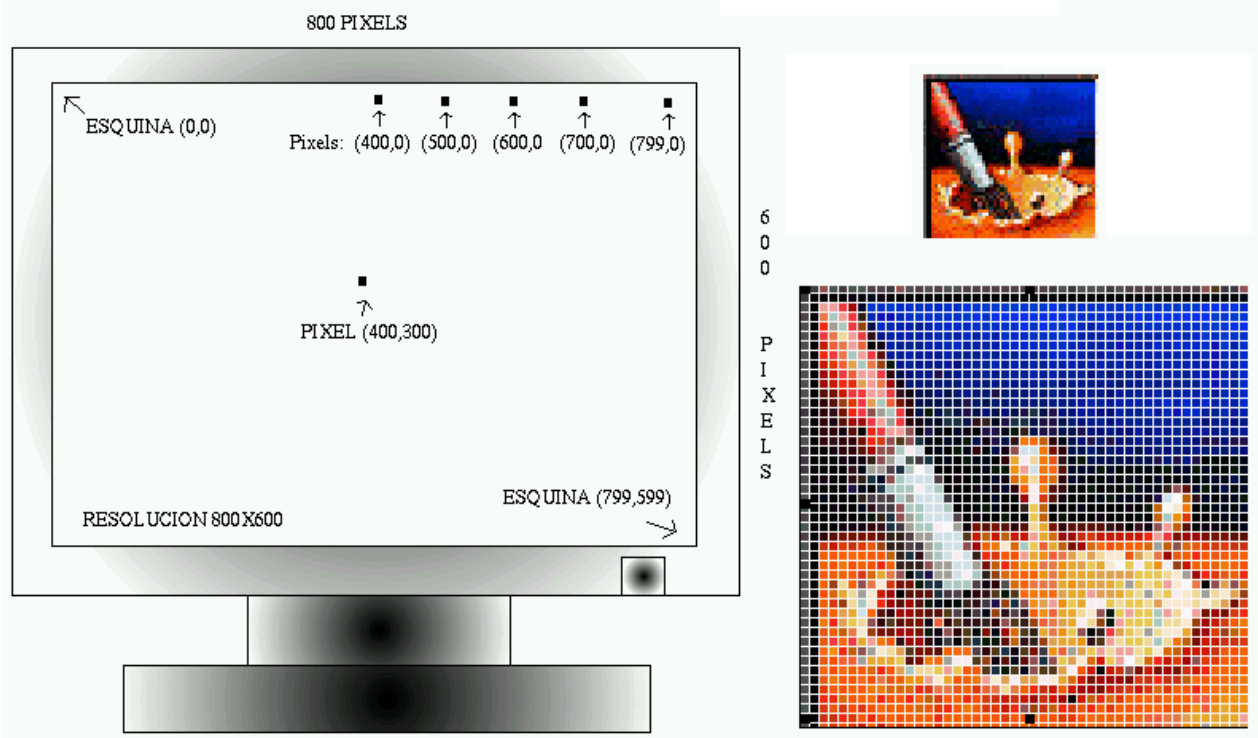
Motivación


Las líneas del objeto parecen suaves y continuas
Sin embargo estas son hechas mediante segmentos de rectas y píxeles

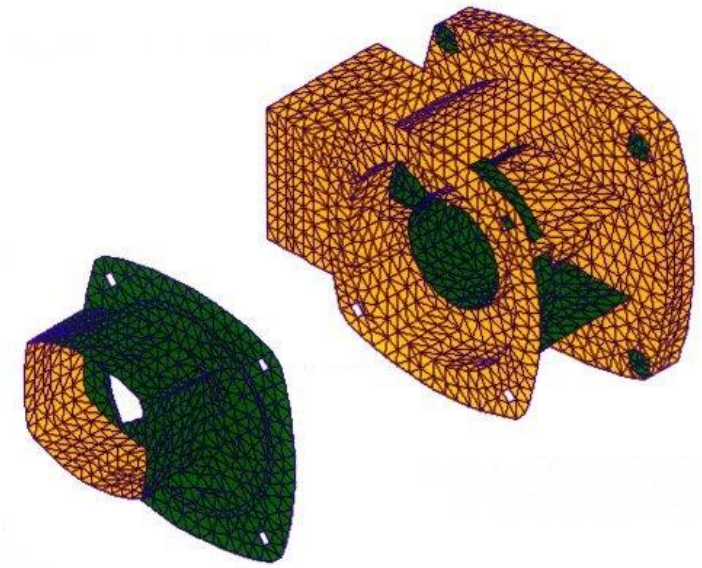
- **Representar figuras => iluminar píxeles apropiadamente**



- Problema:
  - Los dispositivos raster son discretos y los objetos continuos
  - El dispositivo de salida es limitado

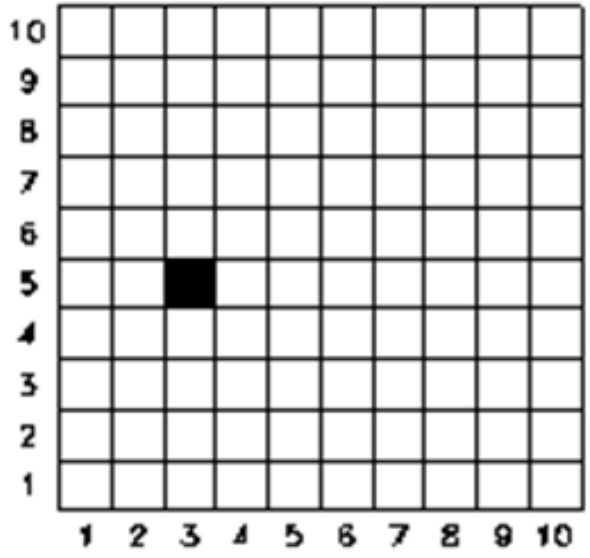


- ¿Cómo convertir figuras continuas en su representación discreta?
- Muestreo o rasterización, filtrado y recortado =>
  - Perdida de información
  - Transformación de información.



Un pixel (de las palabras inglesas picture element) es el elemento direccionable más pequeño de la pantalla.

- **Es la porción más pequeña que podemos controlar.**
- **Todos los pixel tienen un nombre o una dirección.**
- **Los nombres que identifican a los pixel corresponden a las coordenadas que definen a los puntos. Esto es parecido a la forma de seleccionar los elementos de una matriz mediante subíndices.**
- **Las imágenes gráficas creadas por ordenador se obtienen activando la intensidad y el color de los pixel que componen la pantalla.**
- **Se dibujan segmentos dando brillo a un conjunto de pixel situados entre el pixel inicial y el pixel final.**



- Trazado de Rectas:
  1. Algoritmo Básico
  2. Algoritmo DDA (Digital Differential Analizer).
  3. Algoritmo de punto medio. Criterio del punto medio.

  Bresenham, J.E. Algorithm for computer control of a digital plotter, IBM Systems Journal, January 1965, pp. 25-30.
- Trazado de Circunferencias:
  Algoritmo basado en el punto medio.

  Bresenham, J.E. A linear algorithm for incremental digital display of circular arcs Communications of the ACM, Vol. 20, pp. 100-106, 1977.
- Trazado de Elipses y otras cónicas:
  Algoritmos basado en el punto medio.

J.E. Breshenham

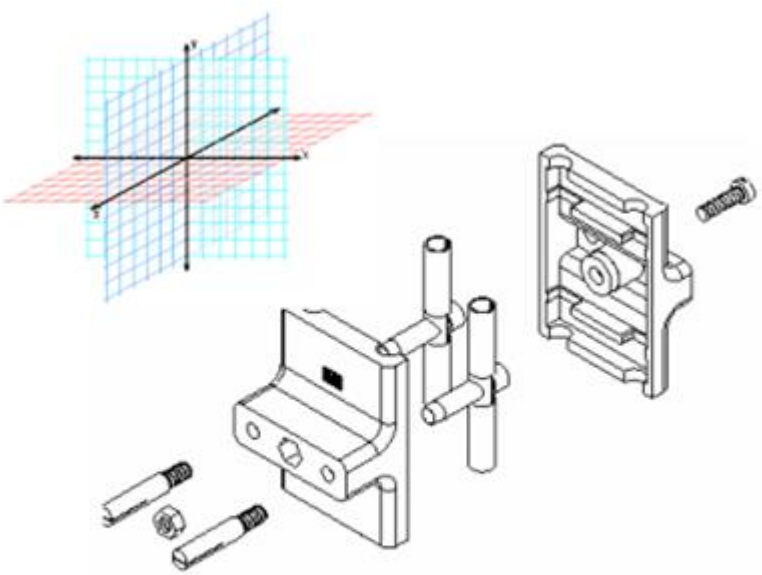
Primitivas Gráficas: Punto

En la "imagen" mental del usuario un punto normalmente se representan en un espacio Euclidiano de una determinada dimensión. En dichas condiciones un punto es una entidad matemática $p = (x, y)$, *donde* $(x, y) \in R^2$.

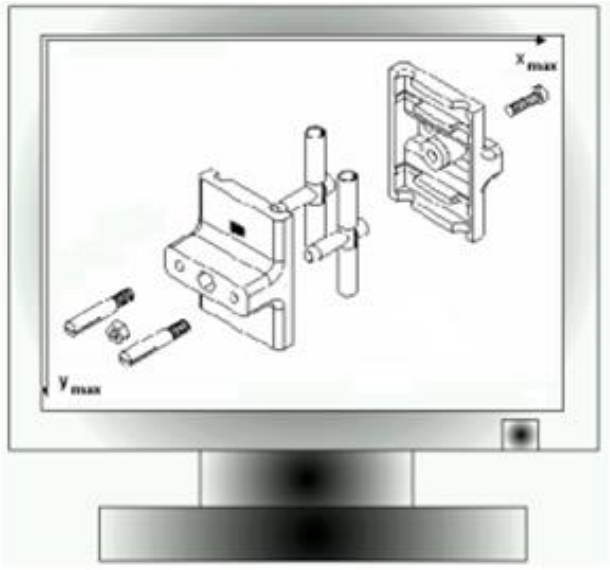
- En el soporte aritmético de la computadora, dicha representación se efectúa con los tipos de datos provistos, que pueden ser números reales con punto flotante de simple o doble precisión (espacio de escena).

- En el soporte gráfico del bufer de pantalla, un punto se representa con un pixel, y dicha representación se efectúa seteando una posición de memoria con un contenido dado (espacio pantalla).

Es la operación de llevar una primitiva del espacio Euclideano (espacio de la escena) al espacio de pantalla.



Espacio Euclideano
(Coordenadas Físicas)



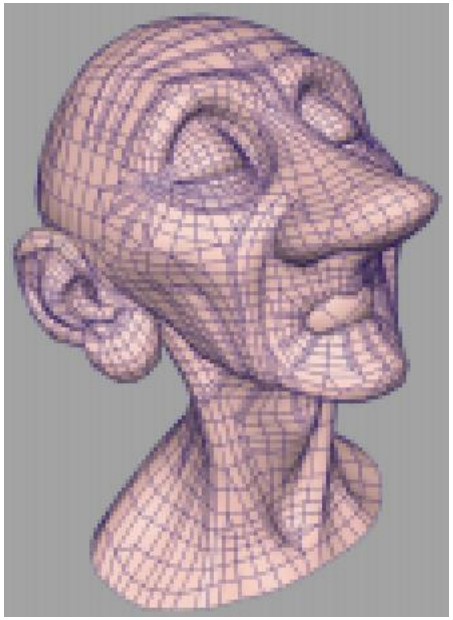
Espacio de pantalla
(Coordenadas de dispositivo)

Discretización

Proceso que aproxima las diferentes figuras geométricas con un conjunto de pixels (del ingles Scan-conversion o rasterización).

El problema principal a tratar se basa en el estudio de primitivas gráficas para la conversión de definiciones geométricas continuas a un esquema discreto de píxeles, y la aproximación necesaria para poder llevar a cabo la transformación.

Discretización



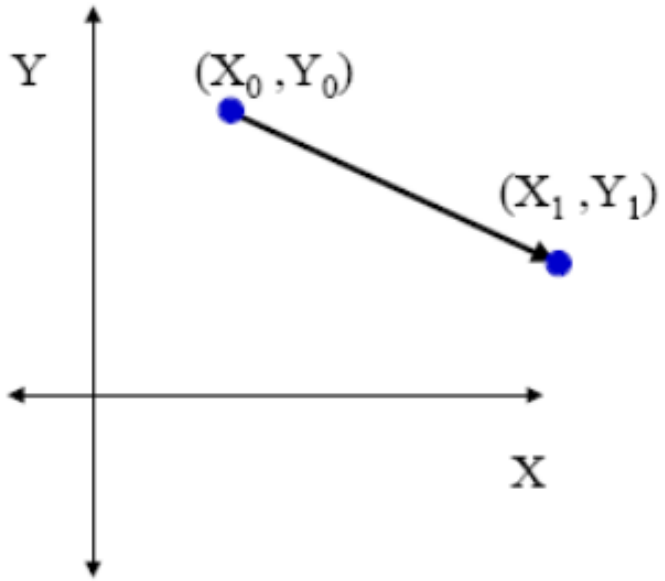
Especificaciones de una discretización:

- **Apariencia:**
  - Evitar discontinuidad o puntos espúreos
  - Debe ser uniforme
- **Simetría e invarianza geométrica:**
  - Producir resultados equivalentes si se modifican algunas propiedades geométricas de la primitiva.
- **Simplicidad y Velocidad de Computo:**
  - Evitar el uso de operaciones aritméticas complejas.
  - Evitar los errores de las operaciones aritméticas

Cuestiones en dibujar líneas y curvas

- Aproximando una curva continua con un número finito de puntos en posiciones fijas.
- Haciendo el algoritmo tan rápido como sea posible.

