Graficas por computadora 2D

Unidad I

José Francisco Pérez Reyes

Hardware y representación de imágenes

¿Consideras importante la representación gráfica en el mundo de las Tecnologías de la Información?

Hardware y representación de imágenes

¿En donde crees que se requieren o implementan soluciones basadas en graficación por computador?

Hardware y representación de imágenes

Los gráficos por computadora surgieron a principios de los **años cincuenta en el famoso MIT** (Instituto Tecnológico de Massachusetts). En él **se conectó un osciloscopio a uno de sus ordenadores, de modo que éste podía controlar directamente la posición del haz catódico**. Este sistema podía realizar dibujos simples, **moviendo el haz de manera que fuese trazando las líneas que lo componían**. Sin embargo, cuando la complejidad del gráfico aumentaba, aparecía un problema de difícil solución: **el parpadeo**. Este surgía porque el punto trazado por el haz desaparecía unas décimas de segundo después de que el haz se hubiese movido, lo que obligaba a redibujar constantemente el gráfico (a este proceso se le denomina renderizado).

Hardware y representación de imágenes

Posteriormente **se ideó un tipo de pantalla que mantenía la luminosidad de un punto incluso después de que el haz dejase de iluminarlo**, usando una especie de condensador entre el fósforo y el tubo. Para borrar la pantalla, simplemente se descargaba ese condensador. **Este sistema tenía la ventaja de que eliminaba totalmente el renderizado** de la imagen, pues ésta se mantenía aunque el haz dejase de pasar. Sin embargo, **tenía el problema de que no se podía borrar una parte de la pantalla**, sino que se debía borrar toda y reescribir la parte común. A esto hay que unir el bajo contraste de la imagen.

Hardware y representación de imágenes

Este sistema de generación de imágenes es denominado **VECTORIAL** pues sus imágenes están compuestas por vectores unidos entre sí. Pronto se hizo evidente que **este sistema no era excesivamente práctico**, así que se empezó a trabajar en uno más perfeccionado. El resultado fue el **sistema RASTER SCAN**. En él, **no es la CPU la que controla el tubo catódico, sino que existe una circuitería independiente que realiza todo el trabajo**. Esta vez, el haz no crea la imagen a partir de segmentos o vectores, sino que lo hace a partir de puntos (los famosos PIXELS). Para ello, el haz recorre toda la pantalla en líneas horizontales, de izquierda a derecha y de arriba a abajo.

Hardware y representación de imágenes

Dos formas de tarjetas gráficas:

1. La placa incorpora un microprocesador propio, y acepta ordenes de alto nivel desde la CPU, como trazar rectas, círculos, mover SPRITES automáticamente, actualizar la paleta de colores, etc. Llevan un chip específico denominado VDP (Video Display Procesor) capaz de realizar automáticamente rotaciones en 3D de una imagen, y multitud de efectos.

2. Una circuitería que simplemente lee la imagen de la memoria de video y la plasma en pantalla. La diferencia es que la CPU accede directamente a esa memoria y escribe en ella los datos necesarios. Esto significa que tiene que estar situada dentro del espacio direccionable por la CPU, como una parte más de la memoria principal. Tiene la ventaja de que es un sistema mucho más barato y simple que el anterior.

Hardware y representación de imágenes

Coordenadas Universales, Visuales y Específicas

José Francisco Pérez Reyes

Hardware y representación de imágenes

Coordenadas Específicas: A la hora de trabajar con un ordenador, uno se da cuenta de la gran diversidad de dispositivos de salida que hay. Así, existen monitores, impresoras, plotters, etc., cada uno con sus propias características y capacidades. Por si fuera poco, nos encontramos con que también existen diferencias entre ellos mismos; así, anteriormente nos encontramos con tarjetas gráficas CGA, EGA, VGA, Súper VGA, VESA, etc., con impresoras y plotters con especificaciones de puntos por pulgada. Todo esto hace que, en principio, el software tenga que estar adaptado a las resoluciones de cada uno. Esto es debido a que trabajamos con las COORDENADAS ESPECIFICAS del aparato.

Hardware y representación de imágenes

Coordenadas Universales: Se usan para definir cada uno de los objetos en la base de datos. Cada uno se define con el origen de coordenadas en su centro. No existe un límite máximo en ningún eje, sino que los objetos pueden ser tan grandes como necesitemos.

Coordenadas Visuales: El origen se encuentra en el ojo del observador, en el centro del dispositivo de representación. Aquí lo que se hace es modificar las coordenadas de cada objeto mediante fórmulas de rotación y traslación para situarlos en el sitio y en la posición del espacio que le corresponde a cada uno. Al igual que antes, no existe límite en el valor de estas coordenadas. No olvidemos que trabajamos con todo el espacio.

Hardware y representación de imágenes

¿Cómo solucionarías este problema de coordenadas?

Hardware y representación de imágenes

Implementando y una técnica de relación de aspecto

Hardware y representación de imágenes

Una imagen puede ser representada como un conjunto de puntos. Cada uno de esos puntos tiene un color asociado. El número de puntos que representa una imagen puede variar según la definición que queramos obtener. **Cuantos más puntos sea representada una imagen más calidad tendrá y más espacio necesitaremos para almacenarla**. Este aumento de tamaño provocará un aumento de tiempo de computación para que pueda ser mostrada. Esto se traduce en un mayor consumo de espacio de disco y que las transferencias de datos que hace que la imagen se nos muestre en la pantalla de nuestro ordenador estén más cargadas al tener más datos que intercambiar.

Hardware y representación de imágenes

Estos puntos a los que hacemos referencia **se conocen como píxeles y determinan la resolución de una imagen**. Los monitores tienen una limitación en cuanto al número de píxeles que pueden mostrar. Esta limitación es un máximo ya que siempre podrá mostrar un número menor de píxeles. **Cada uno de estos píxeles tendrá asociado un color para los que se destinará un determinado número de bits (bits por píxel o bpp)** que determinaran la calidad, medida en profundidad de color, de la imagen.

Hardware y representación de imágenes

8 bits (256 colores), debemos usar una paleta para establecer los distintos 256 posibles colores.

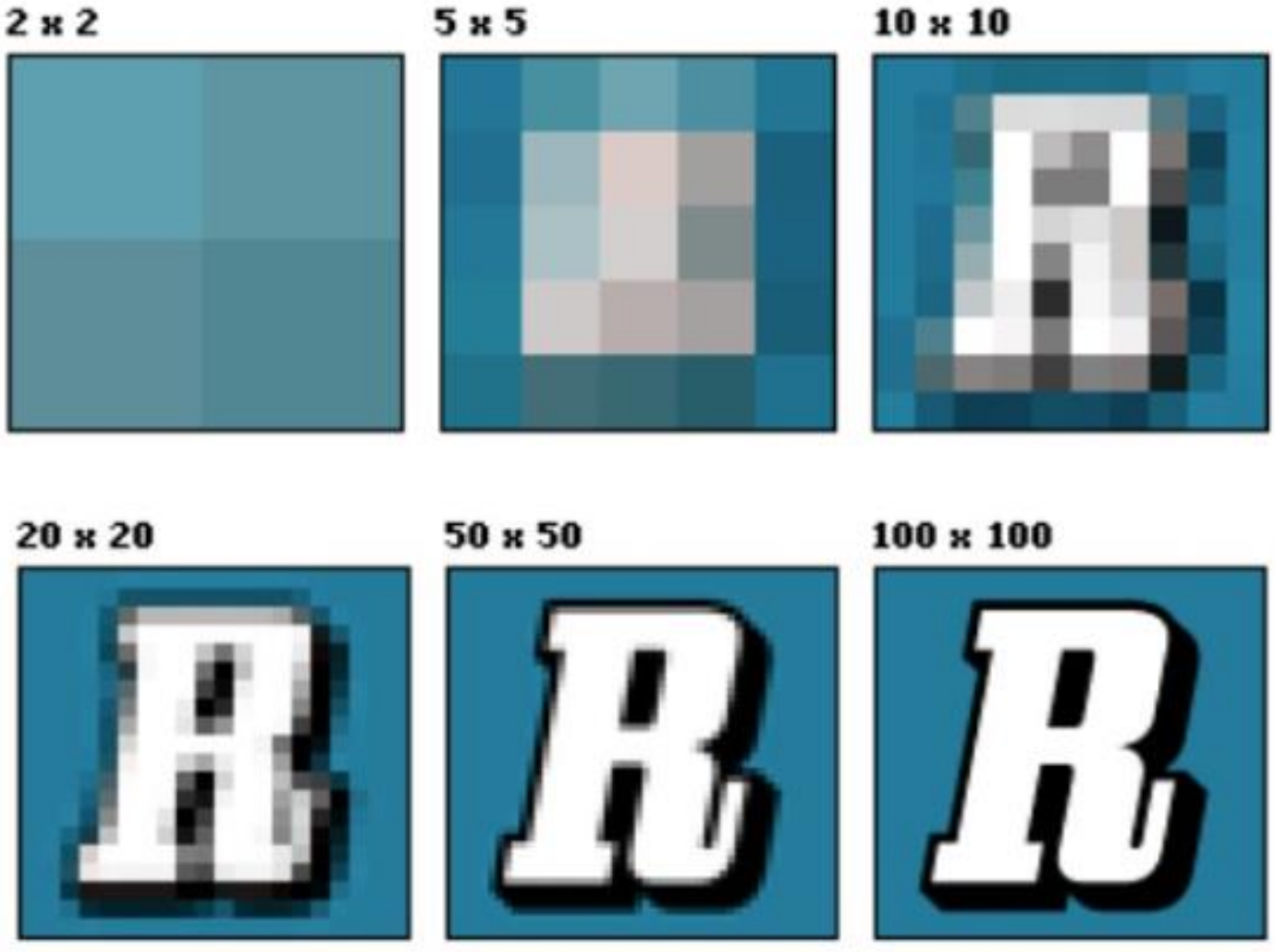
16 bits (65,536 colores o Highcolor), existen distintas combinaciones para el uso de estos 16 bits. Las más comunes son:

- 5 bits para rojo, 6 bits para verde y 5 para azul. Se utiliza esta distribución porque el ojo humano distingue más colores verdes que otros.
- 4 bits para rojo, 4 bits para verde, 4 para azul y 4 para transparencias. Este es un modelo más equitativo.