El problema de dibujar una línea

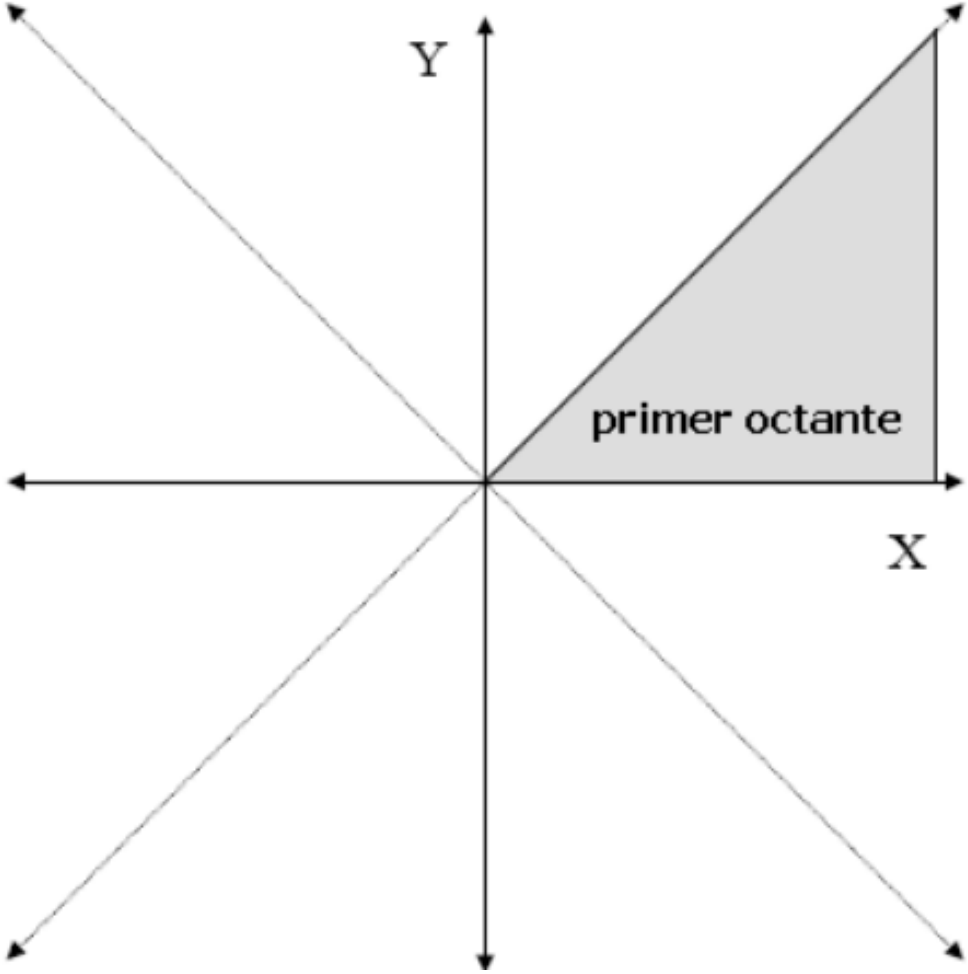


Reduciendo Muchos Problemas a Uno

- Comience con cualquier segmento dirigido de la línea.
- Transformar en un segmento del origen en el primer octante.
- Llevar la cuenta de las transformaciones usadas.
- Dibujar la línea en el primer octante.
- Pero ejerza transformaciones en orden inverso antes de sacar cada punto.

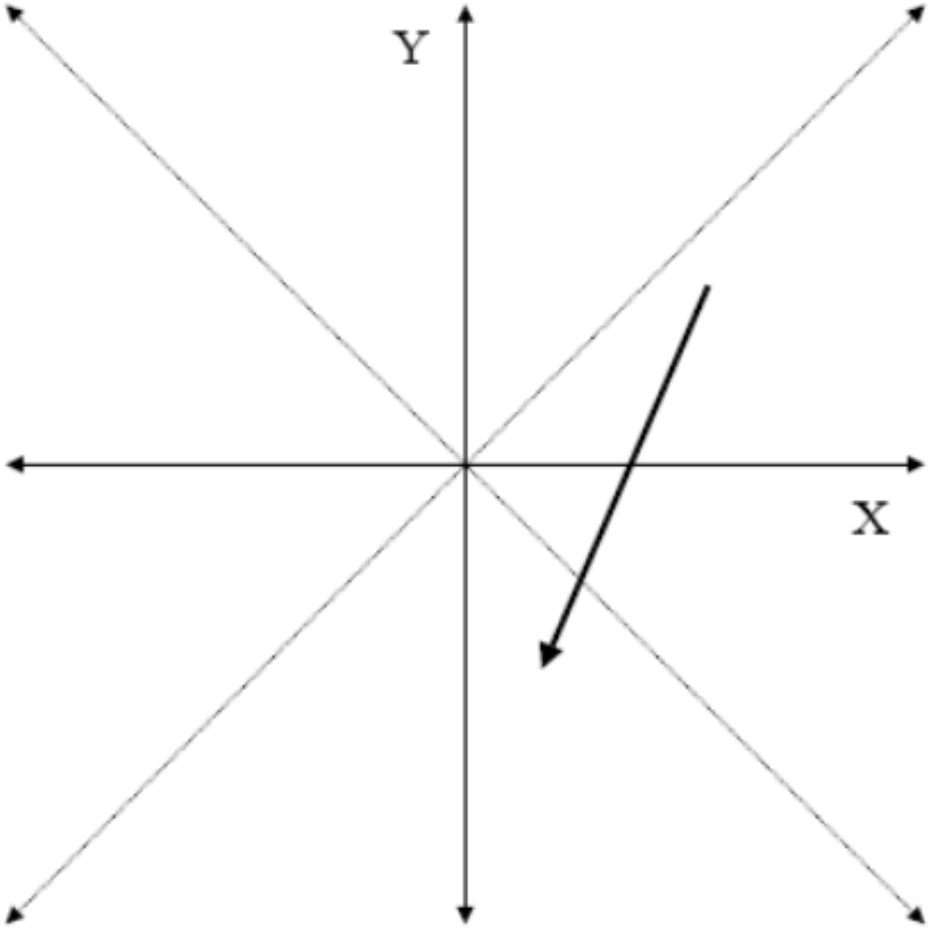
Reduciendo Muchos Problemas a Uno

Dividir el plano en ocho octantes



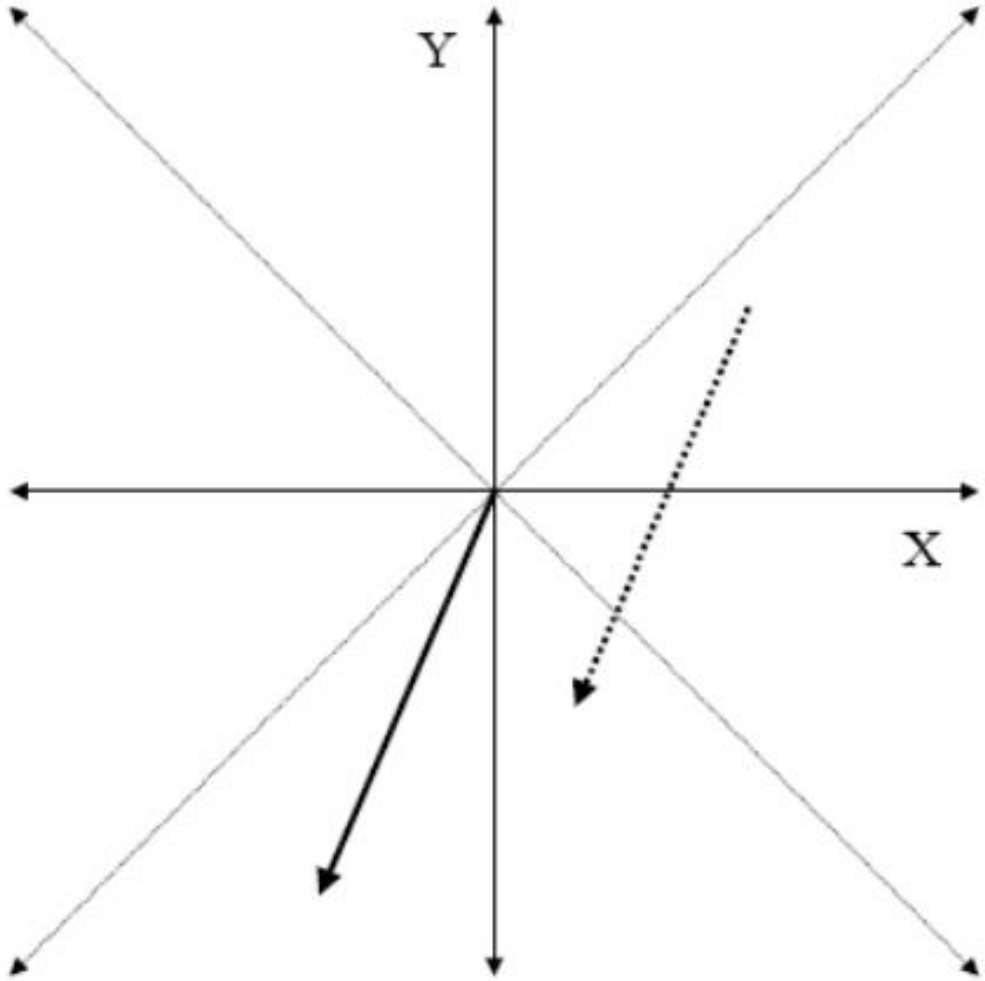
Reduciendo Muchos Problemas a Uno

Comience con cualquier segmento dirigido de la línea.



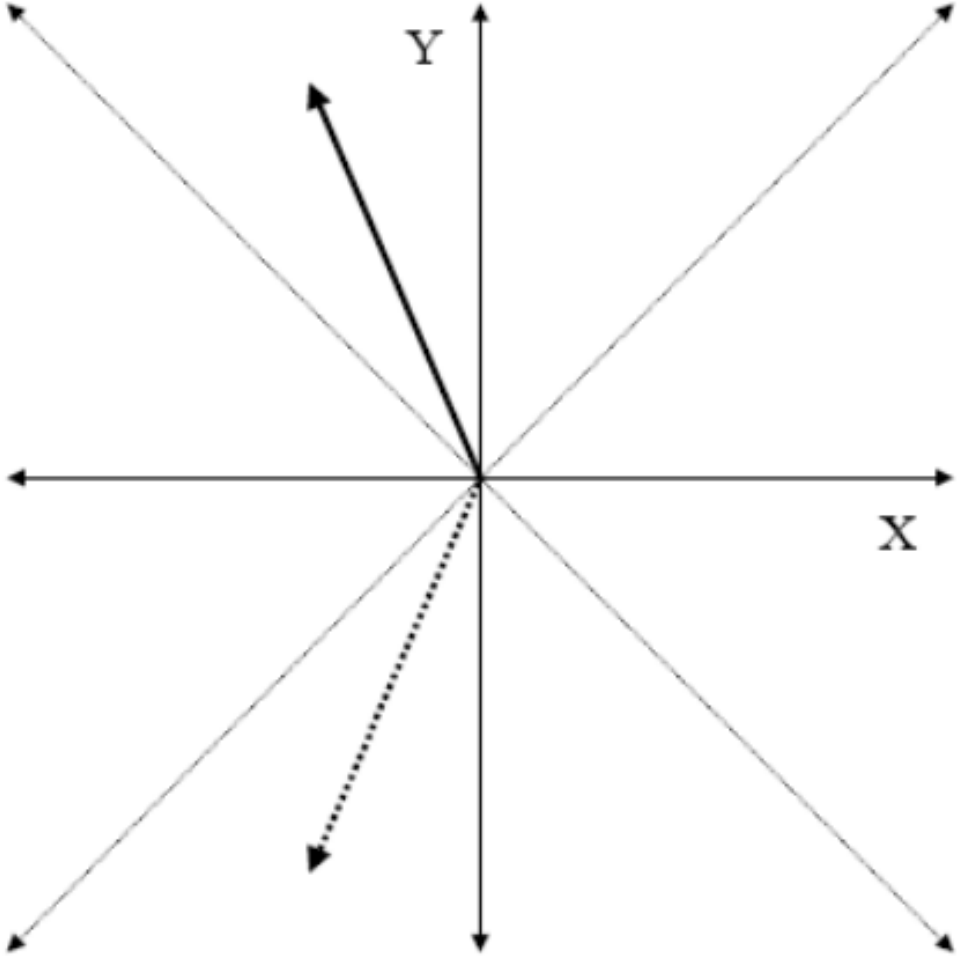
Reduciendo Muchos Problemas a Uno

trasladarlo al origen



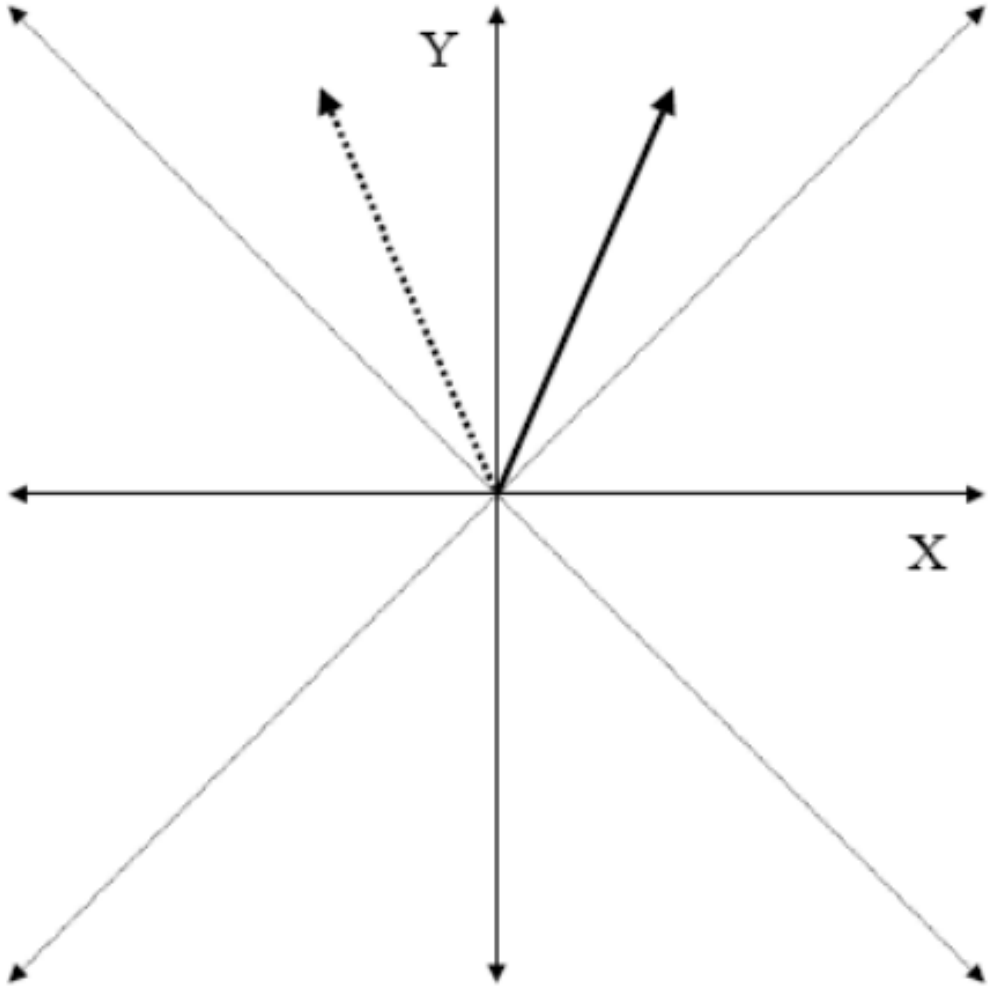
Reduciendo Muchos Problemas a Uno

Reflexión respecto al eje X

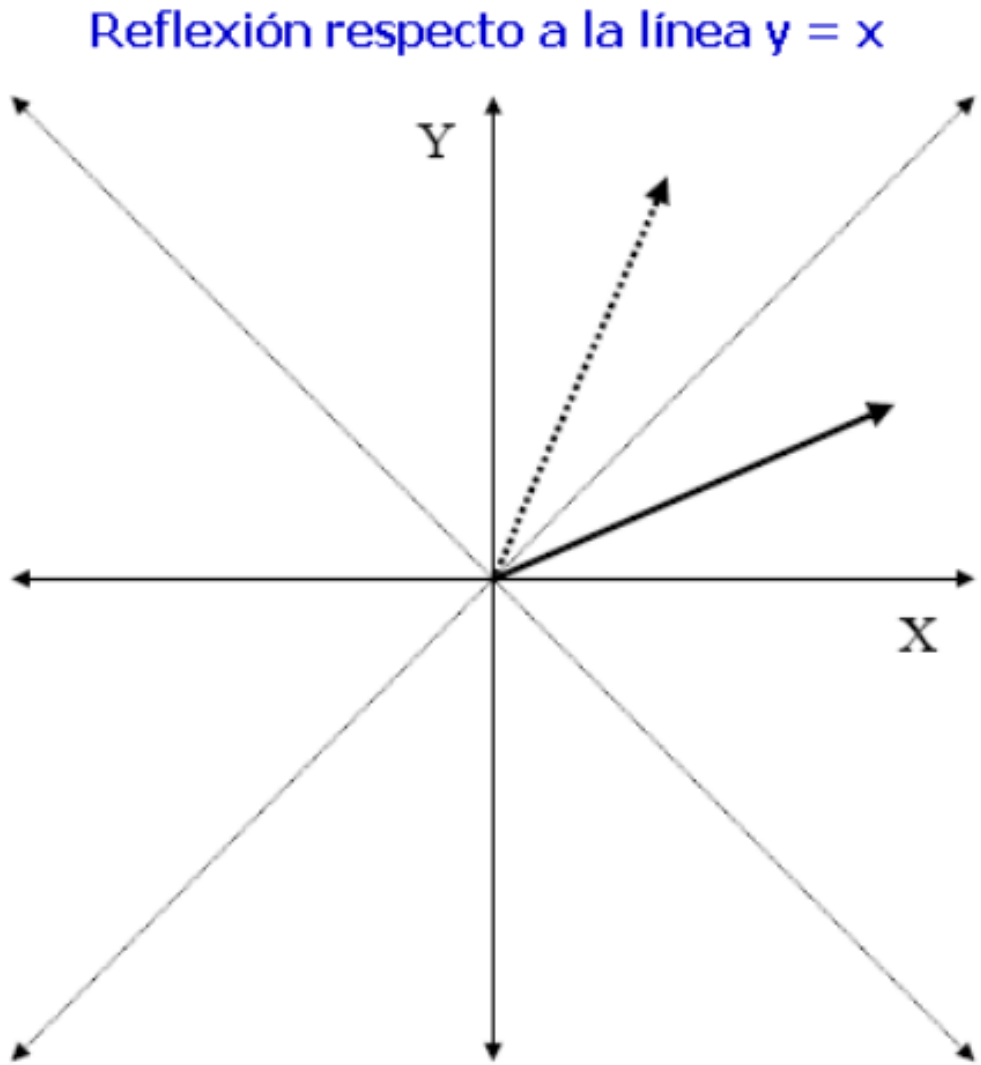


Reduciendo Muchos Problemas a Uno

Reflexión respecto al eje Y



Reduciendo Muchos Problemas a Uno



Discretización de Líneas (Segmento de recta)

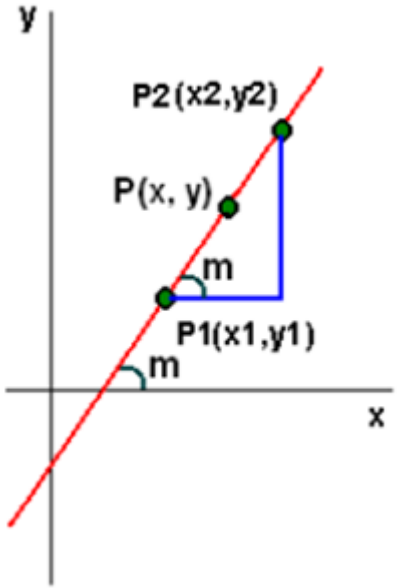
La recta es la línea más corta que une dos puntos y el lugar geométrico de los puntos del plano (o el espacio) en una misma dirección.

Si conocemos dos puntos de la recta, P1(x1,y1) y P2(x2, y2), la pendiente es:

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

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

$$y - y_1 = m(x - x_1)$$



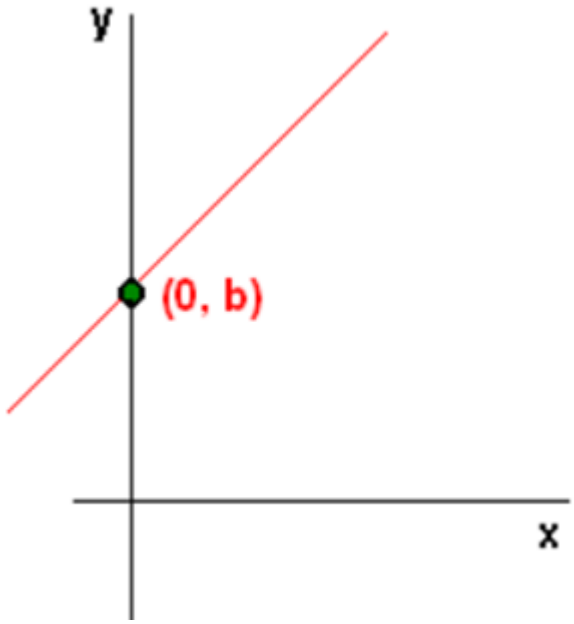
Discretización de Líneas (Segmento de recta)

$$y = m x + y_1 - m x_1$$

$$y = m x + b$$ Ecuación de la recta punto pendiente

La ecuación general de la recta es de la siguiente forma:

$$Ax + By + C = 0$$



Discretización de Líneas (Segmento de recta)

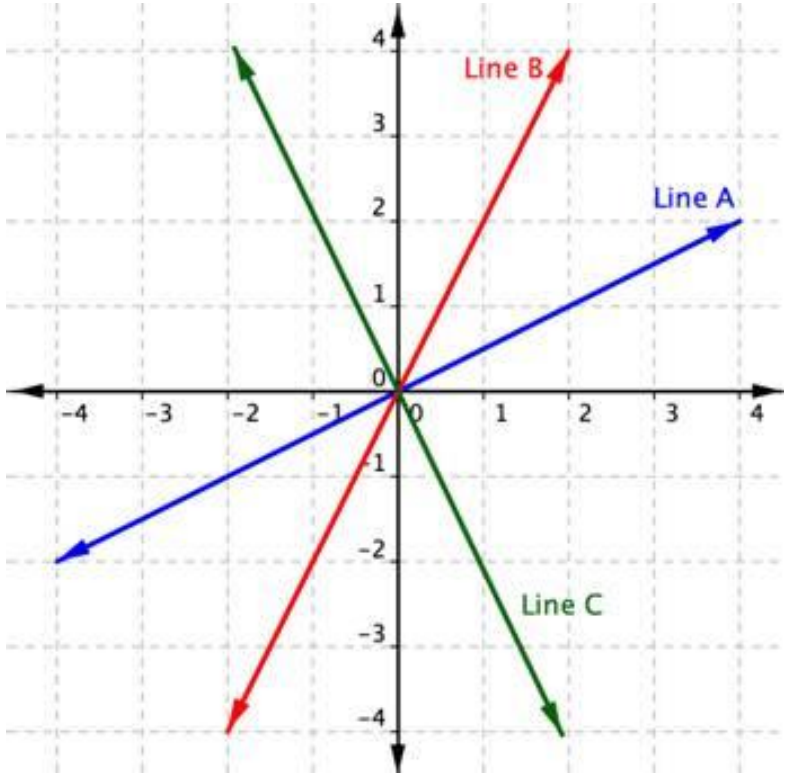
y = mx + b Ecuación de la recta punto pendiente

donde **m = Δy / Δx** es la pendiente de la recta

$b = -mx_0 + y_0$

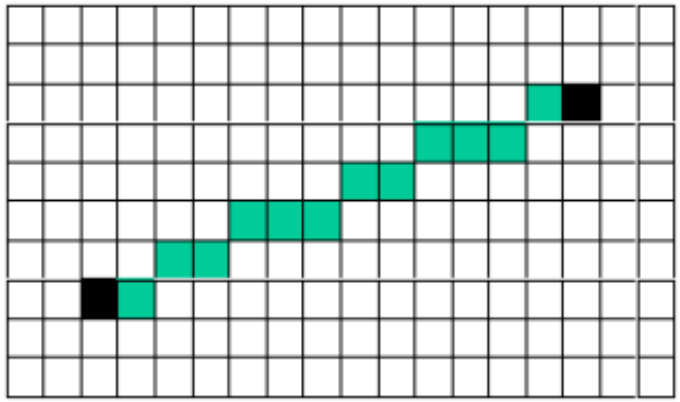
- **Para |m| < 1, la línea es mas horizontal que vertical.**
  Si m → 0 la línea tiende a ser horizontal

- **Para |m| > 1, la línea es mas vertical que horizontal.**
  Si m → ∞ la línea tiende a ser vertical



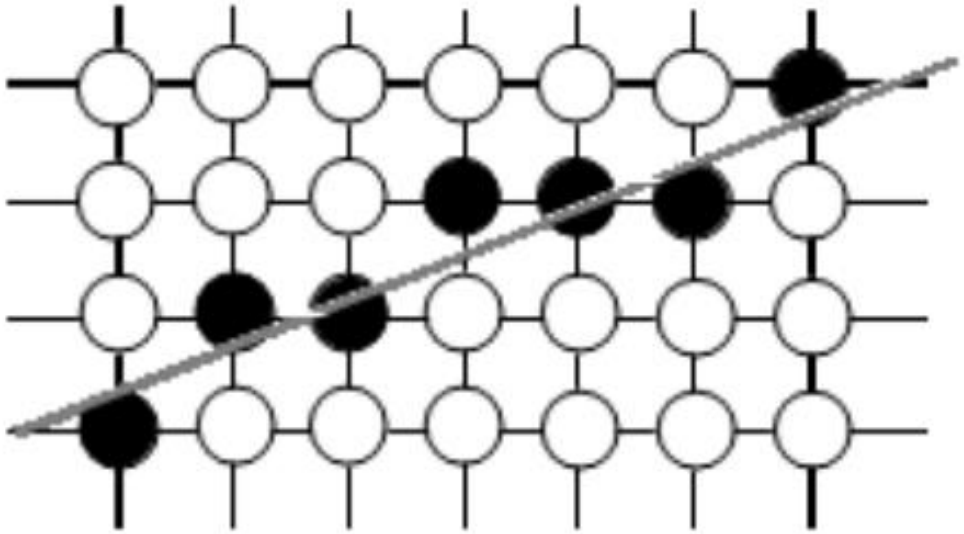
Discretización de Líneas (Segmento de recta)

- Para dibujar líneas rectas, hay que calcular las posiciones intermedias entre los dos extremos
- Este problema no existía en las pantallas vectoriales o plotters
- Las posiciones de los pixels son valores enteros, y los puntos obtenidos de la ecuación son reales, entonces existe un error (aliasing)
- A menor resolución, mayor es el efecto



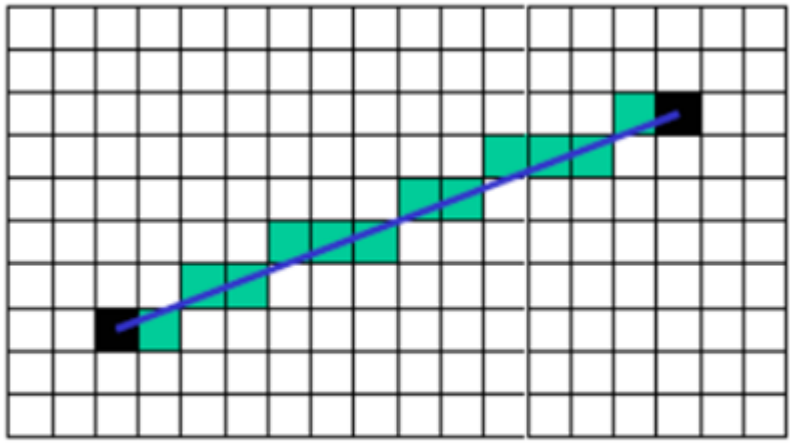
Discretización de Líneas (Segmento de recta)

- Hay que calcular las coordenadas de los pixels que estén lo más cerca posible de una línea recta ideal, infinitamente delgada, superpuesta sobre la matriz de pixels.