24 bits (16,777,216 colores o Truecolor), este es el modelo RGB puro. Se destinan 8 bits para cada color que cubren todo el espectro.

32 bits (16,777,216 colores), utiliza RGB más 8 bits destinados a canales alpha o transparencias.

Hardware y representación de imágenes



Dibujo de pixeles

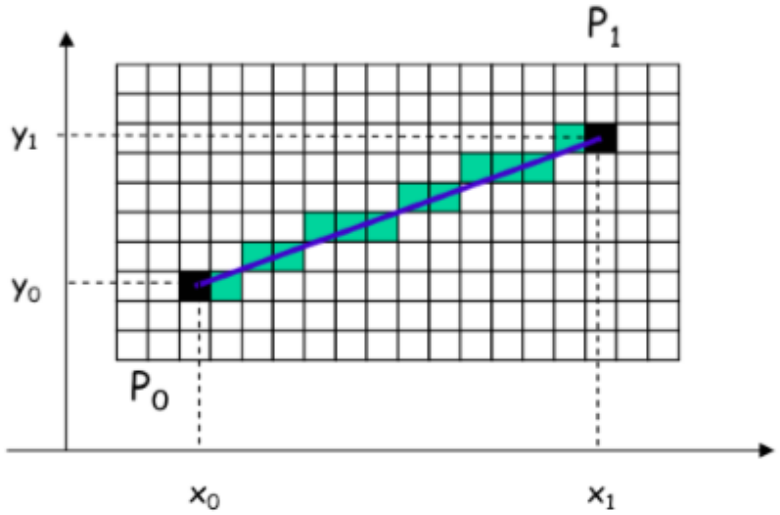
Existe un inconveniente, Java no cuenta con un método para dibujar un pixel y los métodos para dibujar figuras (drawLine, drawRect, drawArc, etc.) presentan algunos comportamientos extraños para trabajar con pixeles, por esa razón vamos a diseñar nuestra propio método para dibujar un pixel.

```
public class Ventana extends Jframe {
   private BufferedImage buffer;
   private Graphics graPixel;
   public Ventana() {
      ...
      buffer = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
      graPixel = (Graphics2D) buffer.createGraphics();
      ...
   }
   public void putPixel(int x, int y, Color c) {
      buffer.setRGB(0, 0, c.getRGB());
      this.getGraphics().drawImage(buffer, x, y, this);
   }
   ...
}
```

Dibujo de líneas y su comportamiento...

La ecuación de la recta es:

$$y = mx + b$$



Dibujo de líneas y su comportamiento...

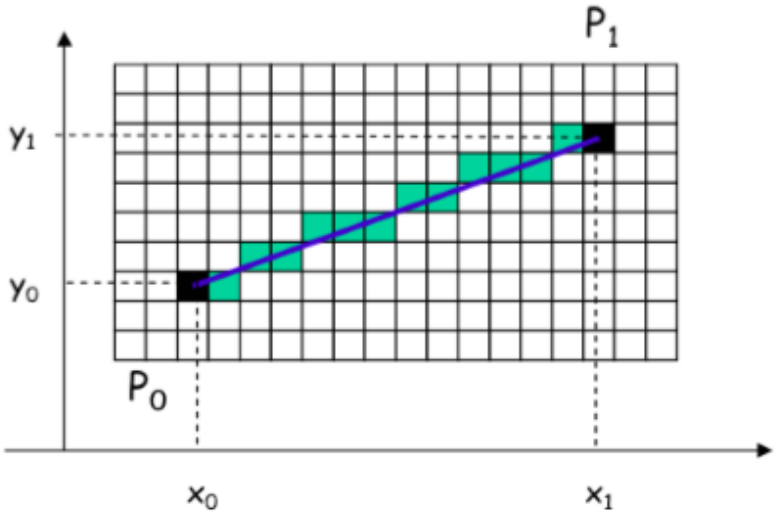
La ecuación de la recta es:

$$y = mx + b$$

Algoritmo:

Calcular $m = \frac{dy}{dx} = \frac{y1 - y0}{x1 - x0}$

Calcular $b = y0 - mx0$

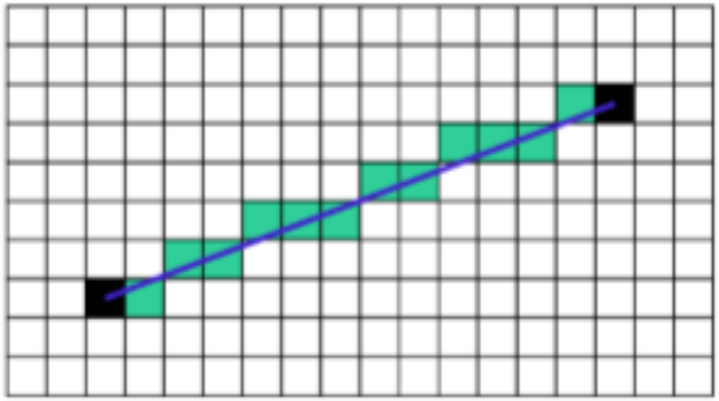



```
Ciclo x = x0 hasta x = x1
    y = mx + b
    Pintar pixel(x, round(y))
```

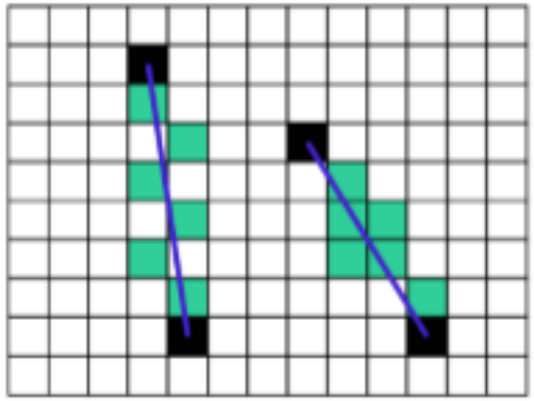
Dibujo de líneas y su comportamiento...

Deficiencias del algoritmo más sencillo para dibujar líneas rectas:

- No es muy eficiente para encontrar los pixeles correspondientes al trazo de la línea.
- Cada paso requiere una multiplicación flotante, una suma y un redondeo.



correcto



incorrecto

Dibujo de líneas y su comportamiento...

Algoritmo básico incremental (DDA)

Con éste algoritmo podemos eliminar la multiplicación, calculando cada pixel en función del anterior y no hace falta calcular b

$$y_i = mx_i + b$$

$$y_{i+1} = mx_{i+1} + b$$

Como eliminar ***b***?

Dibujo de líneas y su comportamiento...

Algoritmo básico incremental (DDA)

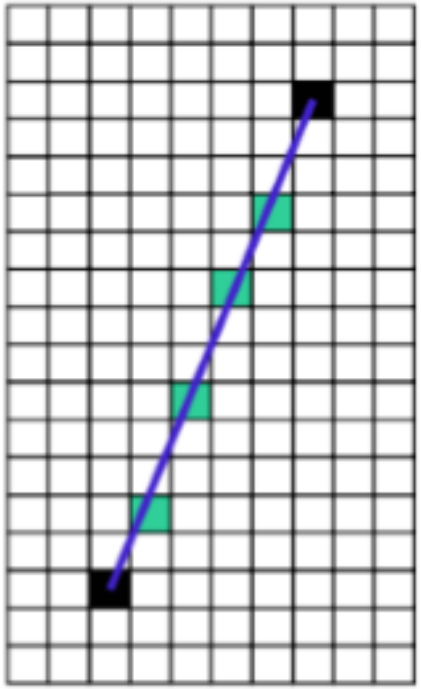
Con éste algoritmo podemos eliminar la multiplicación, calculando cada pixel en función del anterior y no hace falta calcular b

$$y_i = mx_i + b$$

$$y_{i+1} = mx_{i+1} + b$$

Como eliminar ***b***?

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



Si ***m*** > ***1*** falla

quedan huecos

Dibujo de líneas y su comportamiento...