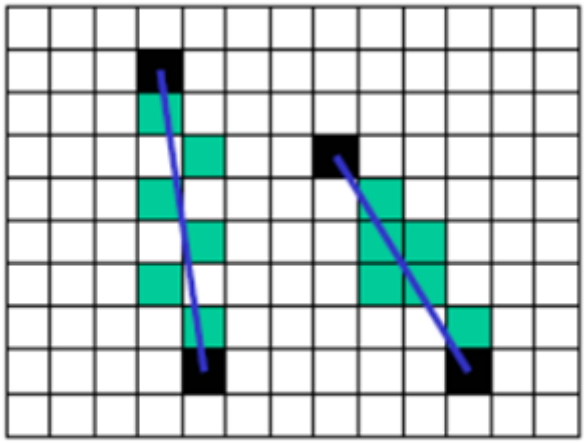


Discretización de Líneas (Segmento de recta)

- **Consideraciones para la discretización de líneas**
  - La secuencia de pixels debe ser lo más recta posible.
  - Las líneas deben dibujarse con el mismo grosor e intensidad independiente de su inclinación



correcto



incorrecto

Discretización de Líneas (Segmento de recta)

❑ Algoritmo de recta simple

La ecuación de una recta es $y = mx + b$

Donde m es la pendiente de la recta

b es el corte con el eje y

$$m = \frac{\Delta y}{\Delta x} = \frac{y_1 - y_0}{x_1 - x_0}$$

$$b = y_0 - mx_0$$

Para un valor determinado x_i

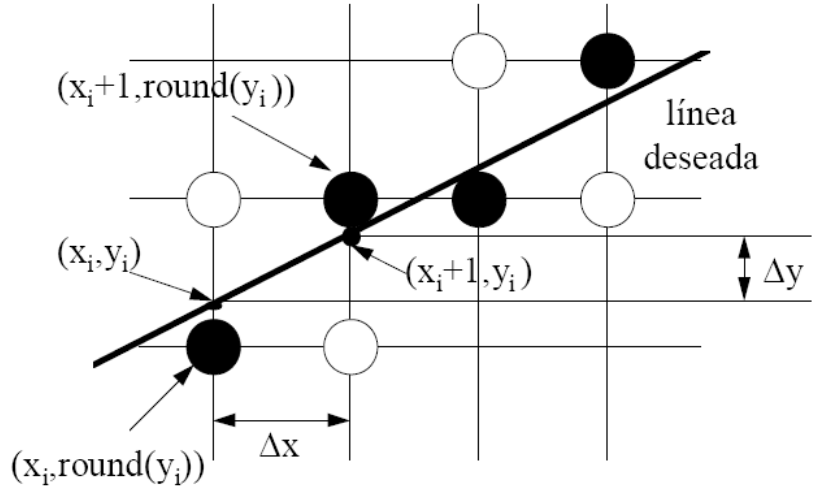
obtenemos $y_i = m\,x_i + b$

pintamos el pixel $(x_i\ ,\ \text{round}(y_i))$

Discretización de Líneas (Segmento de recta)

❑ Algoritmo de recta simple

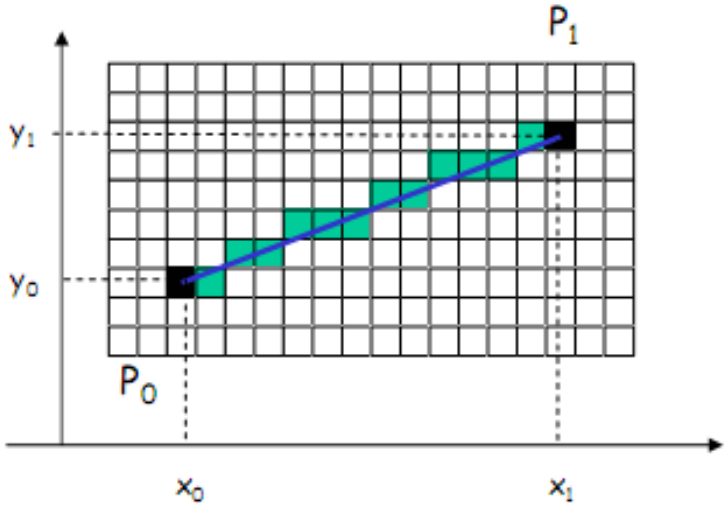
```
void recta_simple (int x0, int y0, int x1, int y1)
{
  int x, y;
  float dx, dy, m;
  dx = x1 - x0;
  dy = y1 - y0;
  m = dy/dx;
  b = y0 - m*x0;
  y = y0;
  for (x=x0; x<=x1; x++)
  {
        pintar(x, round(y), atributo);
        y = m*x + b;
  }
}
```

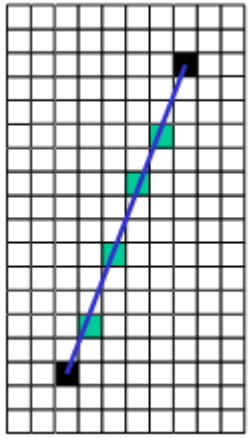


Discretización de Líneas (Segmento de recta)

Algoritmo de recta simple

- No es muy eficiente
- Cada paso requiere una multiplicación flotante, una suma y un redondeo.
- Acumulación de errores
  - Producto
  - Suma
  - Redondeo
- Es valido si la coordenada del primer extremo de la recta es menor que la del segundo extremo: $x_0 < x_1$
- Solo funciona bien con pendientes pequeñas, con pendientes mas altas deja huecos indeseables.

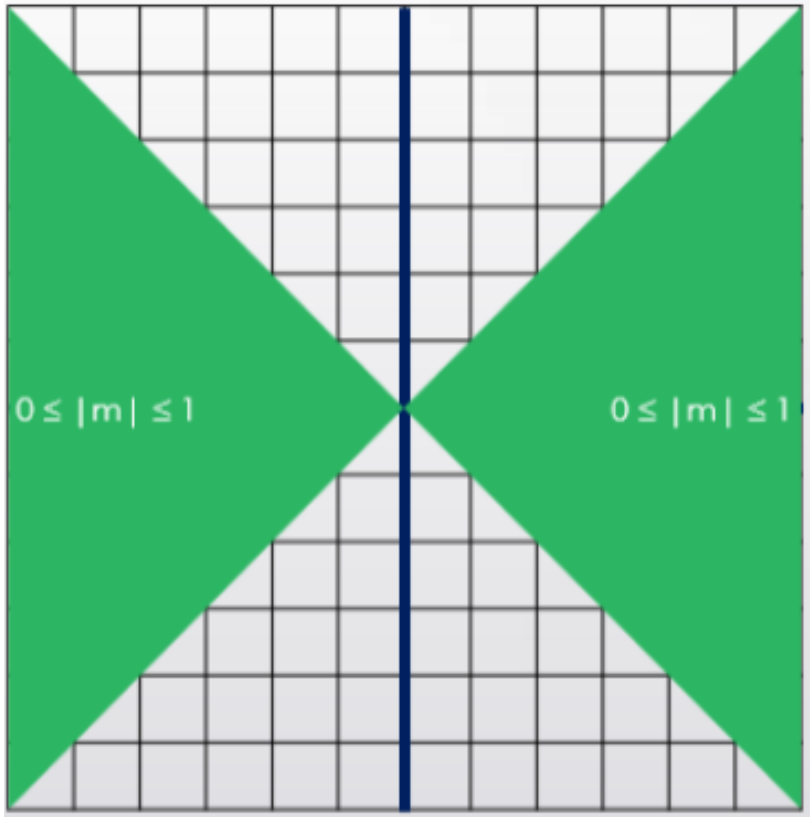




Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)

Caso 1: $0 \leq | m | \leq 1$



Ecuación de la recta:

y=mx+b

Donde

- b es la intersección con el eje y
- m es la pendiente

$$m = \frac{Y_2 - Y_1}{X_2 \_ X_1}$$

Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)

Caso 1: $m < 1$

Sea $x_i \Rightarrow y_i = m\,x_i + b$

Para x_{i+1} tambien $y_{i+1} = m\,x_{i+1} + b$

Sabemos que $\Delta x = x_{i+1} - x_i$

$$\Rightarrow \quad x_{i+1} = x_i + \Delta x_i$$

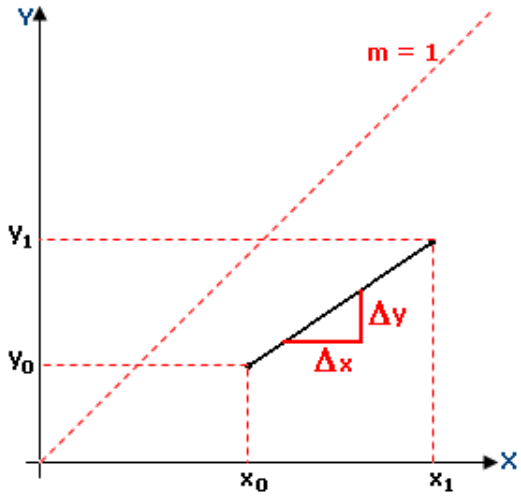
Reemplazando $y_{i+1} = m(x_i + \Delta x_i) + b$

$$y_{i+1} = mx_i + b + m\Delta x_i$$

$$y_{i+1} = y_i + m\Delta x_i$$

Haciendo que $\Delta x = 1 \Rightarrow x_{i+1} = x_i + 1$

y $y_{i+1} = y_i + m$



En general:

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i + m$ **tal que** $m = \Delta y / \Delta x$

Pintar $(x_{i+1}$, round$(y_{i+1}))$

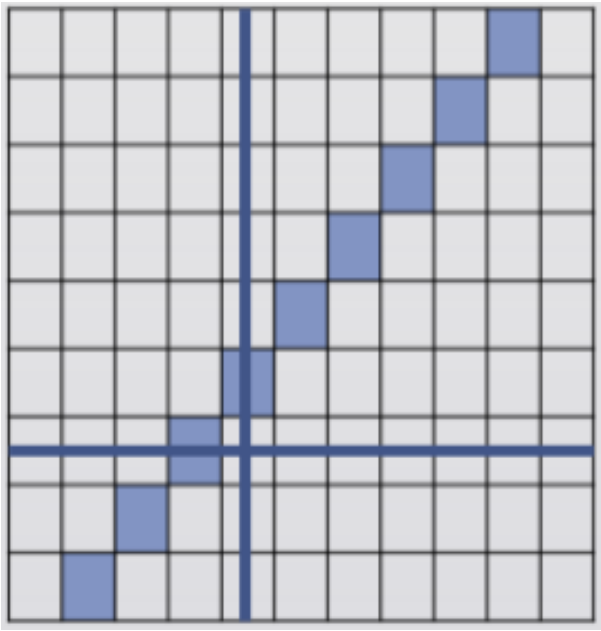
Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)

(-3,-2) → (5,6)

m=1. ⟹ $y_{i+1}=y_{i+1}$

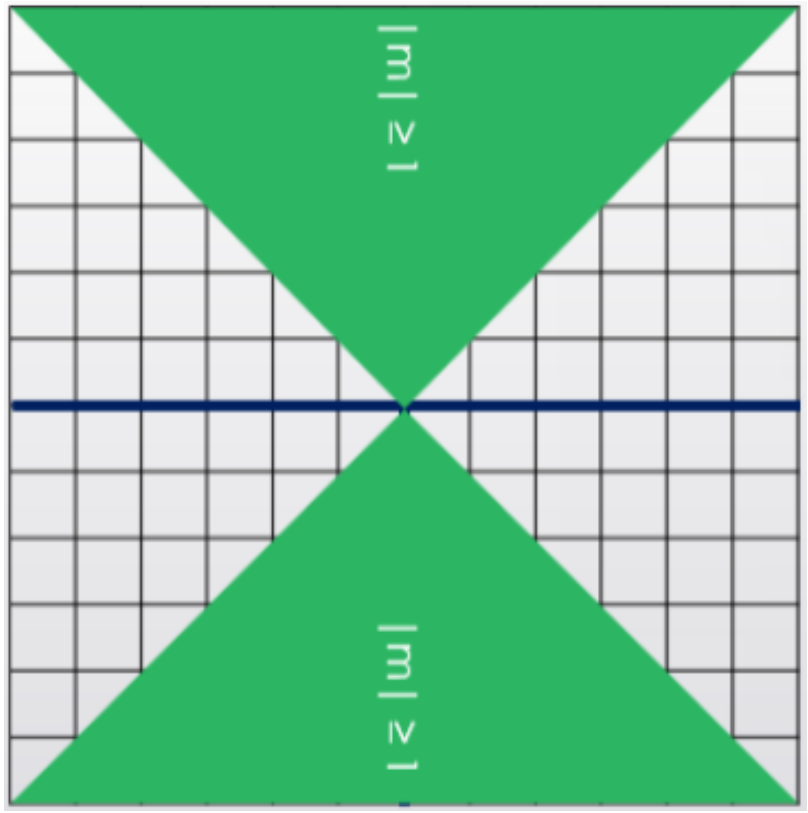
x	y
-3	-2
-2	-1
-1	0
0	1
1	2
2	3
3	4
4	5
5	6



Discretización de Líneas (Segmento de recta)

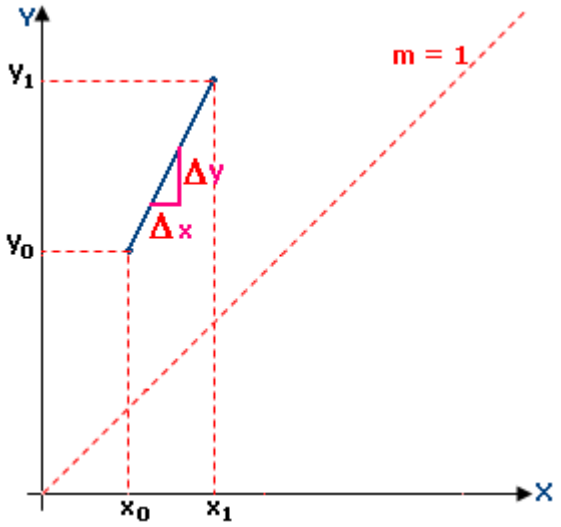
❑ Algoritmo DDA (Digital Diferential Analizer)