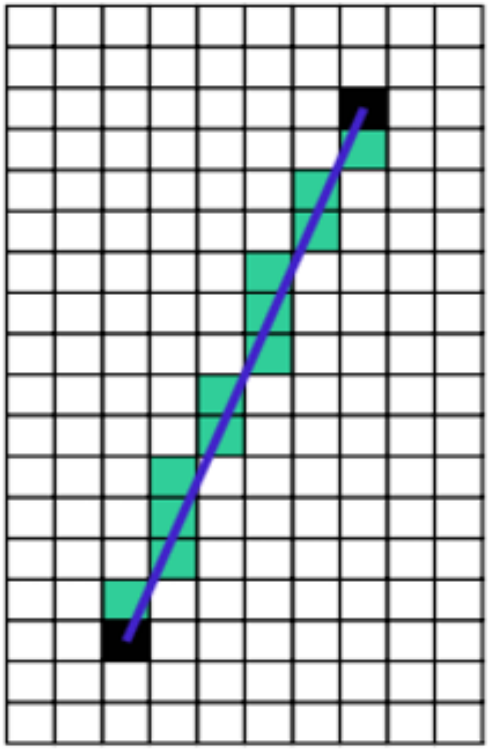
Algoritmo básico incremental (DDA) SOLUCIÓN

Intercambiamos las variables x e y

$$x_i = \frac{y_i - b}{m}$$

$$x_{i+1} = \frac{y_{i+1} - b}{m}$$

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



Dibujo de líneas y su comportamiento...

Algoritmo básico incremental (DDA)

```
Función Línea_DDA (int x0, y0, x1, y1)
   dx = x1 – x0
   dy = y1 – y0
   Si abs(dx) > abs(dy) entonces steps = abs(dx)
      Si no steps = abs(dy)
   xinc = dx / steps
   yinc = dy / steps
   x = x0
   y = y0
   Pintar Pixel(round(x), round(y))
   Para k = 1 hasta k = steps
      x = x + xinc
      y = y + yinc
      Pintar Pixel(round(x), round(y))
```

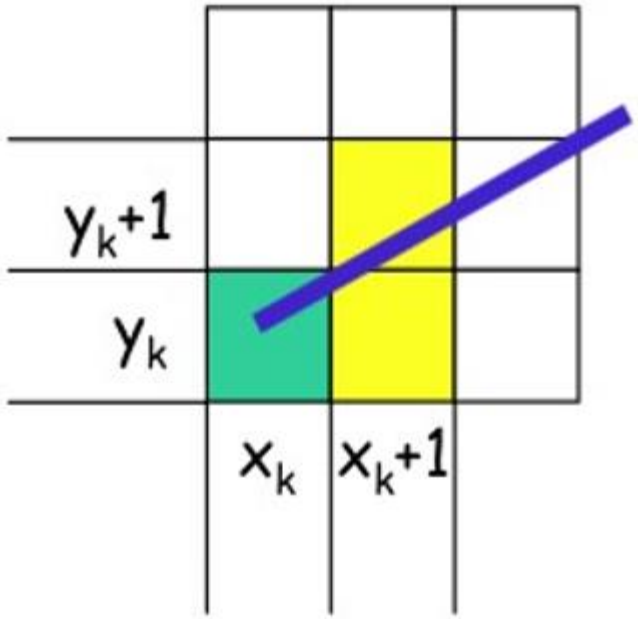
Inconvenientes:

- Existen errores de acumulación
- El redondeo es muy lento

Dibujo de líneas y su comportamiento...

Algoritmo de Bresenham

- Solo se usa aritmética entera.
- Supongamos el caso $0 < m < 1$, entonces hay que decidir qué pixel dibujamos a continuación, y solo hay dos opciones.



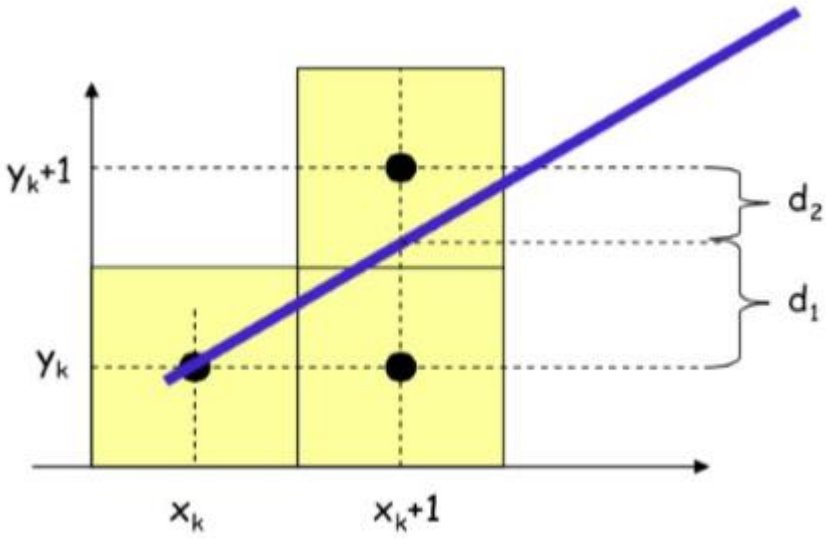
Dibujo de líneas y su comportamiento...

Algoritmo de Bresenham

Pixel: (x_k, y_k)

Opciones: $(x_k + 1, y_k)$

$(x_k + 1, y_k + 1)$



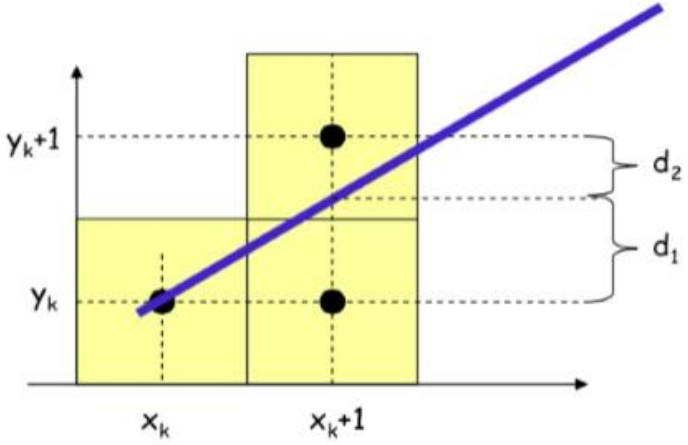
$$y = m(x_k + 1) + b$$

$$d_2 = (y_k + 1) - y = (y_k + 1) - m(x_k + 1) - b$$

$$d_1 = (y - y_k) = m(x_k + 1) + b - y_k$$

Dibujo de líneas y su comportamiento...

Algoritmo de Bresenham



La diferencia entre ambas constantes

nos ayudará a decidir qué pixel pintar

$$p_k = (d_1 - d_2) = 2m(x_k + 1) - 2y_k + 2b - 1$$

Multiplicando ***dx*** eliminamos ***m***

$$p_k = dx(d_1 - d_2) = 2dyx_k - 2dxy_k + C \qquad C = 2dy + dx(2b - 1)$$

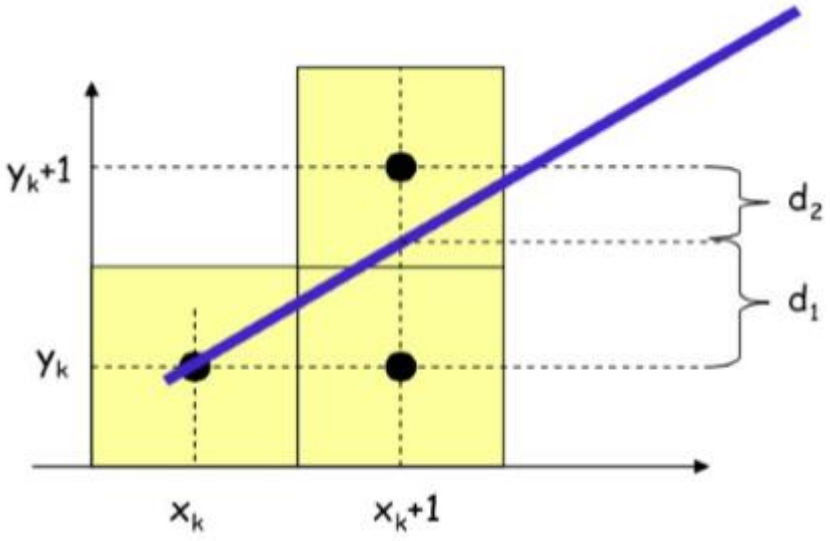
Si $p_k > 0 \rightarrow d_1 > d_2$ entonces tenemos que pintar el pixel $(x_k + 1, y_k + 1)$

Si $p_k < 0 \rightarrow d_1 < d_2$ entonces tenemos que pintar el pixel $(x_k + 1, y_k)$

Dibujo de líneas y su comportamiento...

Algoritmo de Bresenham

La gran ventaja es que puede calcularse la diferencia del siguiente punto a partir del anterior, utilizando solamente aritmética entera



$$p_{k+1} = \ldots = p_k + 2dy - 2dx\left(y_{k+1} - y_k\right)$$

0 ó 1 dependiendo del signo de ***pk***

Dibujo de líneas y su comportamiento...

Algoritmo de Bresenham

```
Función Bresenham (int x0, y0, x1, y1)
   // sólo para el caso 0 < m < 1, siendo x0 < x1
   Pintar Pixel (x0, y0)
   Calcular las constantes A = 2dy, B = 2dy-2dx
   Obtener el valor para p0 = 2dy-dx
   Para cada xk sobre la línea
   Si pk < 0
      Pintar Pixel (xk + 1, yk)
      Pk + 1 = pk + A
   Si pk > 0
      Pintar Pixel (xk + 1, yk + 1)
      Pk + 1 = pk + B
```

- Si m > 1, intercambiamos las variables x e y
- Si m < 0, el cambio es similar

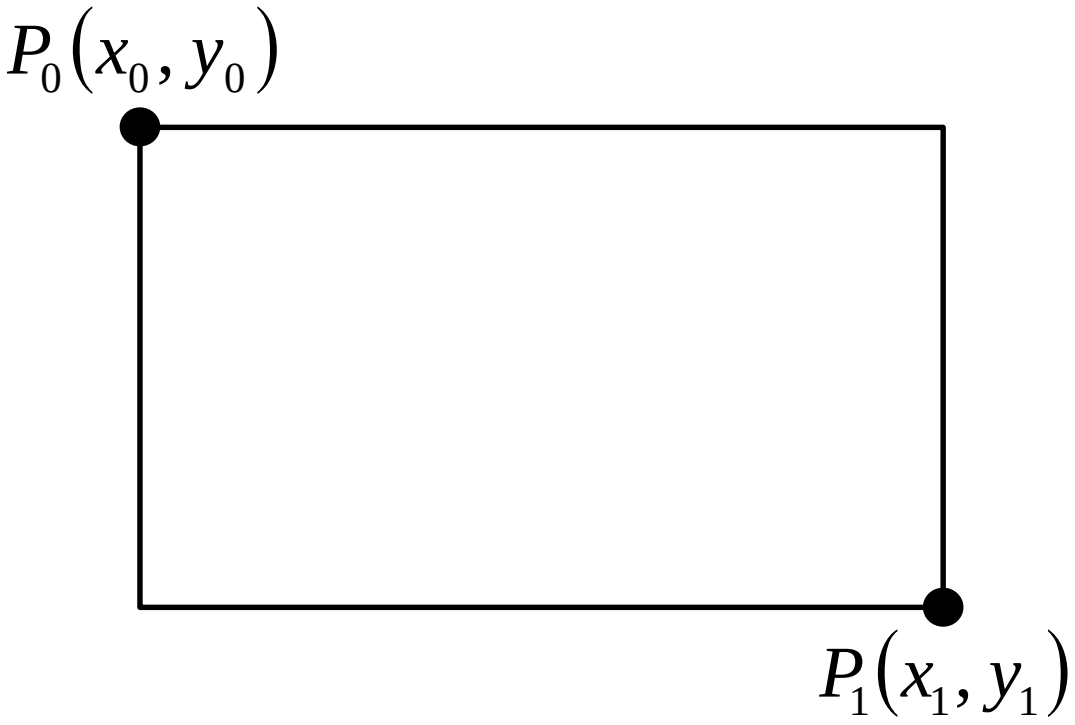
Dibujo de líneas y su comportamiento...

Algoritmo de Punto Medio

Del algoritmo de Bresenham, implementa el algoritmo de punto medio...

Dibujo de cuadros y rectángulos

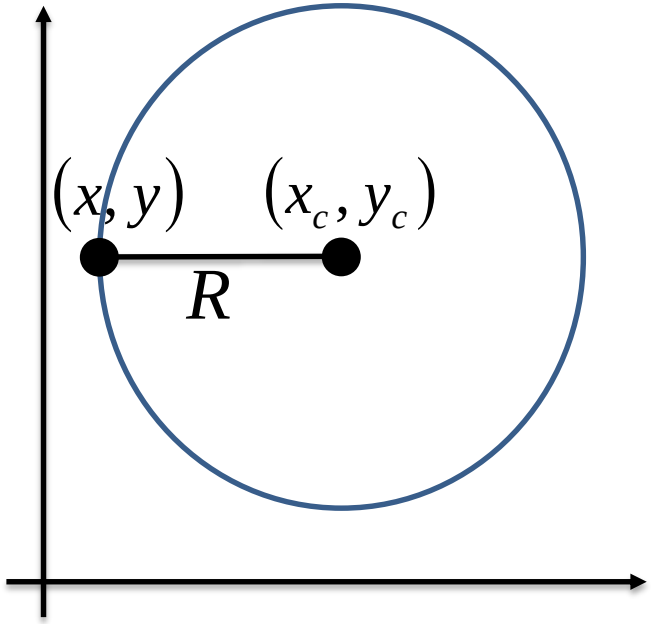
Para dibujar rectángulos se deben considerar dos coordenadas y dibujar las 4 líneas correspondientes.



Dibujo de círculos y elipses...

La ecuación del un círculo de radio $\boldsymbol{R}$ centrado en las coordenadas (x_c, y_c)

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



Dibujo de círculos y elipses...

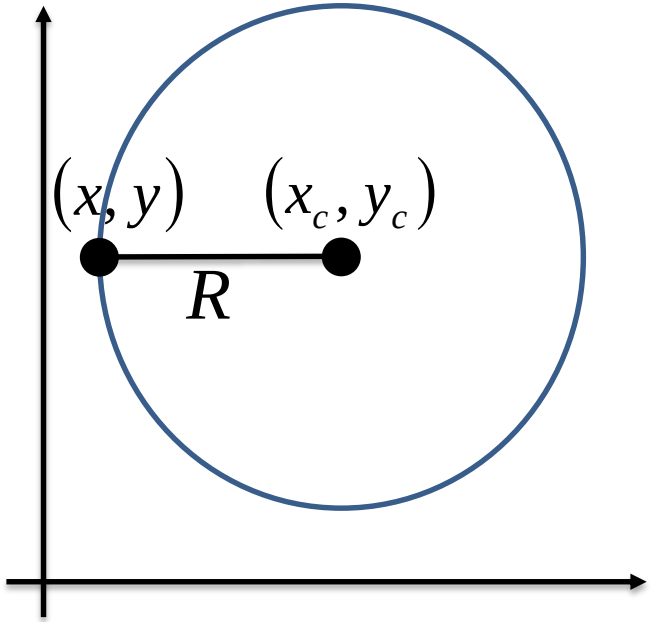
La ecuación del un círculo de radio R centrado en las coordenadas (x_c, y_c)

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

Algoritmo :

```
Para x=xc-R hasta x=xc+R
    Calcular  y = yc ± sqrt(R^2 - (x - xc)^2)
    Pintar pixel(x, round(y))
```

- No es nada eficiente
- Cada paso requiere una raíz cuadrada
- El espacio entre pixeles no es uniforme



Dibujo de círculos y elipses...

Otra forma...

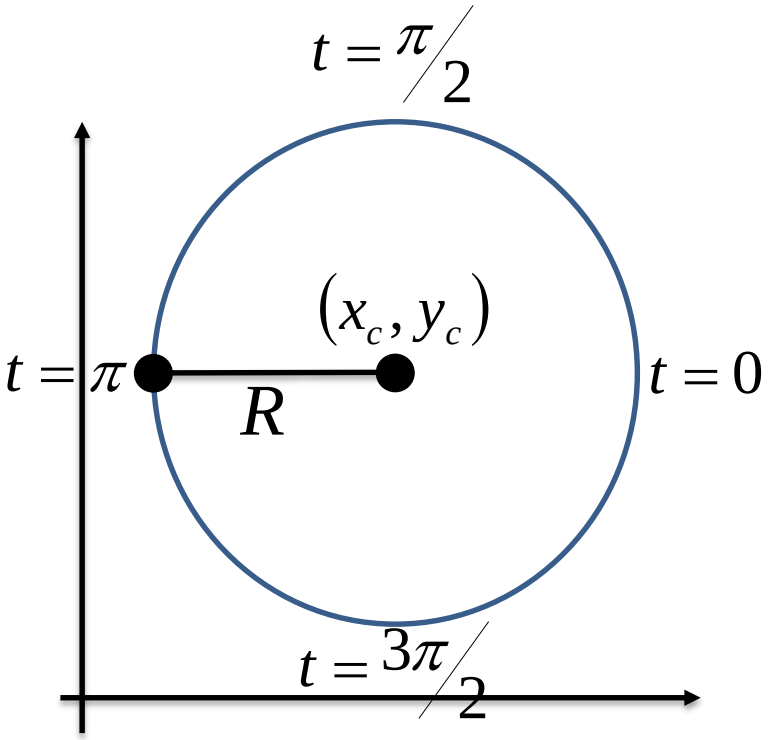
¿Cómo solucionar lo de los agujeritos?

Pasando a coordenadas polares.

El valor del incremento del ángulo ***t***

debe ser lo suficientemente pequeño

para evitar los huecos.



Dibujo de círculos y elipses...