Caso 2: $|m| \geq 1$



Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)



Caso 2: m > 1

haciendo $\Delta y = 1 \Rightarrow$ $y_{i+1} = y_i + 1$

tambien $y_{i+1} = m\, x_{i+1} + b$

reemplazando $y_i + 1 = m\, x_{i+1} + b$

$(y_i + 1)/m = x_{i+1} + b/m$

$\Rightarrow x_{i+1} = y_i /m - b/m + 1/m$

Por tanto $x_{i+1} = x_i + 1/m$

En general:

$y_{i+1} = y_i + 1$

$x_{i+1} = x_i + 1/m$ **tal que** $m = \Delta y/\Delta x$

Pintar (round(x_{i+1}), y_{i+1})

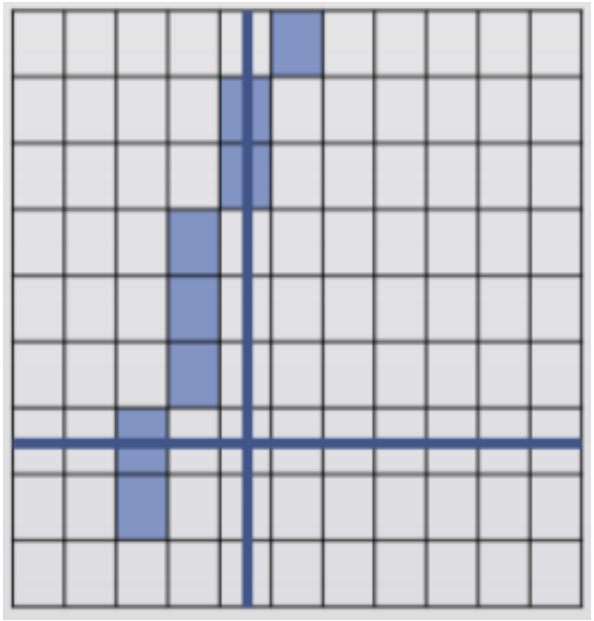
Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)

$(-2,-1) \rightarrow (1,7)$

$m = 2.6 \Longrightarrow x_{i+1} = \frac{1}{2.6} + x_i$

x	y
-2	-1
-1.625	0
-1.25	1
-0.875	2
-0.5	3
-0.125	4
0.25	5
0.625	6
1	7

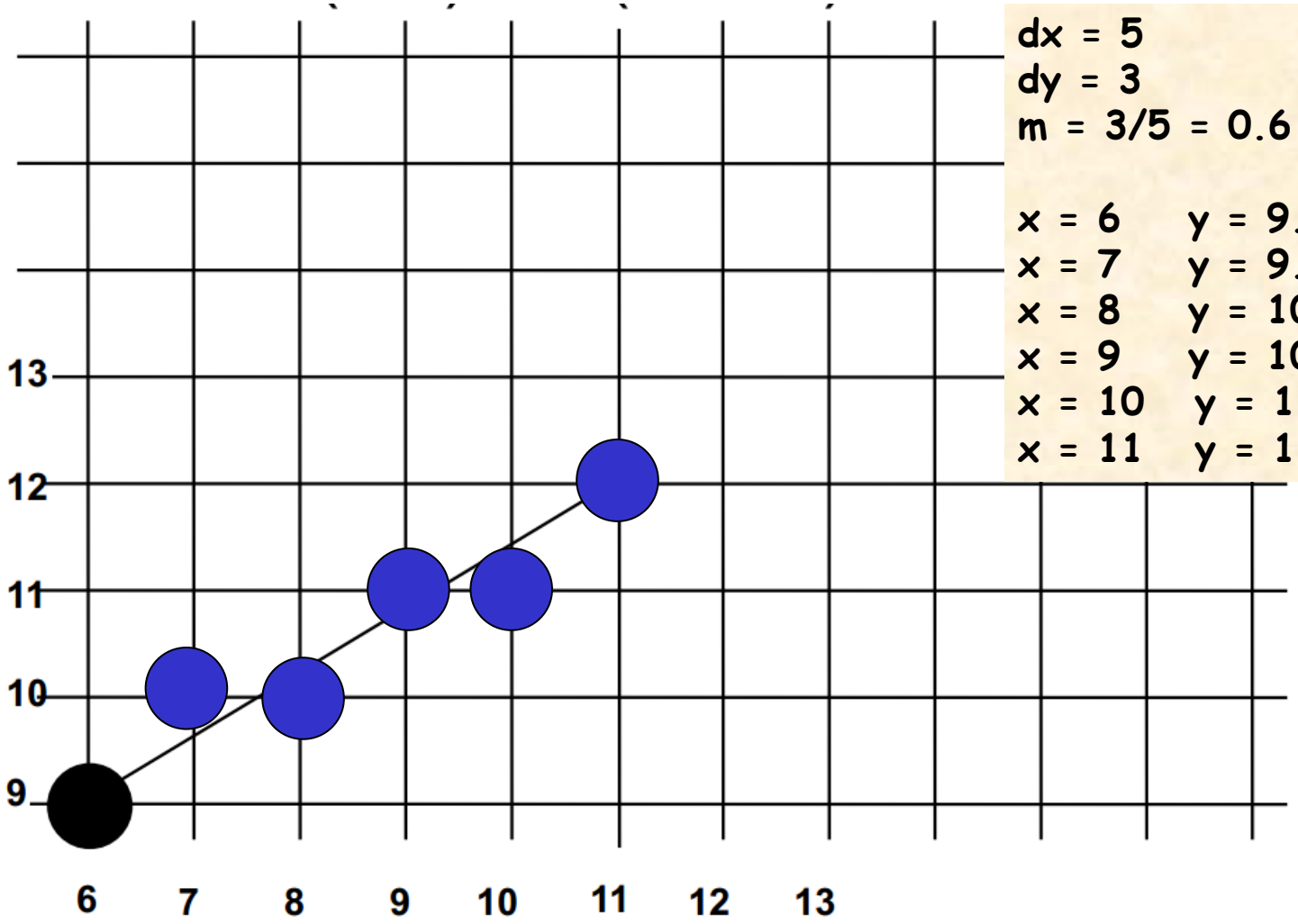


Discretización de Líneas (Segmento de recta)

❑ Algoritmo DDA (Digital Diferential Analizer)

```
void recta_dda(int x0, int y0, int x1, int y1)
{
  int x;
  float dx, dy, m, y;
  dx = x1-x0;
  dy = y1-y0;
  m = dy/dx;
  y = y0;
  for(x=x0; x<=x1; x++)
  {
        pintar(x, round(y), atributo);
        y = y + m;
  }
}
```

Discretización de Líneas (Segmento de recta)



dx = 5
dy = 3
m = 3/5 = 0.6

x = 6	y = 9.00	y = 9
x = 7	y = 9.6	y = 10
x = 8	y = 10.2	y = 10
x = 9	y = 10.8	y = 11
x = 10	y = 11.4	y = 11
x = 11	y = 12.00	y = 12

Discretización de Líneas (Segmento de recta)

Algoritmo DDA (Digital Diferential Analizer)

- **Tiene mayor precisión**
- **El algoritmo discretiza líneas eliminando la multiplicación dentro del bucle, pero aún emplea aritmética en coma flotante.**
- **Acumulación de errores**
  - **Suma repetida de m**
  - **Redondeo**

❑ Algoritmo de Punto Medio (Bresenham)

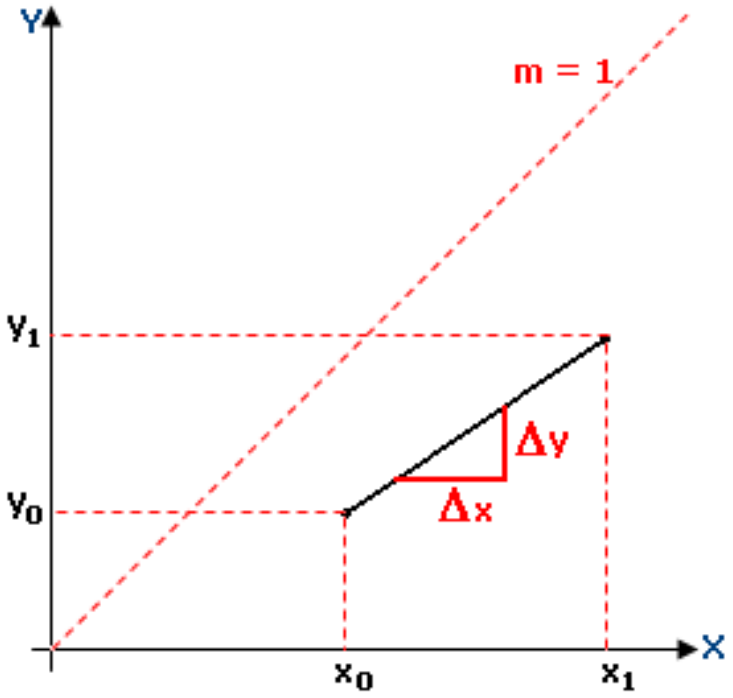
Solo emplea aritmética entera

Calcula el pixel (x_{i+1} , y_{i+1}) a partir de (x_i , y_i)

Supongamos el caso de $0 < m < 1$

Punto extremo inferior: (x_0 , y_0)

Punto extremo superior: (x_1 , y_1)



Discretización de Líneas (Segmento de recta)

❑ Algoritmo de Punto Medio (Bresenham)

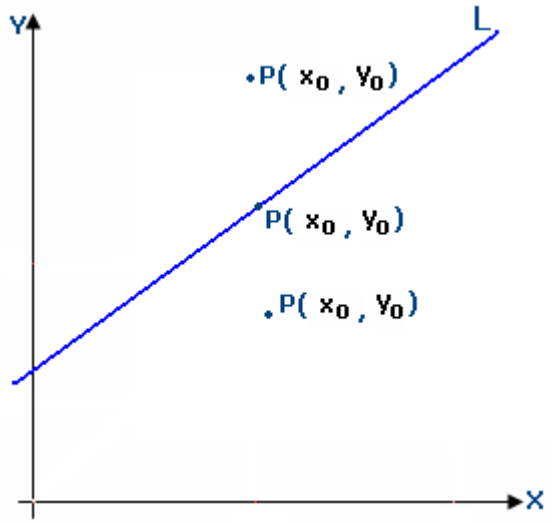
Sea L: F(x, y) = Ax + By +C ecuación general de la recta.

Sea $P(x_0 , y_0)$ un punto cualesquiera, entonces:

i) Si $F(x_0 , y_0) = 0 \Rightarrow P(x_0 , y_0) \varepsilon L$

ii) Si $F(x_0 , y_0) > 0 \Rightarrow P(x_0 , y_0)$ se encuentra debajo de la recta

iii) Si $F(x_0 , y_0) < 0 \Rightarrow P(x_0 , y_0)$ se encuentra encima de la recta



Discretización de Líneas (Segmento de recta)

❑ Algoritmo de Punto Medio (Bresenham)

NE: Pixel en dirección Nor Este

E : Pixel en dirección Este

Q : Punto perteneciente a la recta

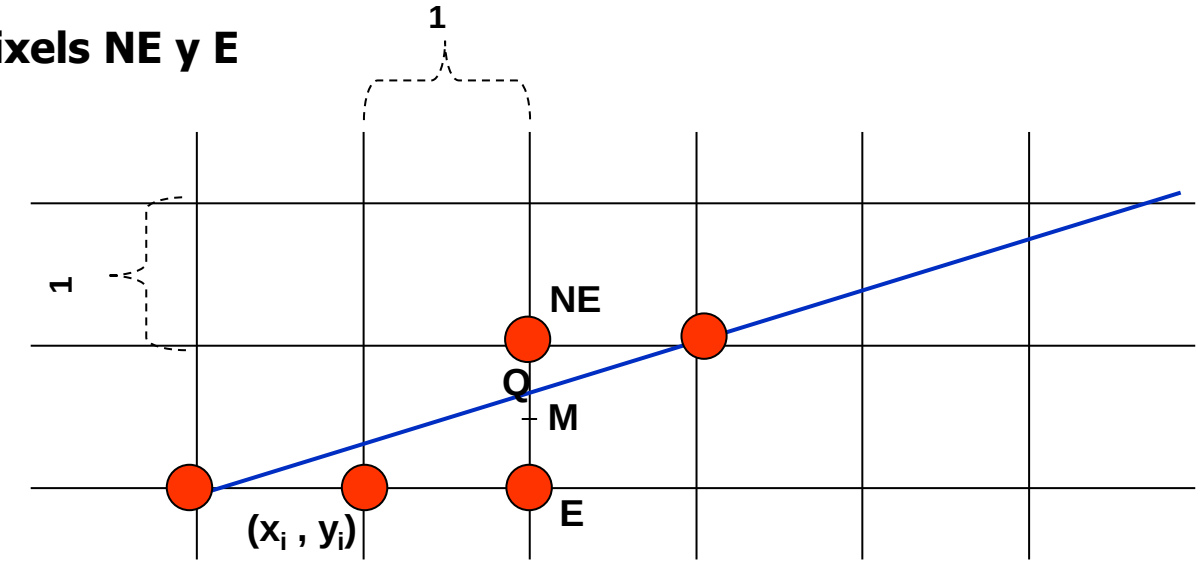
M : punto medio entre pixels NE y E

$NE = (x_i + 1, y_i + 1)$

$E = (x_i + 1, y_i)$

$Q = (x , y) \varepsilon L$

$M = (x_i + 1, y_i + ½)$



Discretización de Líneas (Segmento de recta)

❑ Algoritmo de Punto Medio (Bresenham)

De la ecuación de la recta: $y = mx + b$

$$y = \Delta y/\Delta x\, x + b$$

$$\Delta y\, x - \Delta x\, y + \Delta x\, b = 0$$

$\Rightarrow$ $F(x, y) = \Delta y\, x - \Delta x\, y + \Delta x\, b = 0 \quad \forall\, (x, y) \in L$

Donde $A = \Delta y$, $B = -\Delta x$, $C = \Delta x b$

Se trata de elegir entre los pixels NE y E, esto dependerá del valor numérico de F(x, y) en el punto medio entre NE y E.

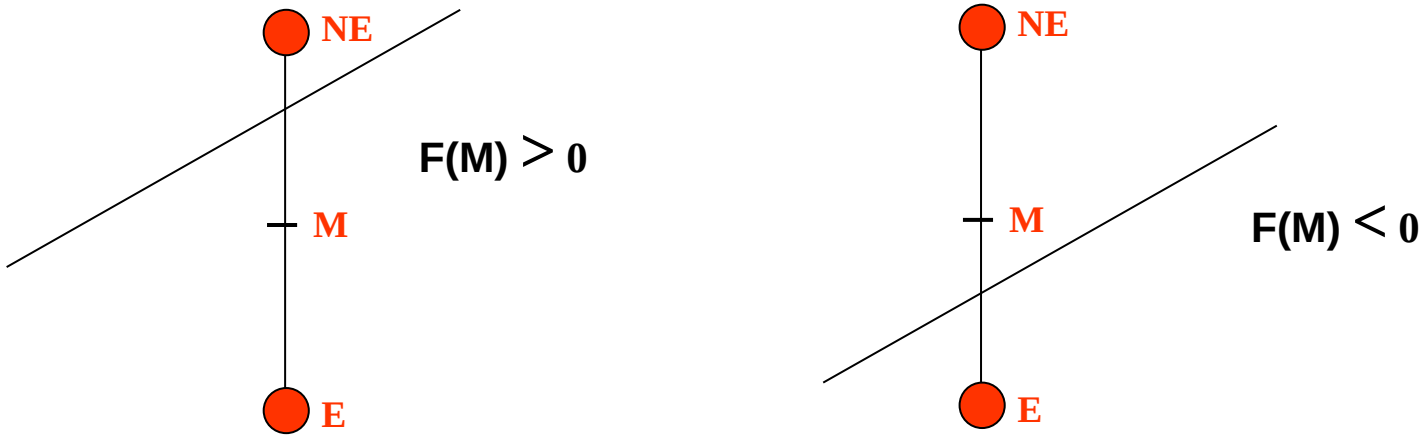
$$d = F(M) \quad \text{donde} \quad M = (x_i + 1, y_i + ½)$$

❑ Algoritmo de Punto Medio (Bresenham)

i) **Si d = F(M) > 0, entonces M se encuentra debajo de la recta y el punto NE se encuentra más próxima a la recta.**

ii) **Si d = F(M) < 0, entonces M se encuentra encima de la recta y el punto E se encuentra más próxima a la recta.**

iii) **Si d = F(M) = 0, entonces se puede elegir cualquiera.**



❑ Algoritmo de Punto Medio (Bresenham)

Caso 1: Cuando elegimos a E:

$d_{nuevo} = F(x_i + 2, y_i + ½)$

$d_{nuevo} = A(x_i + 2) + B(y_i + ½) + C$

$d = F(x_i + 1, y_i + ½)$

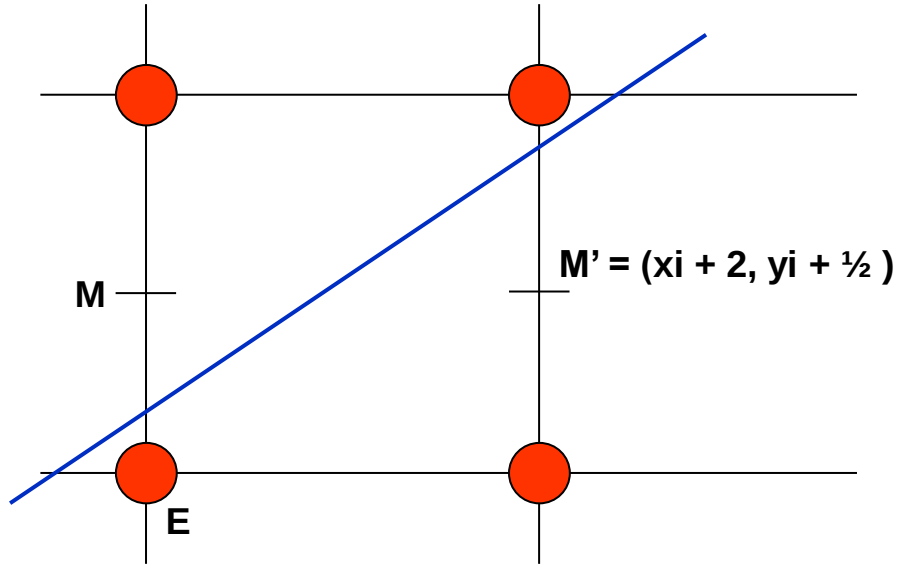
$d = A(x_i + 1) + B(y_i + ½) + C$

Restando d_{nuevo} y d:

$$d_{nuevo} - d = A$$

El incremento que se usa despues de elegir E se denomina ΔE y su valor es:

$$\Delta E = d_{nuevo} - d = A = \Delta y$$



Discretización de Líneas (Segmento de recta)

❑ Algoritmo de Punto Medio (Bresenham)

Caso 2: Cuando elegimos a NE:

$d_{nuevo} = F(x_i + 2, y_i + 3/2)$

$d_{nuevo} = A(x_i + 2) + B(y_i + 3/2) + C$

$d = F(x_i + 1, y_i + ½)$

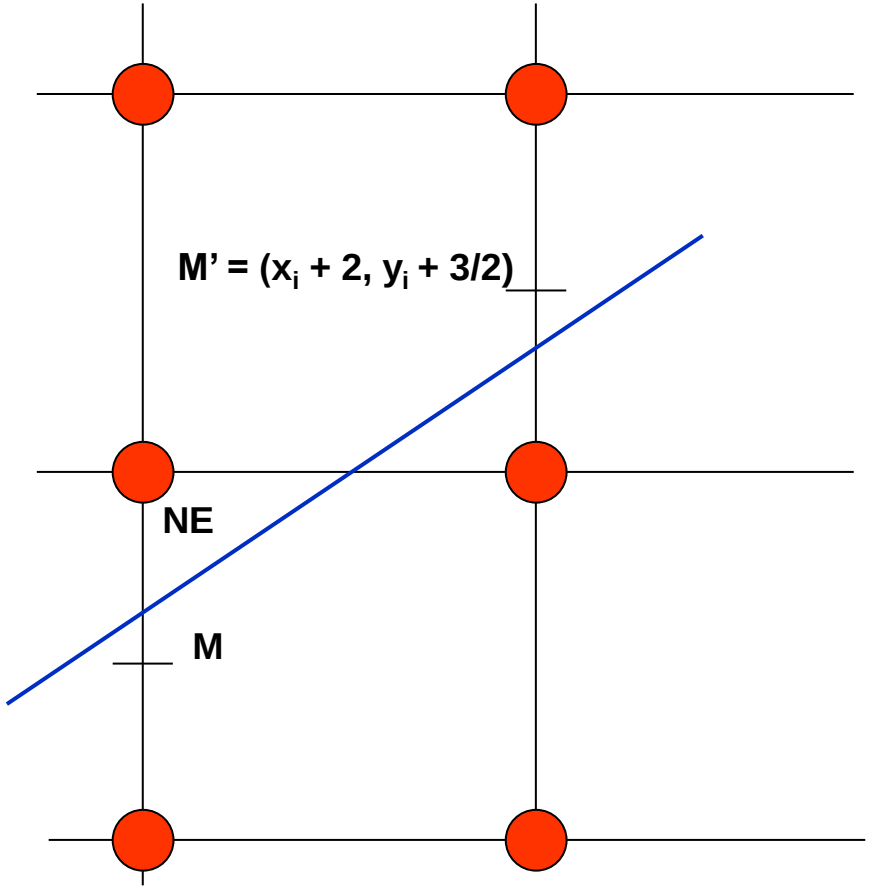
$d = A(x_i + 1) + B(y_i + ½) + C$

Restando d_{nuevo} y d:

$$d_{nuevo} - d = A + B$$

El incremento que se usa despues de elegir NE se denomina ΔNE y su valor es:

$$\Delta NE = d_{nuevo} - d = A+B = \Delta y - \Delta x$$



❑ Algoritmo de Punto Medio (Bresenham)

Para $d_{inicial}$:

$$d_{inicial} = F(x_0 + 1, y_0 + ½)$$

$$d_{inicial} = A(x_0 + 1) + B(y_0 + ½) + C$$

$$d_{inicial} = Ax_0 + By_0 + C + A + B/2$$

$$d_{inicial} = F(x_0 , y_0) + A + B/2$$

Pero $F(x_0 , y_0) \in L \Rightarrow F(x_0 , y_0) = 0$

$$d_{inicial} = A + B/2$$

$$d_{inicial} = \Delta y - \Delta x/2$$

❑ Algoritmo de Punto Medio (Bresenham)

Para eliminar la división por 2 se redefine F(x, y) multiplicándolo por 2, lo cual no afecta el signo de la variable de decisión:

$$F(x, y) = 2\Delta y\, x - 2\Delta x\, y + 2\Delta x\, b = 0 \quad \forall\, (x, y) \in L$$

Por tanto: $d_{inicial} = 2\Delta y - \Delta x$

También: $\Delta E = 2\Delta y$

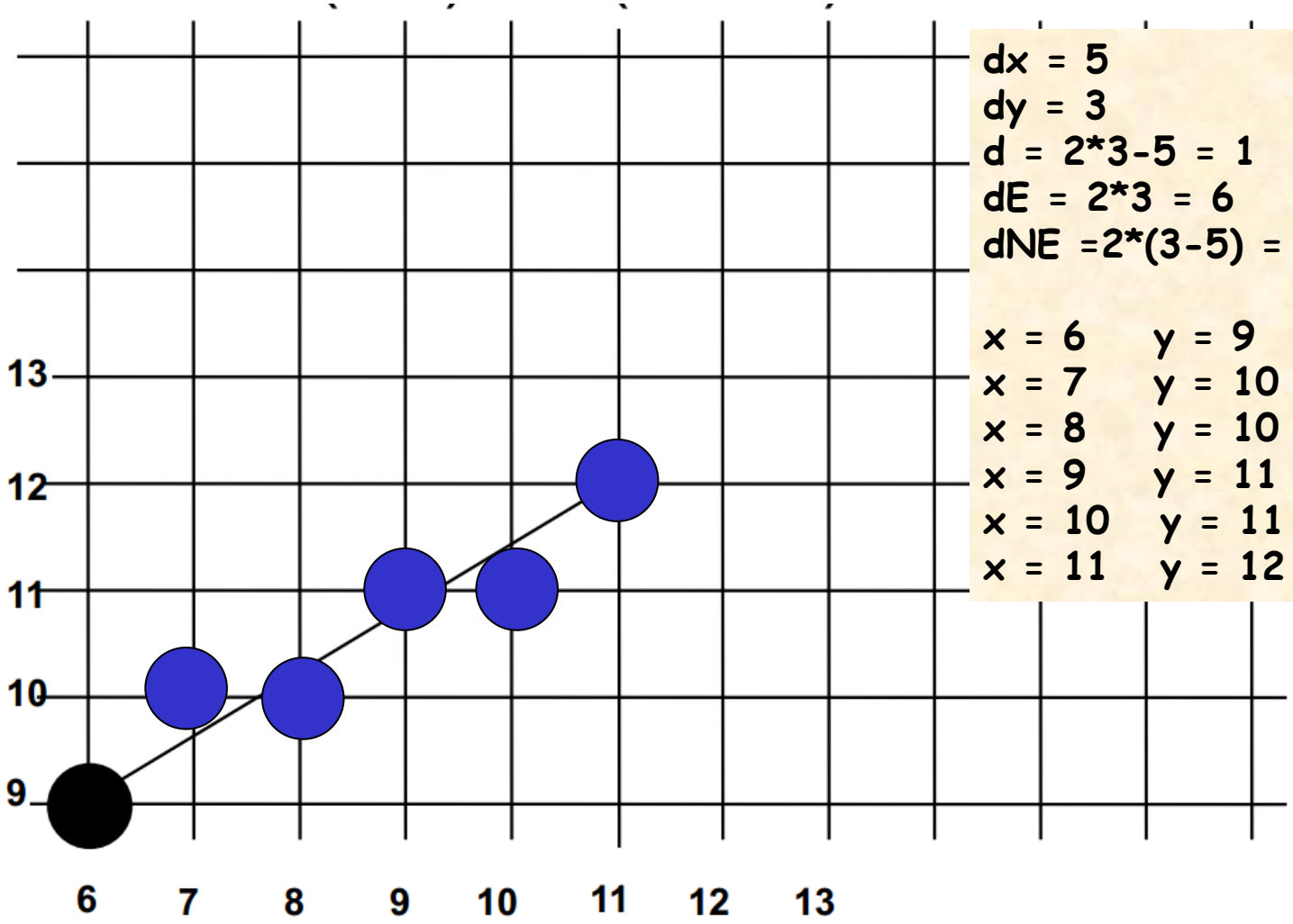
$\Delta NE = 2(\Delta y - \Delta x)$

Discretización de Líneas (Segmento de recta)

❑ Algoritmo de Punto Medio (Bresenham)

```
void recta_punto_medio(int x0, int y0, int x1, int y1)
{
  int dx, dy, dE, dNE, d, x, y;
  dx = x1 - x0;
  dy = y1- y0;
  d = 2*dy - dx;
  dE = 2*dy;
  dNE = 2*(dy - dx);
  x = x0;
  y = y0;
  pintar(x, y, atributo);
  while (x < x1)
  {
        if(d<=0){
              d = d + dE;
              x = x + 1;
        }
        else{
              d = d + dNE;
              x = x + 1;
              y = y + 1;
        }
        pintar(x, y, atributo);
        }
}
```

Discretización de Líneas (Segmento de recta)



dx = 5
dy = 3
d = 2*3-5 = 1
dE = 2*3 = 6
dNE =2*(3-5) = -4

x = 6	y = 9	d = 1
x = 7	y = 10	d = 1-4=-3
x = 8	y = 10	d = -3+6=3
x = 9	y = 11	d = 3-4=-1
x = 10	y = 11	d = -1+6=5
x = 11	y = 12	d = -5+6=1

Discretización de Circunferencias

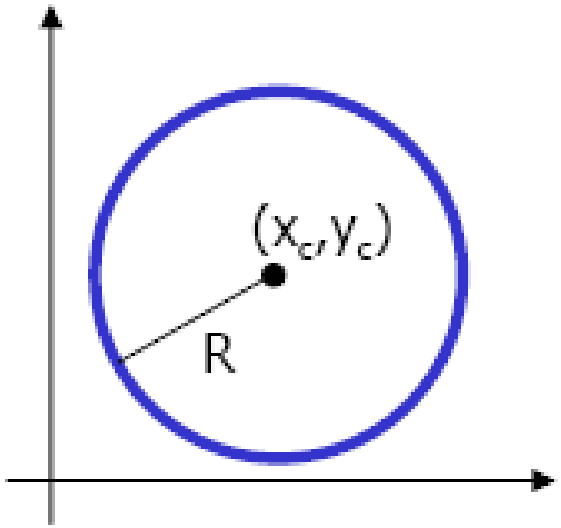
La ecuación de una circunferencia con centro en (x_0 , y_0) es:

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

Donde R es el radio de la circunferencia.