Otra forma...

¿Cómo solucionar lo de los agujeritos?

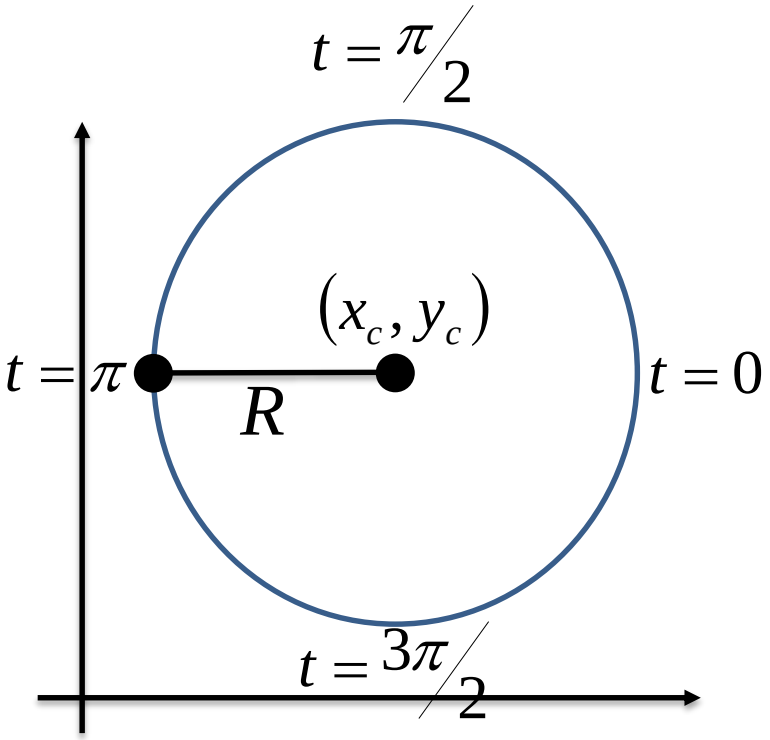
Pasando a coordenadas polares.

El valor del incremento del ángulo ***t***

debe ser lo suficientemente pequeño

para evitar los huecos.

$$x = x_c + R\sin(t)$$
$$y = y_c + R\cos(t)$$

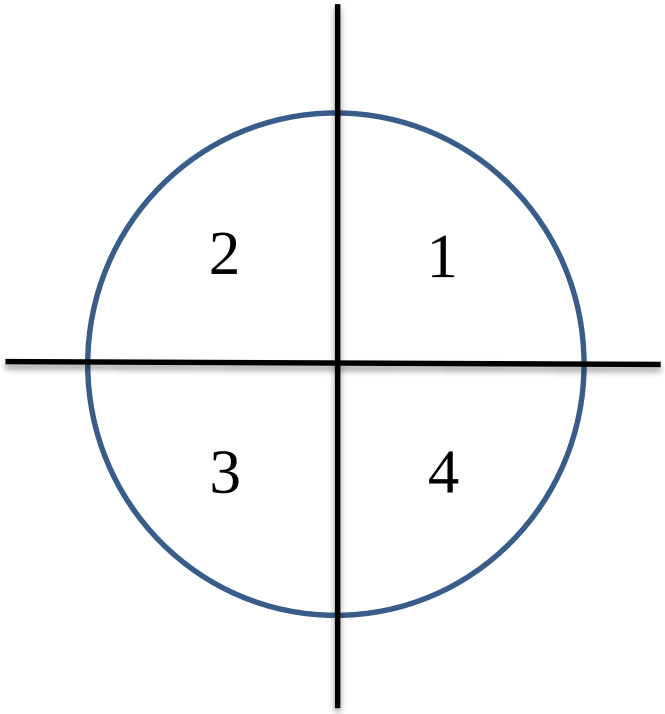


Dibujo de círculos y elipses...

Podemos reducir el cálculo aplicando **simetrías de ocho lados**

Incluso el primer octante es simétrico al segundo a través de una diagonal

Dibujando solo el segundo octante, desde x=0 hasta x=y podemos pintar todo el circulo



Se necesitan raíces cuadradas y funciones trigonométricas, es demasiado costoso

Dibujo de círculos y elipses...

Algoritmo del punto medio

- Hay que determinar el pixel más cercano a la circunferencia
- Consideramos el centro en (0,0)
- Comenzamos en el punto (0,R) e iremos desde x=0 hasta x=y, donde la pendiente va de 0 a -1
- Solo pintaremos el primer octante

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

< 0 entonces (x,y) está dentro

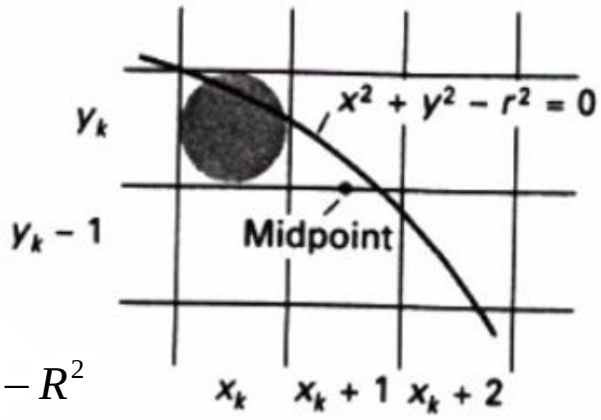
= 0 entonces (x,y) está sobre la circunferencia

> 0 entonces (x,y) está fuera

Dibujo de círculos y elipses...

Algoritmo del punto medio

Supongamos que dibujamos el pixel (x_k, y_k)



$$P_k = f(x_k + 1, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2}) - R^2$$

$$P_{k+1} = f(x_{k+1} + 1, y_{k+1} - \frac{1}{2}) = (x_{k+1} + 1)^2 + (y_{k+1} - \frac{1}{2}) - R^2$$

...

$$P_{k+1} = P_k + 2x_{k+1} + 1 + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$$

Si $P_k < 0$ entonces $P_{k+1} = P_k + 2x_{k+1} + 1$

hay que pintar el pixel $(x_k + 1, y_k)$

Si $P_k < 0$ entonces $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

hay que pintar el pixel $(x_k + 1, y_k - 1)$

Dibujo de círculos y elipses...

Algoritmo del punto medio

Empezamos en el punto (0,R) $P_0 = f(1, R - \frac{1}{2}) = \ldots = \frac{5}{4} - R$

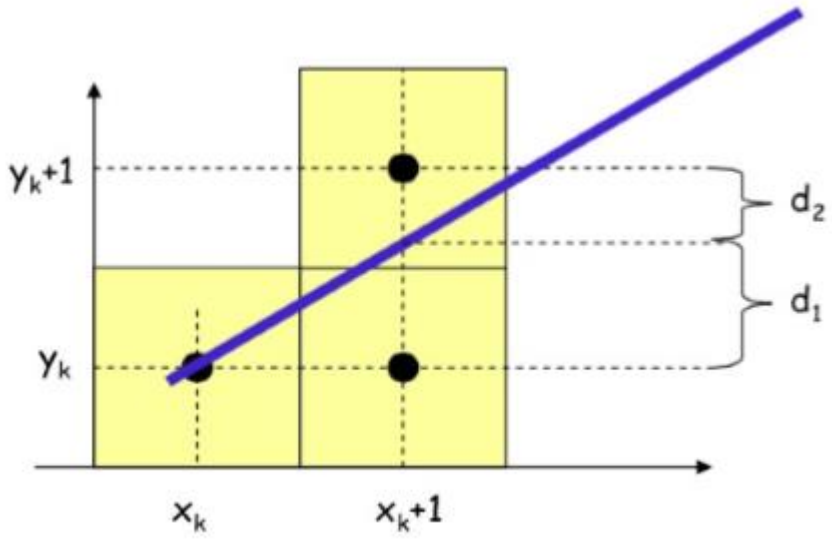
No es entero, sin embargo, da igual. Podemos redondearlo, porque todos los incrementos son enteros, y solo utilizaremos el signo de Pk, y no su valor.

```
Función PuntoMedio(int xc, yc, float R)
Pintar Pixel (0, R)
Calcular p0 = 5/4 – R //si R es entero, p0 = 1 - r
Para cada xk
   Si pk < 0
      Pintar Pixel (xk + 1, yk)
      Pk + 1 = pk + 2xk + 3
   Si pk > 0
      Pintar Pixel (xk + 1, yk – 1)
      Pk + 1 = pk + 2xk – 2yk + 5
Determinar por simetría los puntos de los otros 7
octantes Pintar Pixel (x+xc, y+yc)
```

Dibujo de círculos y elipses...