Restringiremos el estudio de la circunferencia con centro en el origen de coordenadas:

$$x^2 + y^2 = R^2$$



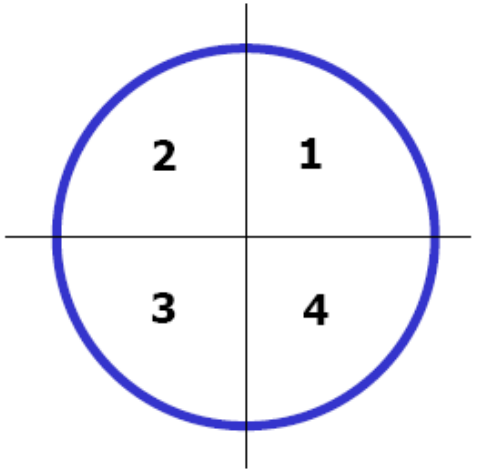
Una primera aproximación para discretizar una circunferencia es resolver la ecuación:

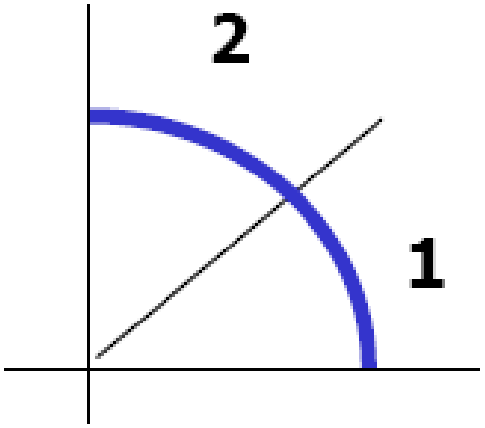
$$Y = \pm (R^2 - x^2)^{1/2} \quad \forall\, x \in Z \quad y \quad 0 \le x \le R$$

❑ Simetría de la circunferencia

Para dibujar la circunferencia Es posible también reducir el cálculo al considerar la simetría de las circunferencias.

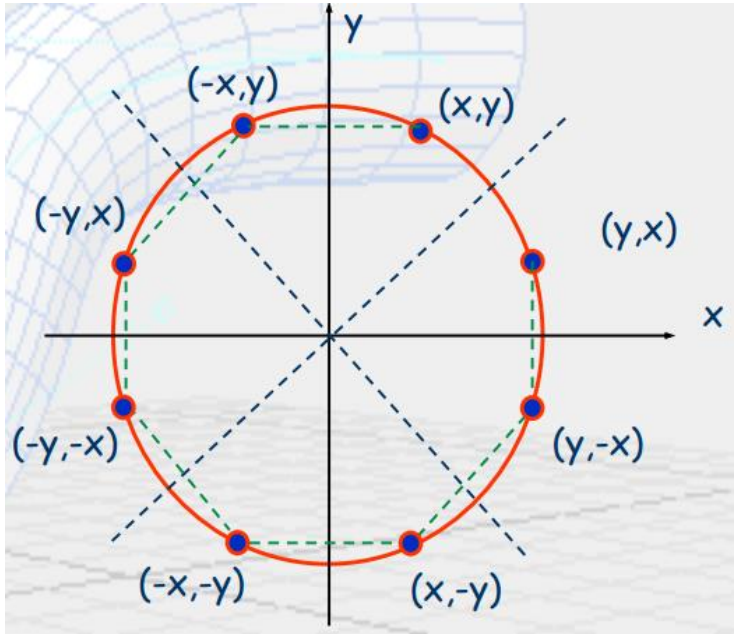
Incluso el primer octante es simétrico al segundo con respecto a la recta y = x.





❑ Simetría de la circunferencia

Dibujando sólo el segundo octante, desde x = 0 hasta x = y podemos pintar todo el círculo




```
void pintar_circunferencia (int x, int y, int atributo);
{
      pintar (x, y, atributo);
      pintar (y, x, atributo);
      pintar (y, -x, atributo);
      pintar (x, -y, atributo);
      pintar (-x, -y, atributo);
      pintar (-y, -x, atributo);
      pintar (-y, x, atributo);
      pintar (-x, y, atributo);
}
```

❑ Algoritmo de fuerza bruta

Usamos la ecuación: $Y = \pm (R^2 - x^2)^{1/2} \quad \forall x \in Z$

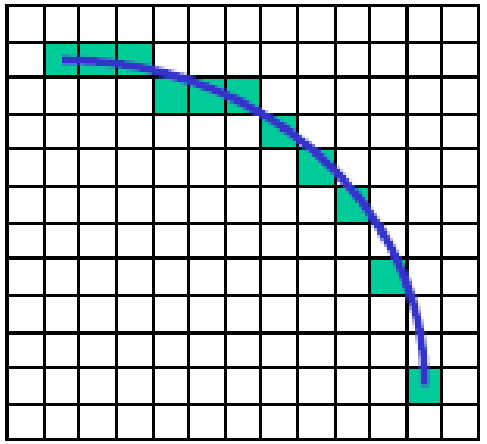
Para el primer octante: $0 \leq x \leq R$

En general:

$x_{i+1} = x_i + 1$

$y_{i+1} = (R^2 - x_{i+1}^2)^{1/2}$

Pintar $(x_{i+1}, \text{round}(y_{i+1}))$



A medida que los valores de x_i se acercan a R los pixels pintados (x_i, y_i) serán muy discontínuos.

❑ Algoritmo de fuerza bruta

Se divide el primer cuadrante en dos partes (octantes).

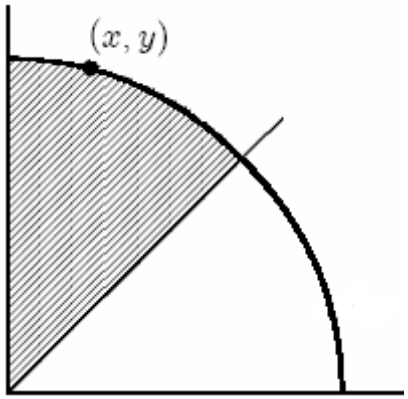
En general:

$x_i < y_i$

$x_{i+1} = x_i + 1$

$y_{i+1} = (R^2 - x_{i+1}^2)^{1/2}$

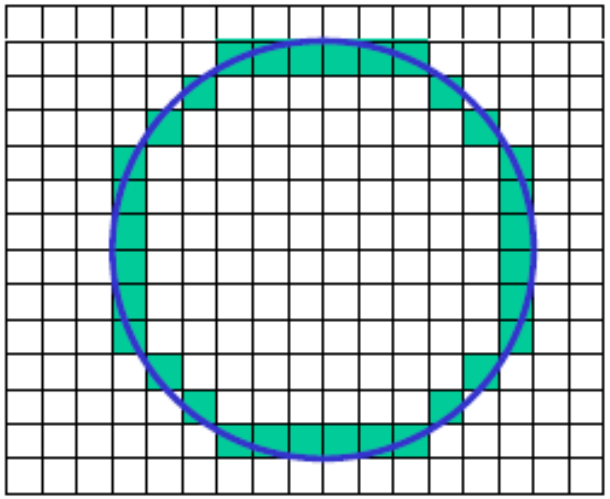
Pintar $(x_{i+1}, \text{round}(y_{i+1}))$




```
void circunferencia_fuerza_bruta (int R)
{
     float x = 0,y = R;
     while (x < y)
     {
          y = sqrt(pow(R,2)-pow(x,2));
          pintar (x, round(y),atributo);
          x = x + 1;
     }
}
```

Algoritmo de fuerza bruta

- No es nada eficiente
- Cada paso requiere una raíz cuadrada
- El espaciado entre pixels no es uniforme
- Demasiado costo computacional

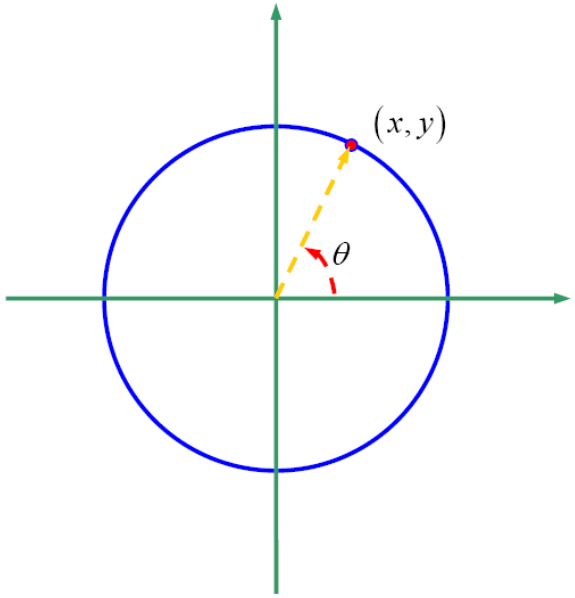


❑ Algoritmo basado en la representación paramétrica

Se usa las siguientes ecuaciones:

$x = R \cos \theta$

$y = R \operatorname{sen} \theta$ donde $0 \leq \theta \leq 90^o$



❑ Algoritmo basado en la representacón paramétrica

- Los cuadrantes II, II y IV se obtienen por simetría de la circunferencia respecto al origen.
- El valor del incremento del ángulo θ debe ser lo suficientemente pequeño para evitar los huecos
- Uso de funciones trigonométricas
- Demasiado costo computacional

```
void circunferencia_parametrica(int R)
{
    float x,y;
    float PI=3.1415……..;
    float teta=PI/4;
    delta=0.1;
    while (teta<PI/2)
    {
        x = R*cos(teta);
        y = R*sin(teta);
        teta = teta + delta;
        pintar(round(x),round(y),atributo);
    }
}
```

❑ Algoritmo de punto medio

Sea la ecuación de la circunferencia con centro en el origen de coordenadas (0, 0):

$x^2 + y^2 = R^2$, R: radio de la circunferencia

Comenzamos a pintar en el punto (0, R) y se va desde X =0 hasta x = y, donde la pendiente va de 0 a -1

Sea C: $F(x, y) = x^2 + y^2 - R^2 = 0$ ecuación general de la de la circunferencia.

Sea $P(x_0 , y_0)$ un punto cualesquiera, entonces:

i) Si $F(x_0 , y_0) = 0 \Rightarrow P(x_0 , y_0) \varepsilon C$

ii) Si $F(x_0 , y_0) > 0 \Rightarrow P(x_0 , y_0)$ se encuentra fuera de la C

iii) Si $F(x_0 , y_0) < 0 \Rightarrow P(x_0 , y_0)$ se encuentra dentro de la C

❑ Algoritmo de punto medio

Queremos pasar del pixel (x_i , y_i) al pixel (x_{i+1} , y_{i+1})

E: Pixel en dirección Este

SE : Pixel en dirección Sur Este

Q : Punto perteneciente a la circunferencia

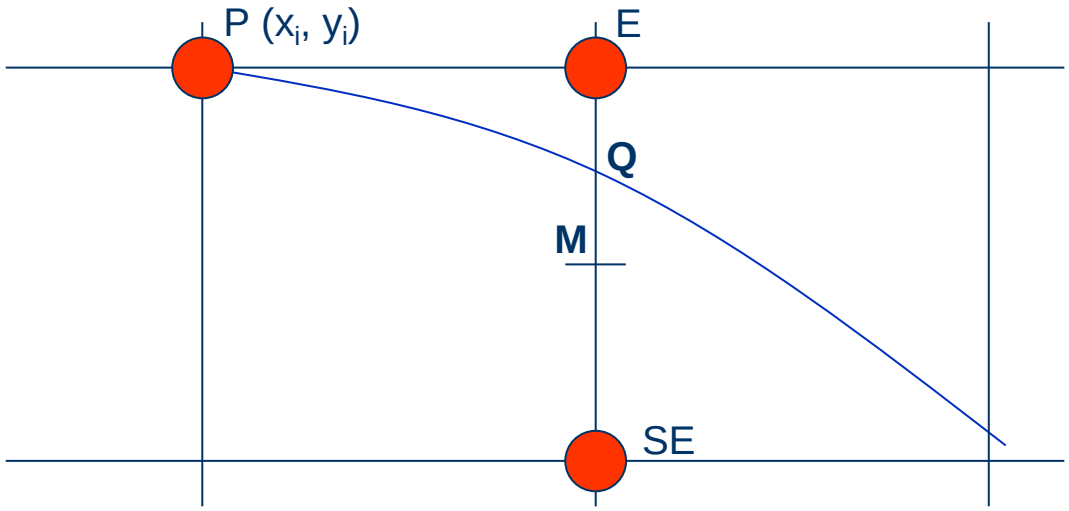
M : punto medio entre pixels E y SE

$E = (x_i + 1, y_i)$

$SE = (x_i + 1, y_i - 1)$

$Q = (x , y) \varepsilon C$

$M = (x_i + 1, y_i - ½)$



❑ Algoritmo de punto medio

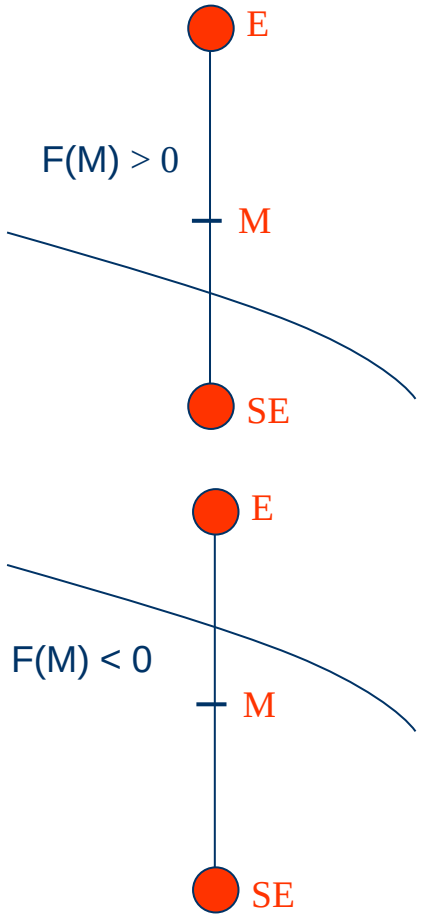
Se trata de elegir entre los pixels E y SE, esto dependerá del valor numérico de F(x, y) en el punto medio entre E y SE.

$$d = F(M) \quad \text{donde} \quad M = (x_i + 1, y_i - ½)$$

i) **Si d = F(M) > 0, entonces M se encuentra fuera de la circunferencia y el punto SE se encuentra más próximo a la circunferencia.**

ii) **Si d = F(M) < 0, entonces M se encuentra dentro de la circunferencia y el punto E se encuentra más próxima a la circunferencia.**

iii) **Si d = F(M) = 0, entonces se puede elegir cualquiera.**



❑ Algoritmo de punto medio

Los valores de M y d para el siguiente punto depende de la elección de E o SE.