Algoritmo Bresenham

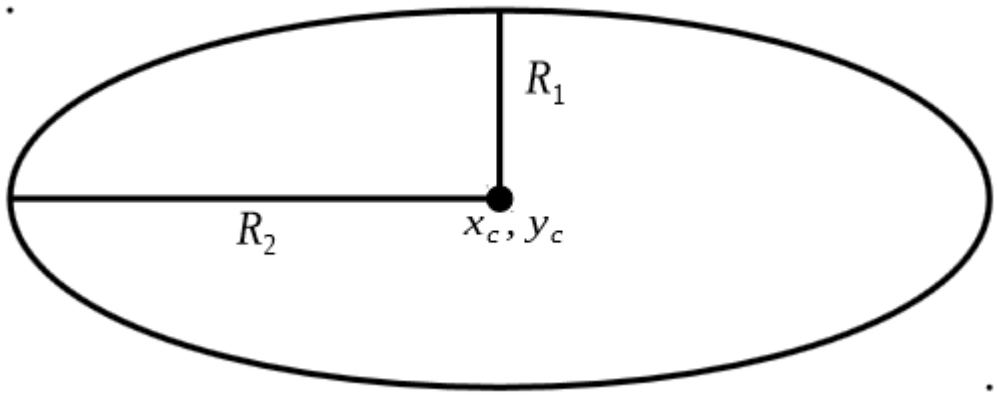
Este algoritmo también se utiliza para circunferencias, el principio es el mismo, ¿como lo implementarías?



Dibujo de círculos y elipses...

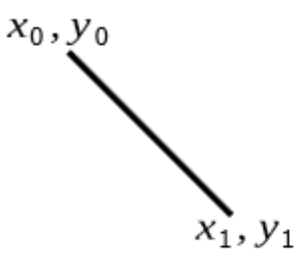
¿Cómo dibujarías un elipse?

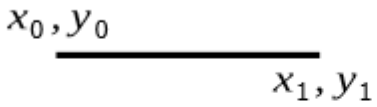
Centrado en las coordenadas (x0, y0)

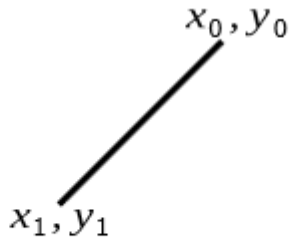


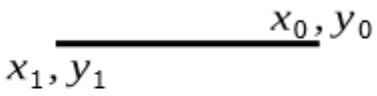
Figuras básicas

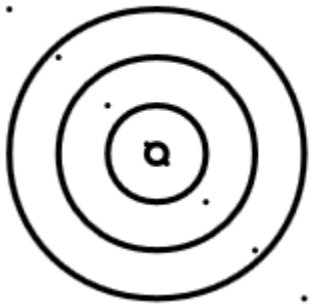
En un JFrame, dibuja las siguientes figuras con tus métodos propios

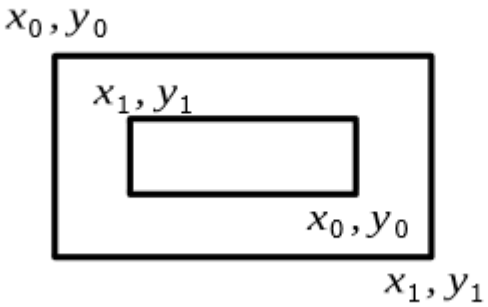


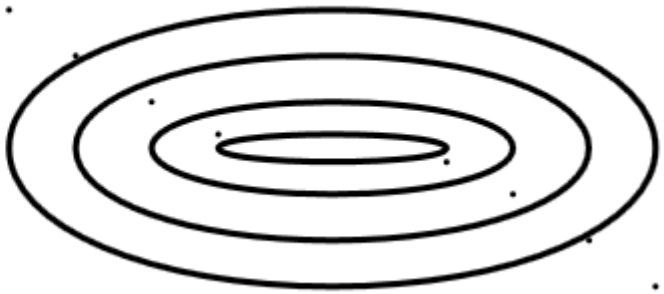












Figuras básicas

Tipos de línea (líneas rectas)

Existen varios tipos: continua, discontinua y con tipos.

Los procedimientos para dibujar éstas líneas van mostrando secciones contiguas de pixeles, y luego se van saltando otros.

Las secciones de pixeles se especifican mediante una mascara, 1111000 se pintan 4 pixeles y se saltan 3

Al fijar el número de pixeles, las longitudes son diferentes según la dirección, por lo que es mejor ajustar el número de pixeles dependiendo de la pendiente.

Otra forma para dibujar líneas discontinuas sería tratar cada tramo como una línea individual.

Figuras básicas

Grosor de línea (líneas rectas)

Si la pendiente es menor que 1, para cada posición de x pintamos en una sección vertical en pixeles, tantos como ancho de línea queramos, por igual de cada lado.

Si la pendiente es mayor que 1, se usan secciones horizontales.

Hay que tener en cuenta que el ancho de las líneas horizontales y verticales será 2 veces más grueso que las diagonales.

Figuras básicas

Tipos de línea (circunferencias)

Para dibujar líneas discontinuas usaremos máscaras como en las líneas rectas.

Al copiar al resto de octantes hay que tener en cuenta la secuencia del interespaciado.

Las longitudes varían con la pendiente.

Figuras básicas

Grosor de línea (circunferencias)

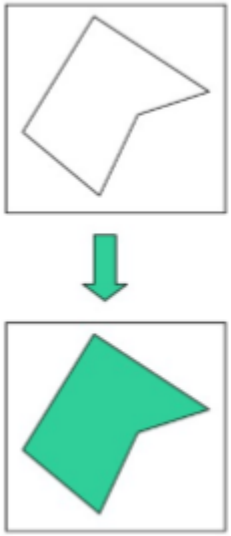
1. Pintado de secciones horizontales o verticales según sea la pendiente mayor o menor que 1.
2. Rellenar el espacio entre dos curvas paralelas, separadas por una distancia igual al ancho que queremos.
3. Usar una brocha (virtual) e irla moviendo a lo largo de la curva.

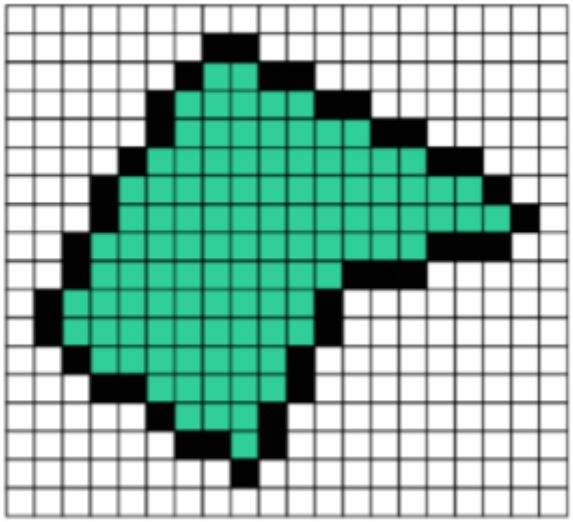
1	1	1
1	1	1
1	1	1

Relleno de primitivas

Principio teórico

Dada una área cerrada, hay que ser capaz de rellenar los pixeles interiores con un color o patrón (imagen) determinada.



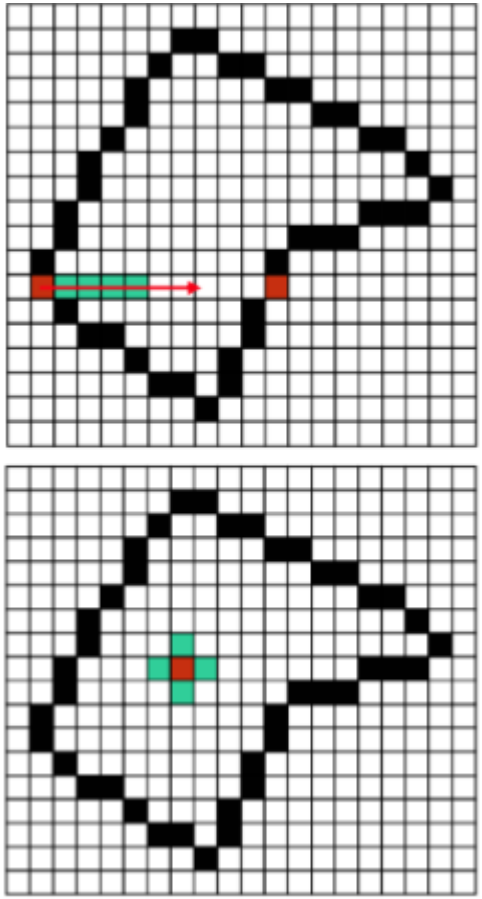


Relleno de primitivas

Existen 2 métodos generales

1. Relleno por scan-line: fila a fila va trazando líneas de color entre aristas.

2. Relleno por inundación: a partir de un punto central, se ***ca*** expandiendo recursivamente hasta alcanzar el borde del objeto.



Relleno de primitivas

- **Investigar el algoritmo optimizado para el relleno scan-line**
- **Investigar la optimización del relleno por inundación implementando scan-line**

Recorte de líneas

Algoritmo de recorte por puntos

$$x_{min} \leqslant x \leqslant x_{max}$$

$$y_{min} \leqslant y \leqslant y_{max}$$

Recorte de líneas

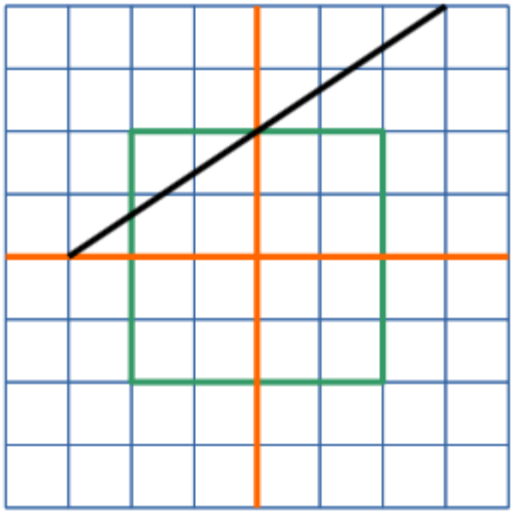
Algoritmo de recorte explícito en 2D

Consideremos una línea recta (x1,y1) a (x2,y2)

$$y = mx + b$$

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

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

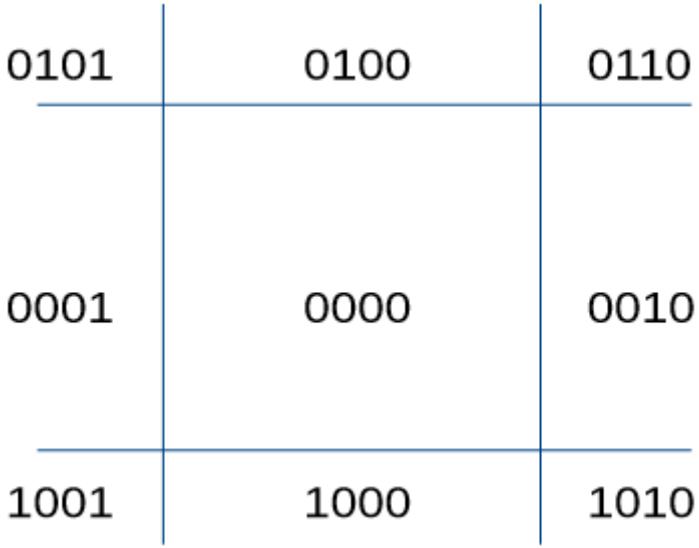


$y = y_{\text{sup}}$ $\quad x = x_{der}$

$y = y_{\text{inf}}$ $\quad x = x_{izq}$

Recorte de líneas

Algoritmo de recorte por códigos de región



Bit 0: Izquierda

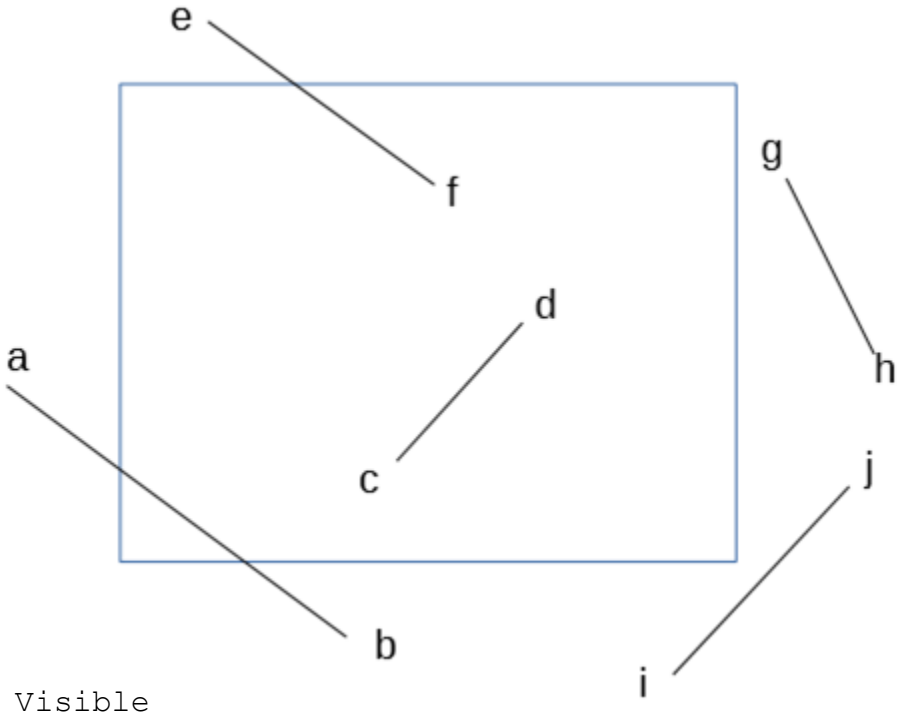
Bit 1: Derecha

Bit 2: Arriba

Bit 3: Abajo

Recorte de líneas

Algoritmo de recorte por códigos de región



	Cod1	Cod2	AND	Observación
ab	0001	1000	0000	Parcialmente Visible
cd	0000	0000	0000	Totalmente Visible
ef	0100	0000	0000	Parcialmente Visible
gh	0010	0010	0010	Trivialmente Invisible
ij	1000	0010	0000	Totalmente Invisible

Recorte de líneas

Algoritmo de recorte Sutherland-Cohen

Por cada extremo de la ventana rectangular de recorte efectuar los pasos (a), (b) y (c)

(a) Para la línea P1-P2, determine si la línea es totalmente visible o si puede ser descartada como trivialmente invisible.

(b) Si P1 está fuera de la ventana de recorte continua, de lo contrario intercambia P1 y P2.

(c) Reemplaza P1 por la intersección de P1-P2 con el extremo en turno

Recorte de líneas

Investigar algoritmo de recorte para regiones convexas.

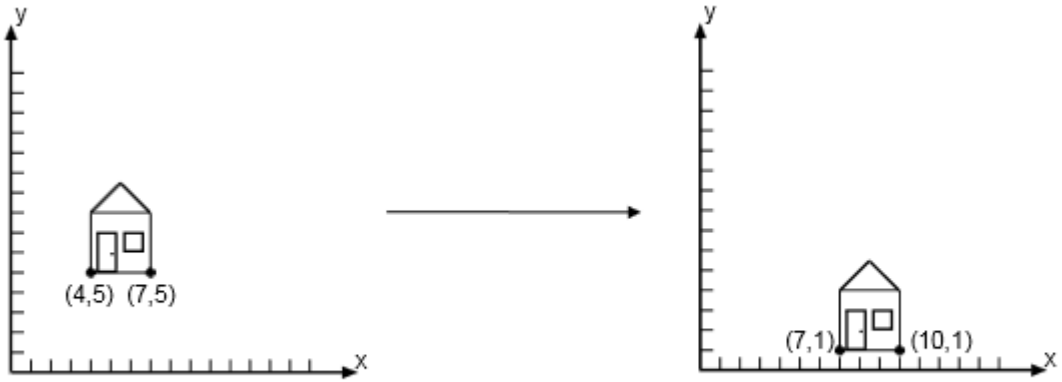
Y alguna propuesta para recorte de circunferencias.

Transformaciones afines bidimensionales

Traslación

Se traslada cada punto ***P(x,y)*** ***dx*** unidades paralelamente al eje ***x*** y ***dy*** unidades paralelamente al eje ***y***, hacia el nuevo punto ***P'(x',y')***.

$$x' = x + d_x \qquad y' = y + d_y$$



Traslación de un objeto

Transformaciones afines bidimensionales

Traslación

Se traslada cada punto ***P(x,y)*** ***dx*** unidades paralelamente al eje ***x*** y ***dy*** unidades paralelamente al eje ***y***, hacia el nuevo punto ***P'(x',y')***.

$$x' = x + d_x \qquad y' = y + d_y$$

Si se definen los vectores columna queda:

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad T = \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

Expresándose como **Ecuación General de la Traslación:**

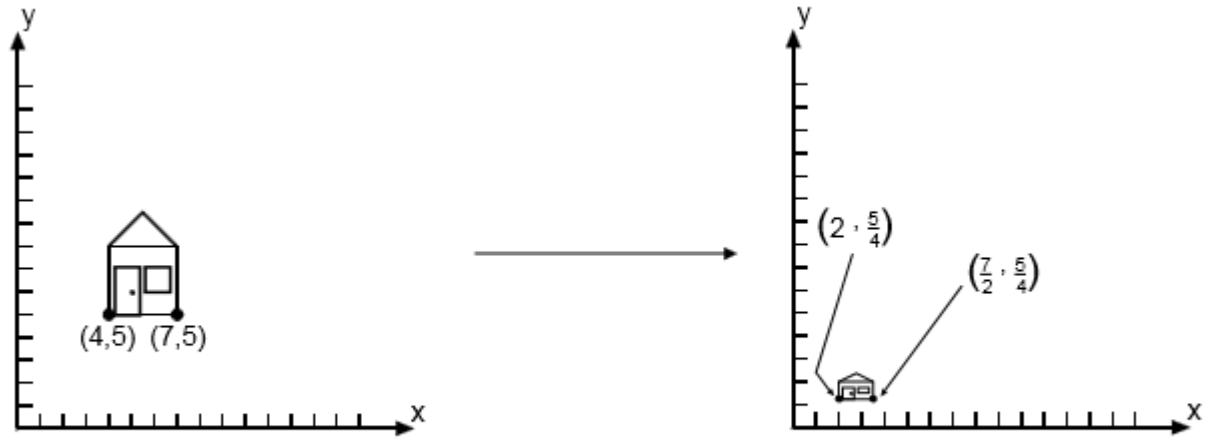
$$P' = P + T$$

Transformaciones afines bidimensionales

Escalamiento

El escalamiento se hace con un factor ***Sx*** en el eje ***x*** y en un factor ***Sy*** en el eje ***y***. Escalamiento uniforme ***Sx*** = ***Sy***, escalamiento diferencial:

$$x' = s_x \cdot x \qquad y' = s_y \cdot y$$



Antes del escalamiento

Después del escalamiento

Transformaciones afines bidimensionales

Escalamiento

El escalamiento se hace con un factor ***Sx*** en el eje ***x*** y en un factor ***Sy*** en el eje ***y***.

$$x' = s_x \cdot x \qquad y' = s_y \cdot y$$

En forma matricial

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

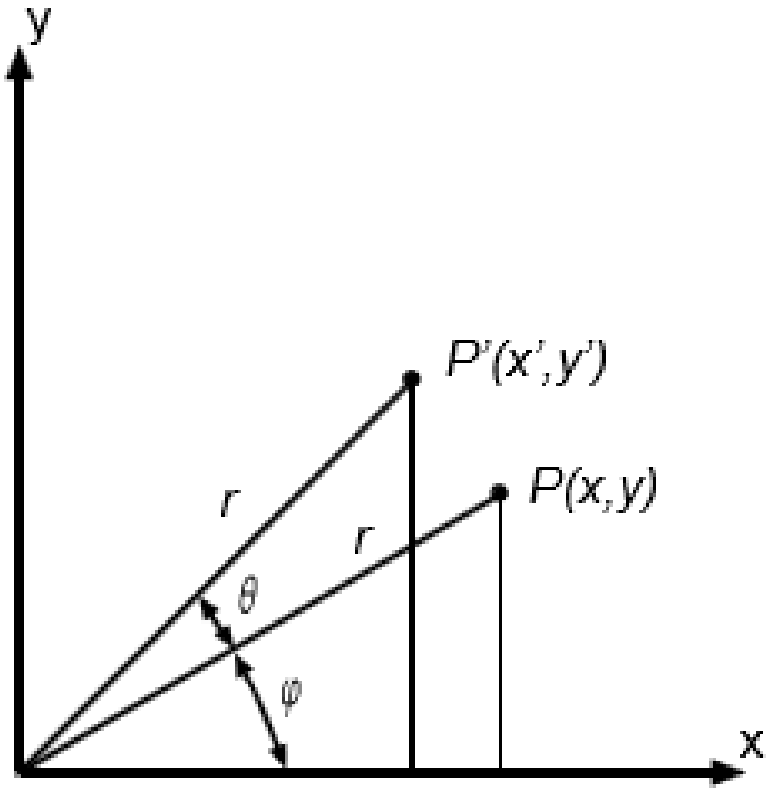
Expresándose como **Ecuación General del Escalamiento:**

$$P' = P \cdot S$$

Transformaciones afines bidimensionales

Rotación

Los puntos también pueden ser rotados un ángulo θ con respecto al origen



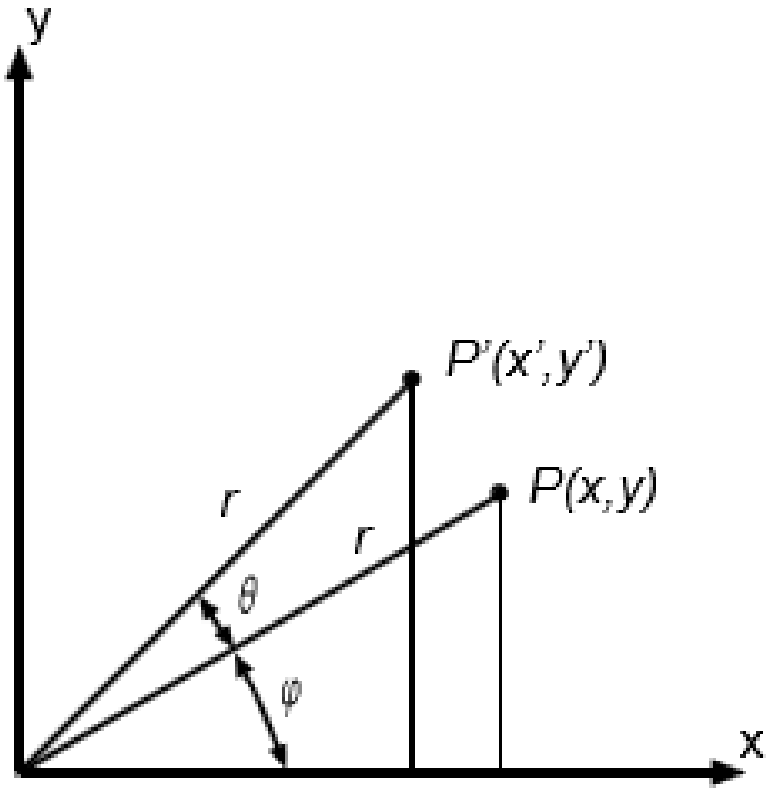
Transformaciones afines bidimensionales

Rotación

Los puntos también pueden ser rotados un ángulo θ con respecto al origen

$$x = r\cos(\varphi)$$
$$y = r\sin(\varphi)$$

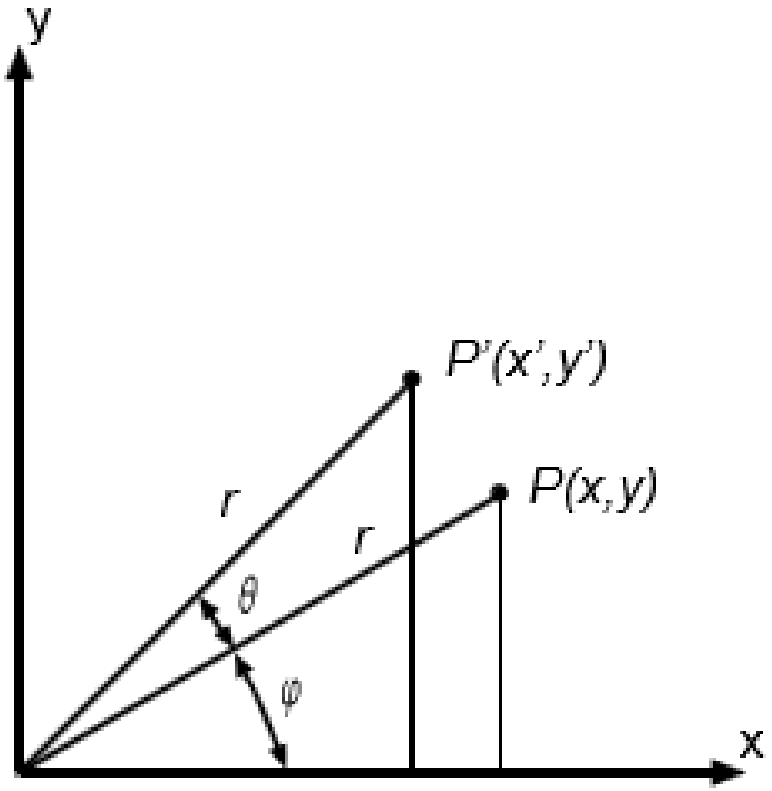
$$x' = r\cos(\theta + \varphi)$$
$$y' = r\sin(\theta + \varphi)$$



Transformaciones afines bidimensionales

Rotación

Los puntos también pueden ser rotados un ángulo θ con respecto al origen



$$x = r\cos(\varphi)$$
$$y = r\sin(\varphi)$$

$$x' = r\cos(\theta+\varphi)$$
$$y' = r\sin(\theta+\varphi)$$

$$x' = r\cos(\varphi)\cos(\theta) - r\sin(\varphi)\sin(\theta)$$
$$y' = r\cos(\varphi)\sin(\theta) + r\sin(\varphi)\cos(\theta)$$

Transformaciones afines bidimensionales

Rotación

Los puntos también pueden ser rotados un ángulo θ con respecto al origen

$$x' = x\cos(\theta) - y\sin(\theta) \qquad y' = x\sin(\theta) + y\cos(\theta)$$

En forma matricial

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \qquad S = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Expresándose como **Ecuación General de la Rotación:**

$$P' = P \cdot R$$

Transformaciones afines bidimensionales

Problemática de las transformaciones afines

La traslación es tratada de forma diferente…

Ejemplos aplicando varias transformaciones

$$P' = (P \cdot R) + T$$

$$P' = T + (P \cdot S \cdot R)$$

$$P' = ((P + T) \cdot R) + T$$

$$P' = (((P + T) \cdot R) + T) \cdot S$$

Coordenadas homogéneas

En las coordenadas homogéneas cada punto se representa siguiendo la forma ***(x,y,W)***.

Para $W \neq 0$ se obtiene los puntos $x / W, y / W$ a los cuales se les llama "coordenadas cartesianas del punto homogéneo".

Las ecuaciones pueden expresarse como una matriz 3x3 en coordenadas homogéneas.

$$P' = P \cdot T$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot S$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot R$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$