Caso 1: Cuando elegimos a E:

$d_{nuevo} = F(x_i + 2, y_i - ½)$

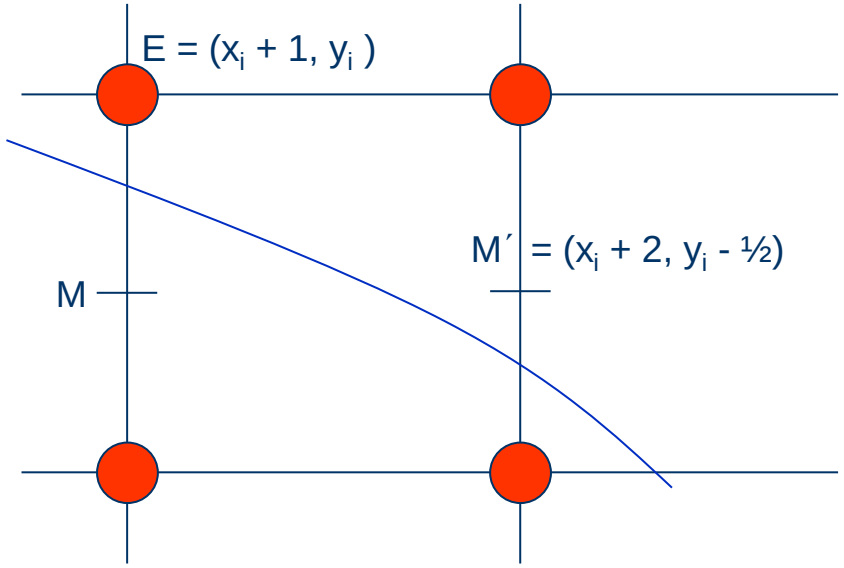
$d_{nuevo} = (x_i + 2)^2 + (y_i - ½)^2 - R^2$

$d = F(x_i + 1, y_i - ½)$

$d = (x_i + 1)^2 + (y_i - ½)^2 - R^2$

Restando d_{nuevo} y d:

$$d_{nuevo} - d = 2x_i + 3$$



El incremento que se usa despues de elegir E se denomina ΔE y su valor es:

$$\Delta E = d_{nuevo} - d = 2x_i + 3$$

❑ Algoritmo de punto medio

Caso 2: Cuando elegimos a SE:

$d_{nuevo} = F(x_i + 2, y_i - 3/2)$

$d_{nuevo} = (x_i + 2)^2 + (y_i - 3/2)^2 - R^2$

$d = F(x_i + 1, y_i - ½)$

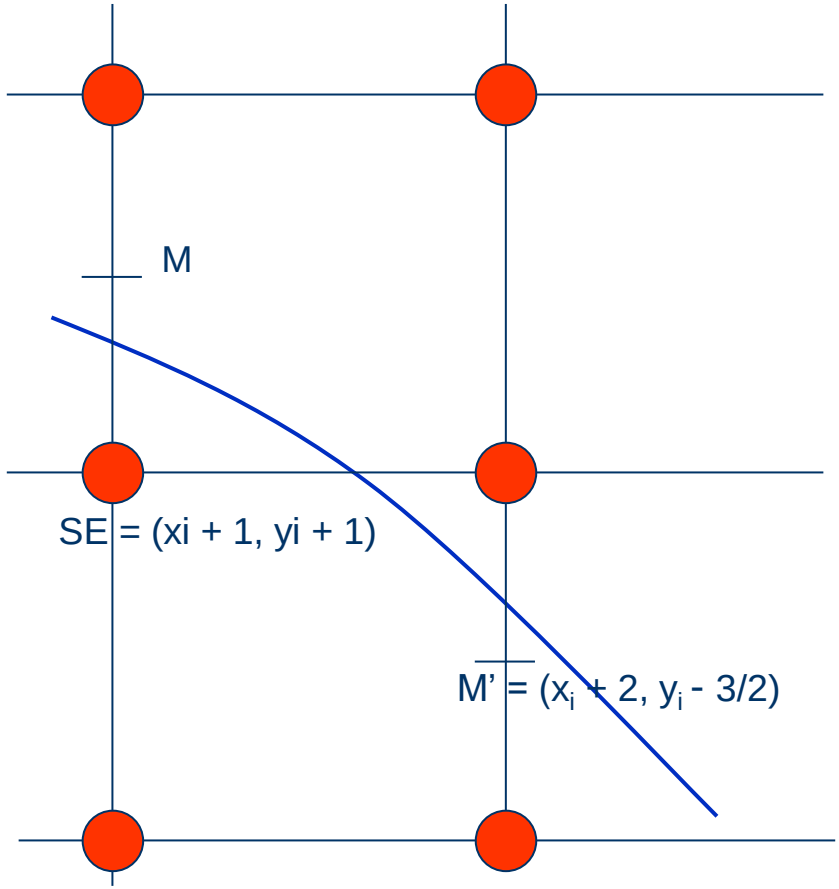
$d = (x_i + 1)^2 + (y_i - ½)^2 - R^2$

Restando d_{nuevo} y d:

$$d_{nuevo} - d = 2x_i - 2y_i + 5$$

El incremento que se usa despues de elegir SE se denomina ΔSE y su valor es:

$$\Delta SE = d_{nuevo} - d = 2x_i - 2y_i + 5$$



❑ Algoritmo de punto medio

Para $d_{inicial}$ el punto de partida es (0, R):

$$d_{inicial} = F(0 + 1, R - ½) = F(1, R - ½)$$

$$d_{inicial} = (1)^2 + (R - ½)^2 - R^2$$

$$d_{inicial} = 5/4 - R$$

Por tanto: $d_{inicial} = 5/4 - R$

También: $\Delta E = 2x_i + 3$

$\Delta SE = 2x_i - 2y_i + 5$

❑ Algoritmo de punto medio

```
void circunferencia_punto_medio(int R)
{
     // discretizacion valida en el II octant
     int x=0, y=R;
     float d=5/4-R;
     pintar(x,y,atributo);
     while (x<y){
     if (d<0)      //se escoge el pixel E
        d = d + 2*x+3;
        x = x + 1:
     else{          //se escoge el pixel SE
      d = d + 2(x-y)+5;
      x = x + 1;
          y = y - 1;
          }
     pintar(x,y,atributo);
     }
}
```

R=10
d=-8.75

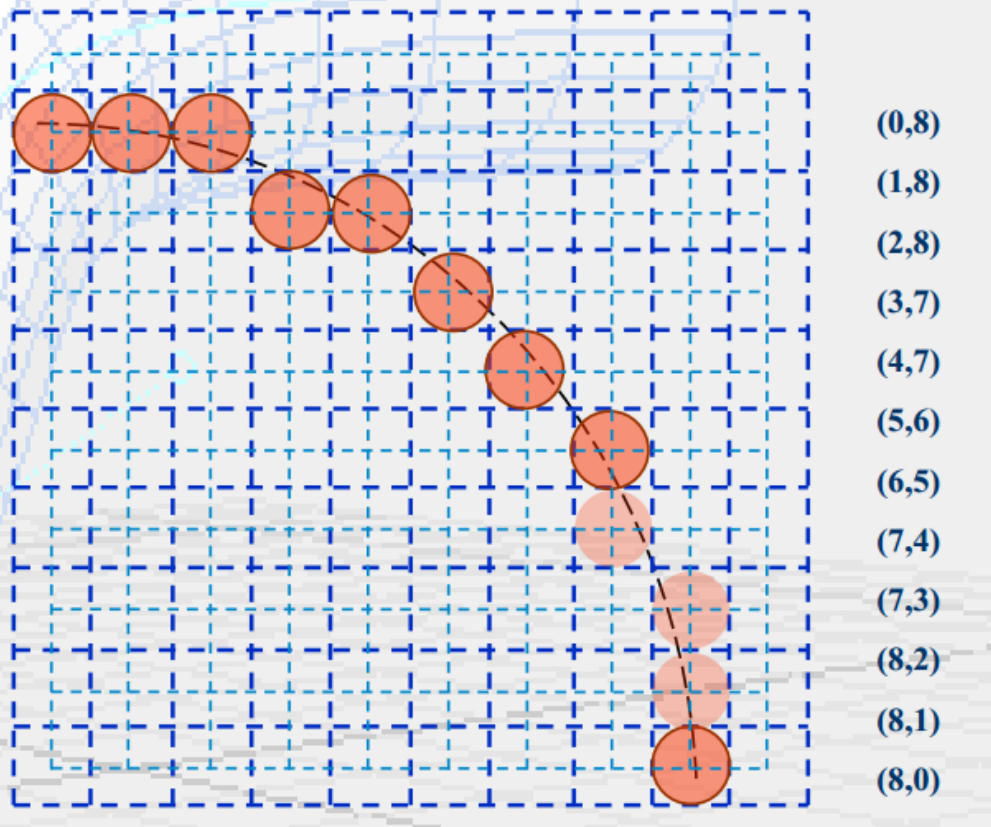
x = 0 y = 10 d = -8.75
x = 1 y = 10 d = -3.75
x = 2 y = 10 d = 4.75
x = 3 y = 9 d = -4.75
x = 4 y = 9 d = 7.75
x = 5 y = 8 d = 6.75
x = 6 y = 7 d = 9.75
x = 7 y = 6 d = -11.75

Suma x = 92, suma y = 100

Suma x = 92, suma y = 100

Discretización de Circunferencias

- Algoritmo de Bresenham:
  - Aritmética totalmente entera.
  - Bajo costo computacional.
  - Bajo consumo de recursos gráficos.
  - Explota la simetría de la gráfica.



❑ Algoritmo de punto medio

Planteamiento similar al despliegue de una circunferencia.

i) Se determina los puntos (x, y) para una elipse en posición estándar centrada en el origen.

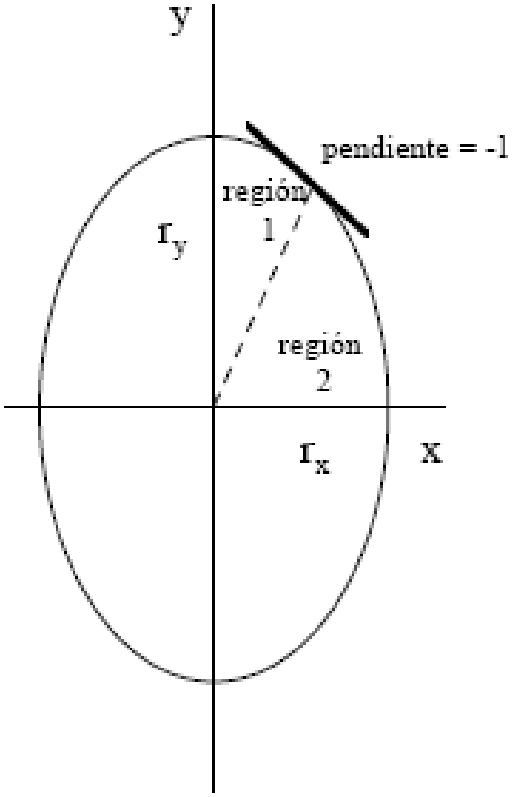
$$X^2 / rx^2 + y^2 / ry^2 = 1$$

ii) La pendiente de la curva (la tangente) se calcula a partir de la ecuación

$$f_{elipse}(x,y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 = 0$$

$$dy/dx = -2 r_y^2 x / 2 r_x^2 y$$

En la frontera entre la región 1 y la región 2 dy/dx = - 1 (dy/dx < -1 en la región 1 y dy/dx > -1 en la región 2),



BIBLIOGRAFIA

- Computação Gráfica – Eduardo Azevedo y Aura Conci
- Computer Graphics: Principles and Practice. Foley J., Van Dame A., Feiner S., Hughes J., Phillips R. Addison – Wesley Publishing Company, Massachusetts. 1996
- Fundamentals of Computer Aided Geometric Design. Hoschek J., Lasser D. A.K. Peters Ltd. Wellesley Massachusetts. 1993
- Gráficas por computadora. Hearn D., Baker M.P. Prentice - Hall Hispanoamericana. 1998

¿PREGUNTAS?

Facultad de Ingeniería de Ingeniería de Sistemas
Departamento de Ciencias de la Computación
UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS