Sección de Informática Gráfica | Computer Graphics Group
VALENCIA

UNIVERSIDAD POLITECNICA DE VALENCIA

Primitivas Gráficas

Primitivas

Atributos

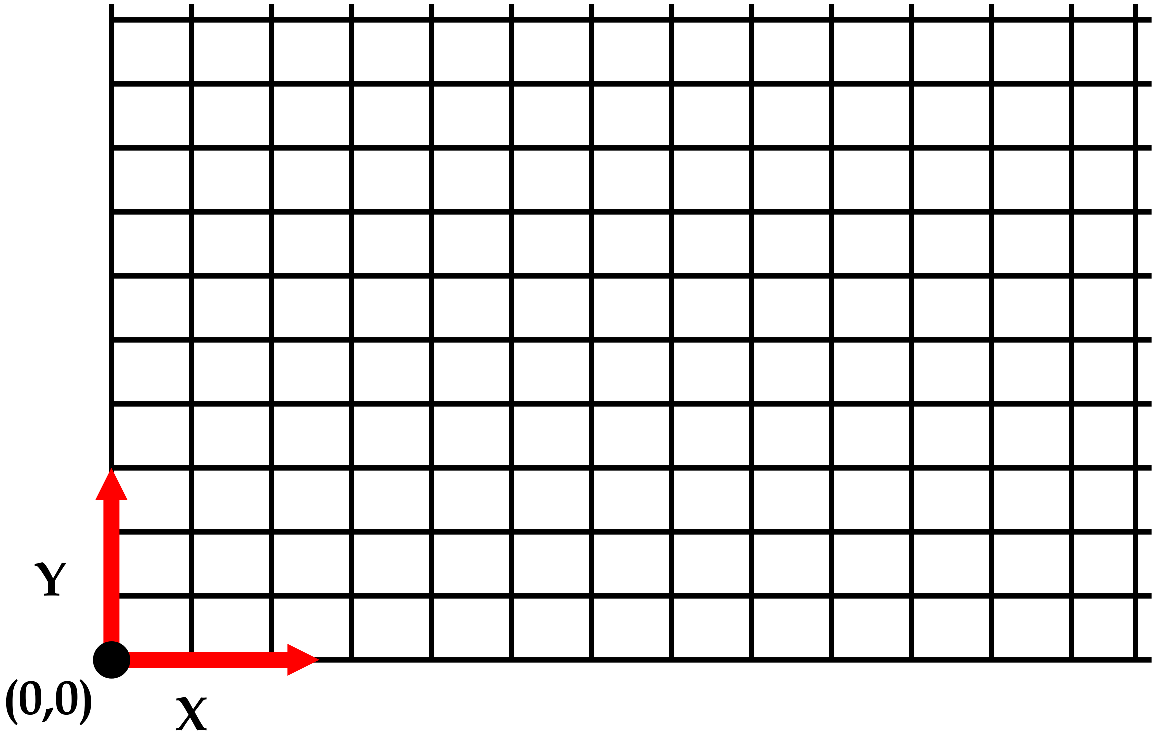
Algoritmos

Primitivas Gráficas

- Una librería gráfica (Computer Graphics Application Programming Interface CG API) proporciona las funciones necesarias para dibujar
  - Las primitivas permiten describir la forma de los componentes de la escena: Geometría
  - Puntos, Líneas, Círculos, Cónicas, Superficies Cuádricas, Curvas y Superficies Splines, Áreas y Polígonos.
  - El aspecto de estas primitivas se define mediante sus atributos.

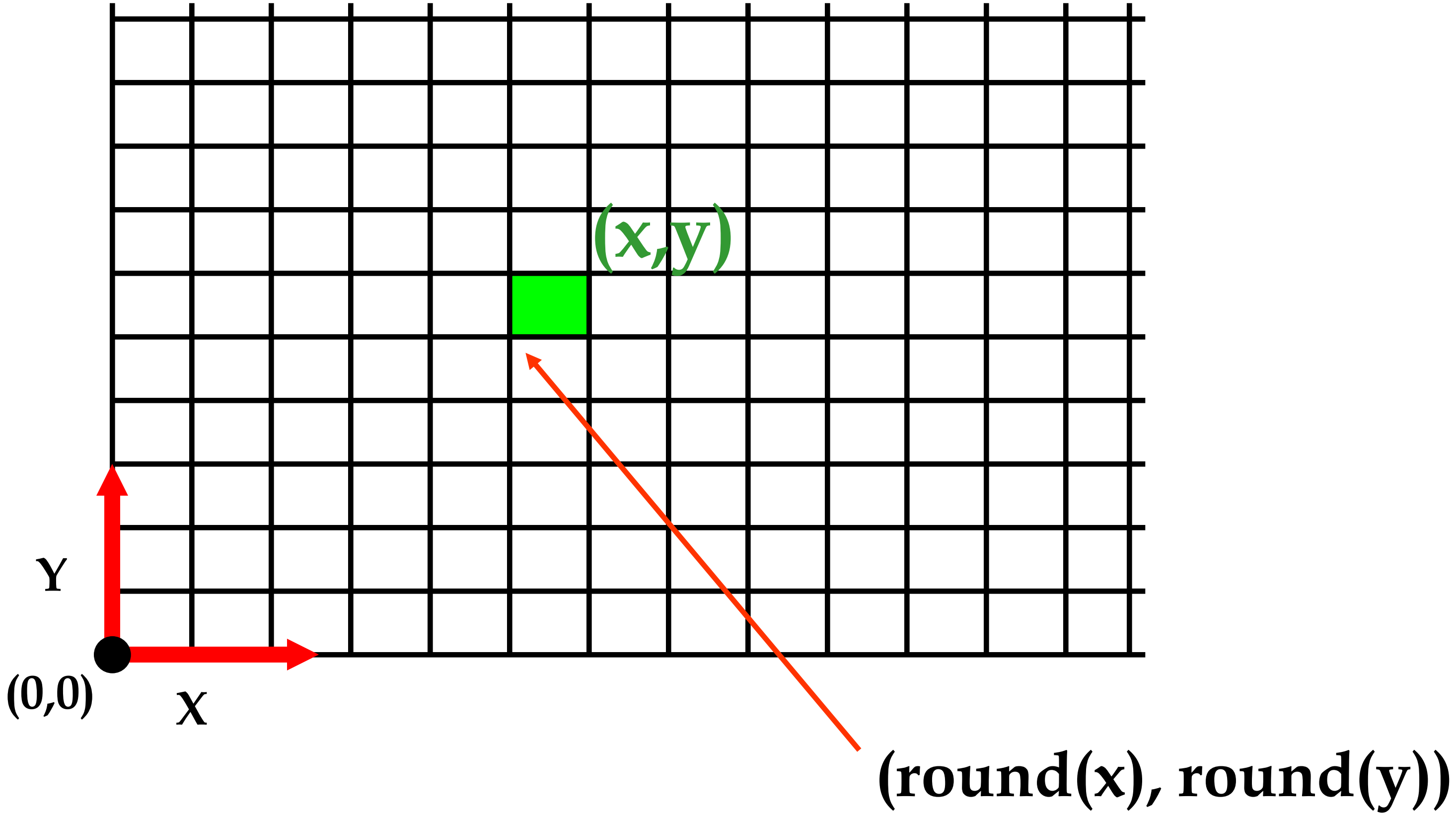
Primitivas Gráficas

- Todas las primitivas se referencian utilizando un sistema de coordenadas
  - Por ejemplo, una línea se define por sus dos puntos extremos
  - En un monitor se utilizan las coordenadas de pantalla



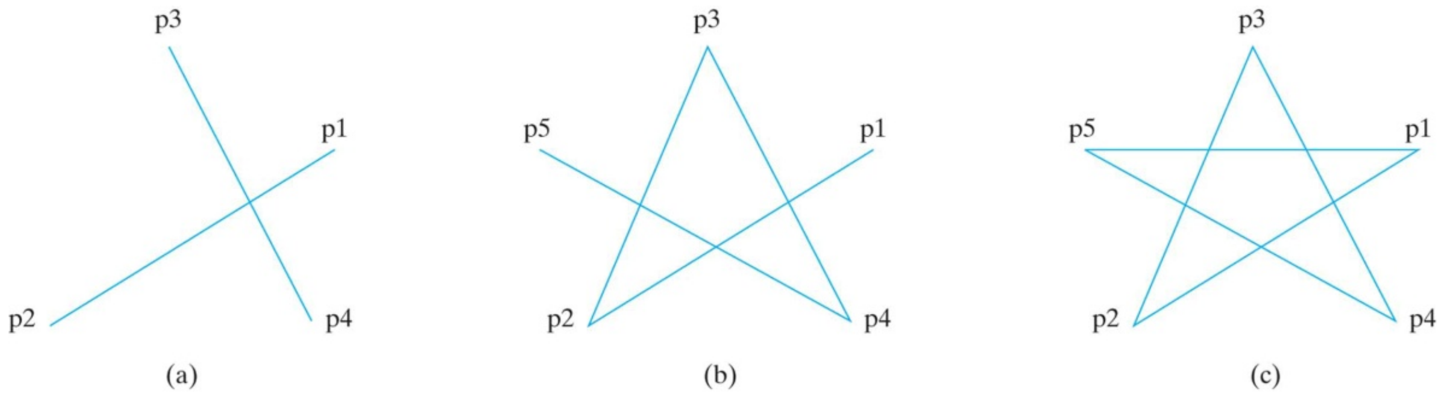
Primitivas Gráficas

El píxel se referencia por su esquina inferior izquierda

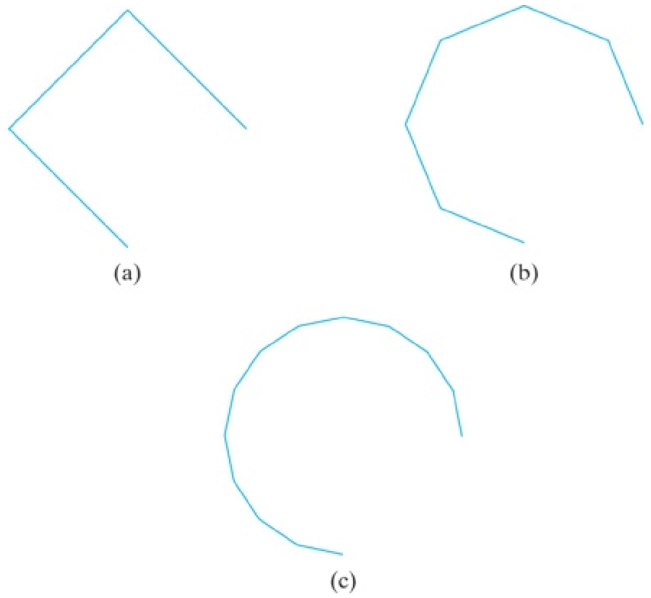


Primitivas Gráficas

- Líneas: (a) conjunto líneas, (b) polilínea, (c) polilínea cerrada
- Curvas: generalmente las librerías gráficas aproximan las curvas mediante polilíneas

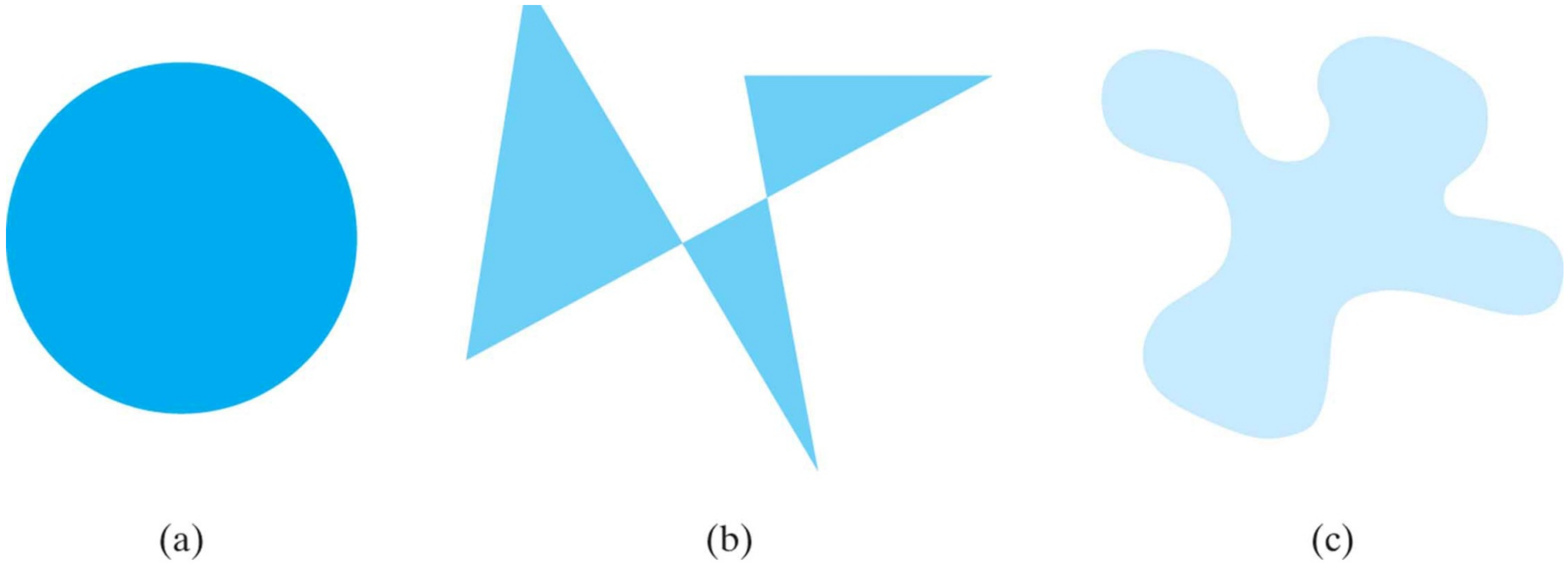


Copyright ©2011 Pearson Education, publishing as Prentice Hall



Primitivas Gráficas

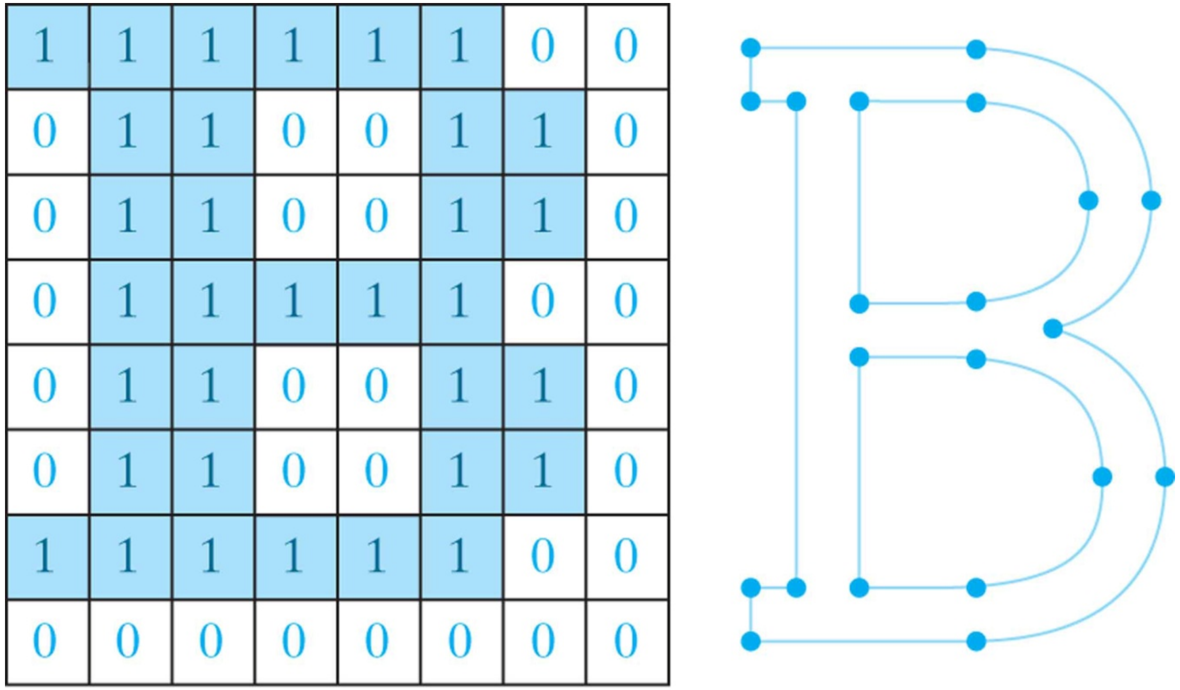
- Áreas rellenas: conjuntos de píxeles conectados con un mismo color o patrón
  - Se utilizan para definir superficies y objetos sólidos
  - Se pueden especificar mediante polígonos



Copyright ©2011 Pearson Education, publishing as Prentice Hall

Primitivas Gráficas

- Texto: (a) mediante patrón bitmap (b) vectorial



(a) (b)

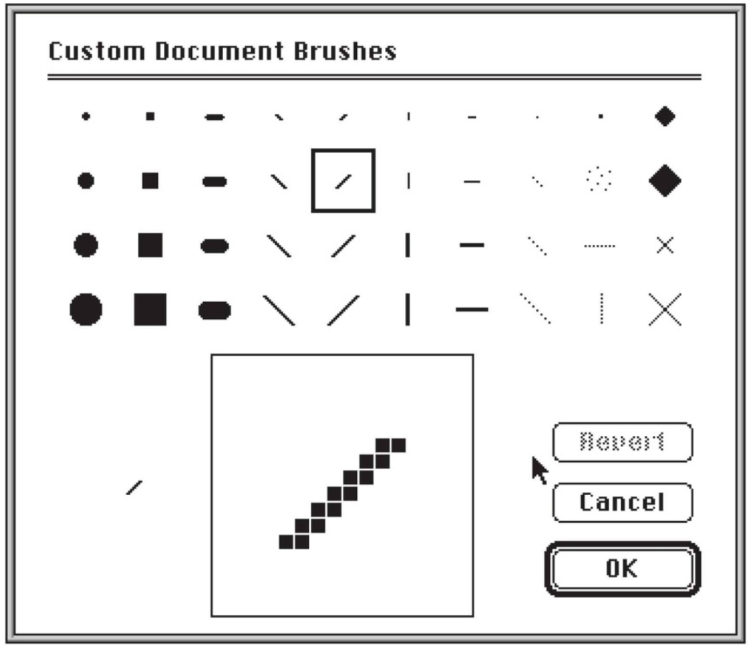
Copyright ©2011 Pearson Education, publishing as Prentice Hall

Atributos

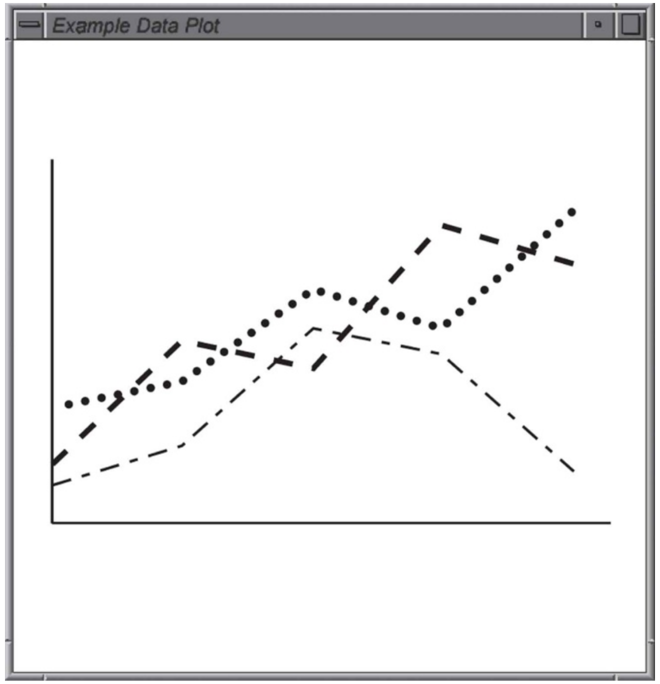
- Las primitivas se definen por su forma (geometría) y por su aspecto (atributos)
- Un atributo es un parámetro que afecta al aspecto que va a tener la primitiva al dibujarse
- Los principales: color , tamaño.
- Especiales: dependen del tipo de primitiva, por ejemplo, el patrón de relleno de un área, la fuente del texto
- Siempre tiene que tener un valor por defecto
- En una librería los atributos se activan mediante variables de estado

Atributos

- Punto: solo dispone de color y tamaño.
- Líneas y Curvas: ancho, estilo, pincel



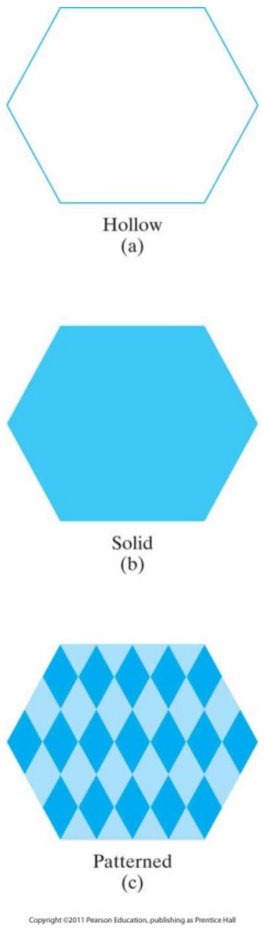
Copyright ©2011 Pearson Education, publishing as Prentice Hall



Copyright ©2011 Pearson Education, publishing as Prentice Hall

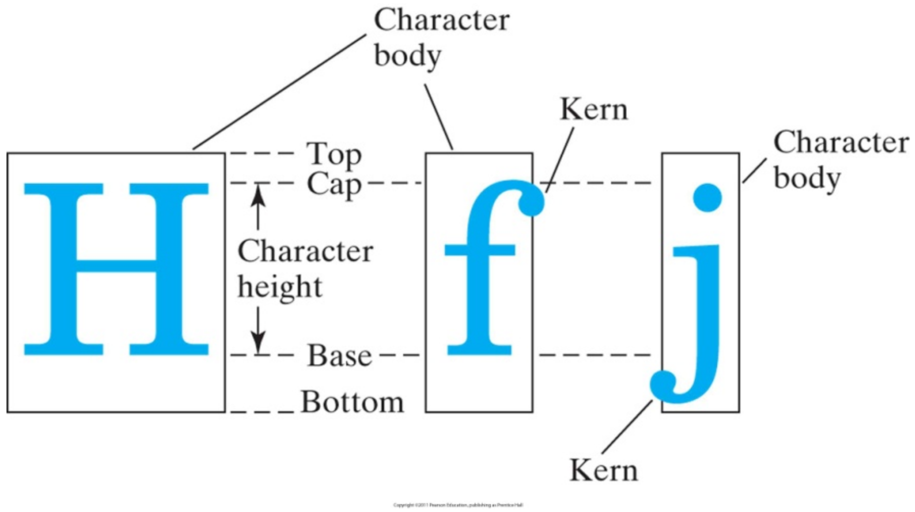
Atributos

- Áreas rellenas: Color, hueco, sólido, con patrón



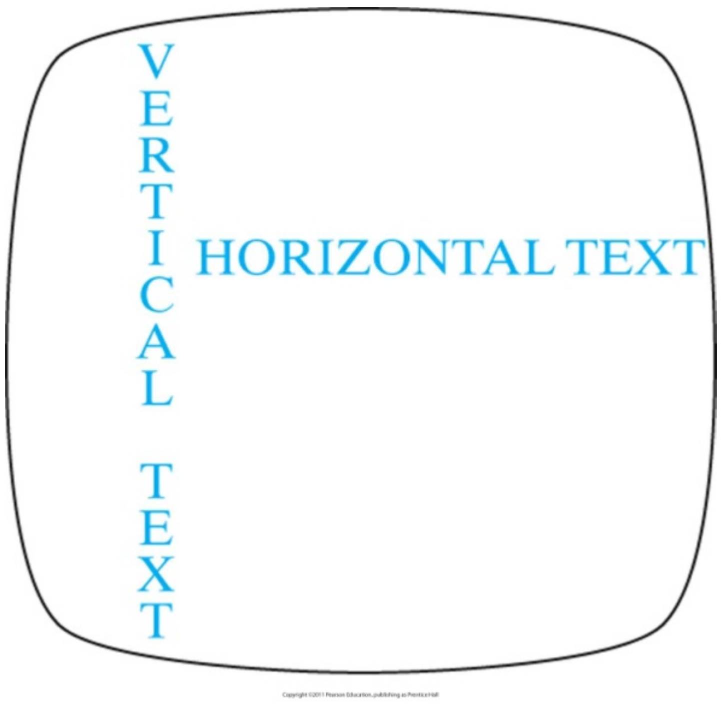
Atributos

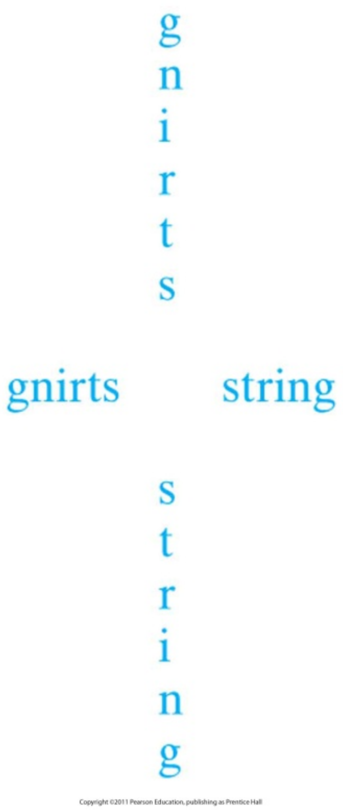
- Caracteres: Fuente, estilo, altura, anchura, espaciado

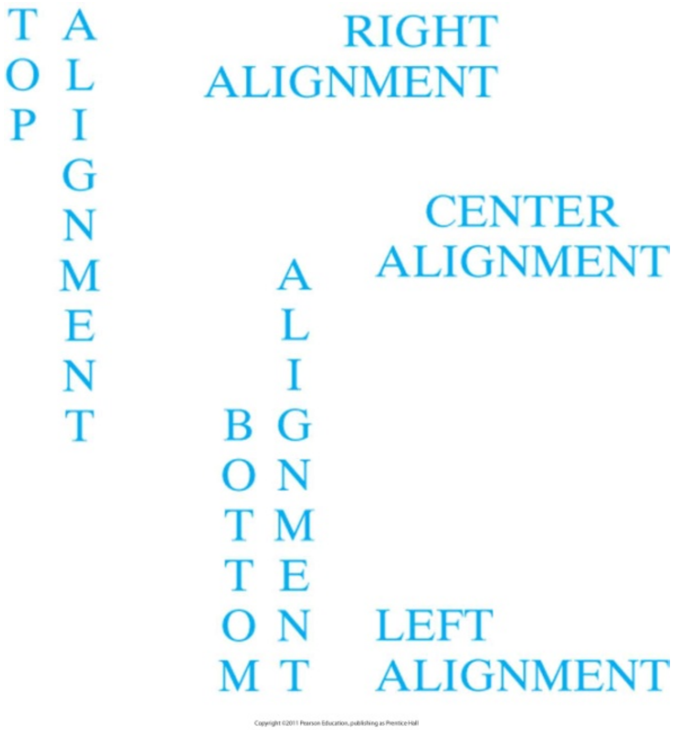


Atributos

- Caracteres: Orientación, sentido, alineación



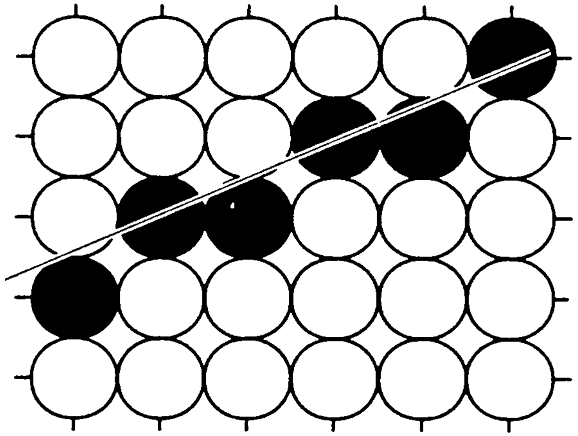




Algoritmos Líneas

- **Descripción del problema.**
  - Consiste en calcular las coordenadas de los píxeles que representan una recta infinitamente delgada colocada sobre la malla de un raster 2D
- **Se asume que las rectas**:
  - Son continuas
  - Con color constante, independiente de su orientación y longitud
  - Se han de dibujar tan rápido como sea posible

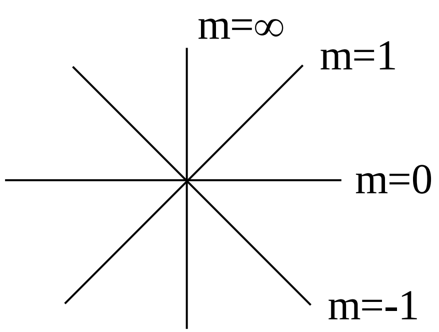
- **No consideramos rectas con atributos:**
  - estilo de línea
  - ancho de línea



Algoritmos Líneas

Algoritmo de fuerza bruta

- Es la estrategia más simple para convertir rectas al raster utilizando su ecuación:



$$y=m*x+b$$

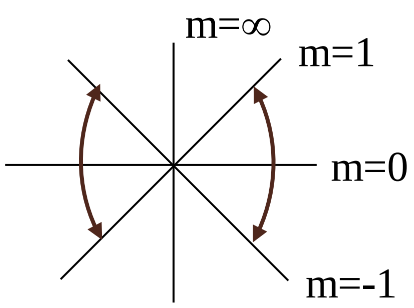
donde $m=dy/dx$

- Algoritmo de conversión de una recta de extremos (x0,y0)-(x1,y1):

```
m=(y1-y0)/(x1-x0)
b=y1-x1·m
Para x desde x0 hasta x1
(increm/decrem unitarios)
  y = x·m + b
  dibuja_pixel(x, round(y))
finPara
```

Algoritmos Líneas
Algoritmo de fuerza bruta

- **Restricción**
  - La restricción que ha de cumplir el algoritmo es que *m* se encuentre entre -1 y 1.
- **Soluciones a la restricción**
  - Calcular la x en función de incrementos unitarios de y para valores de m que no se encuentren entre –1 y 1 (con una pendiente 1/*m*)

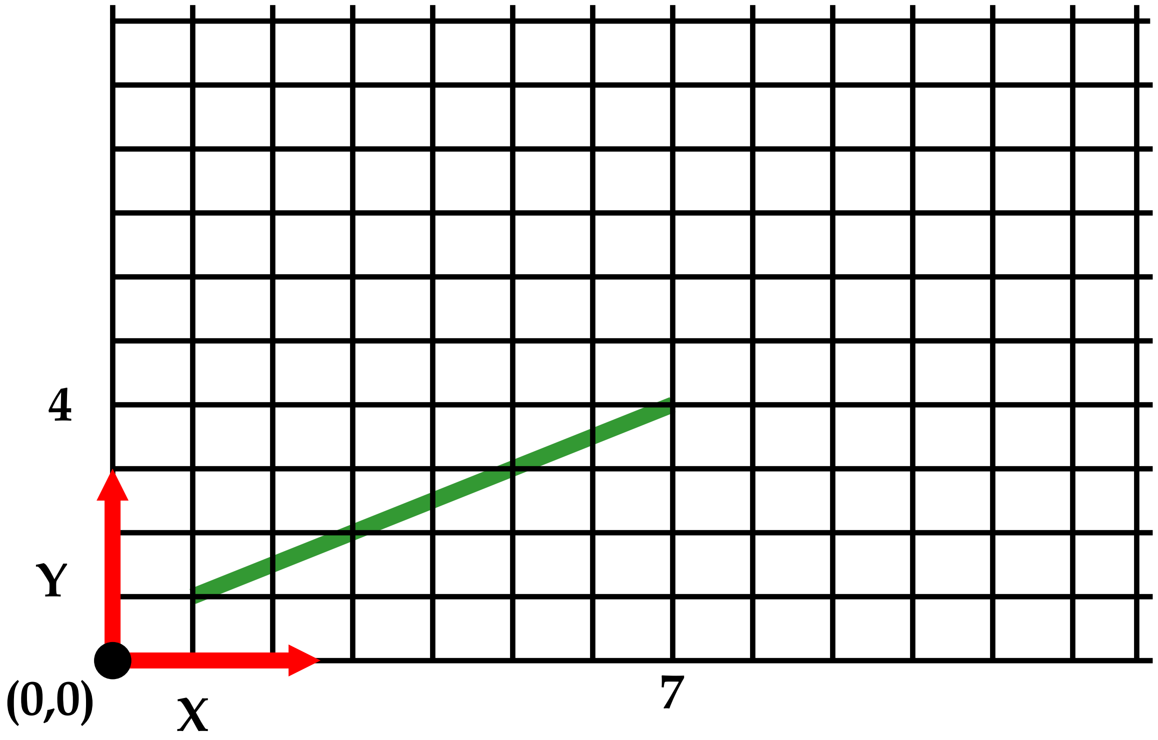


- **Desventajas**
  - El algoritmo es ineficiente, ya que en cada iteración requiere una multiplicación en coma flotante y una invocación a la función **round()**

Algoritmos Líneas

Algoritmo de fuerza bruta

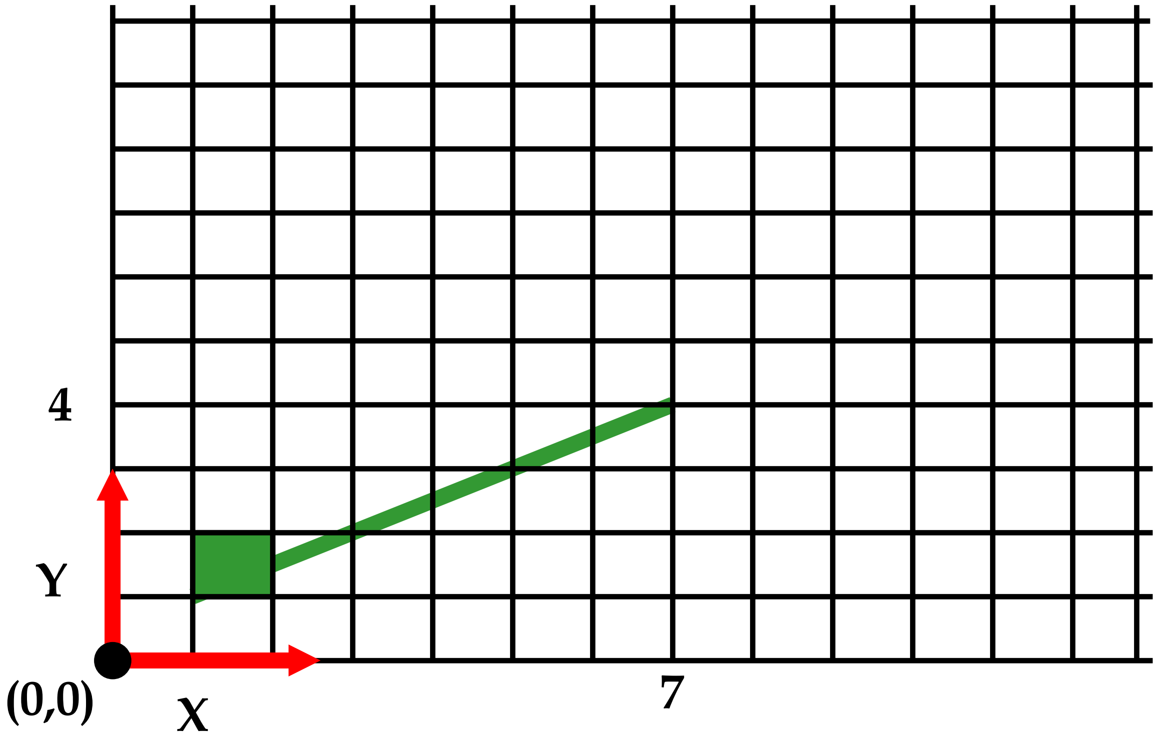
- m = dy/dx ← $m=(4-1)/(7-1)=3/6=1/2=0.5$
- b = y1 – x1·m; ← $b=1 - 1*0.5 = 0.5$
- Para cada *x* desde *x1* hasta *x2* (incrementos unitarios)
  - *y = x·m + b*
  - dibuja_pixel(*x, round(y)*)
- finpara



Algoritmos Líneas

Algoritmo de fuerza bruta

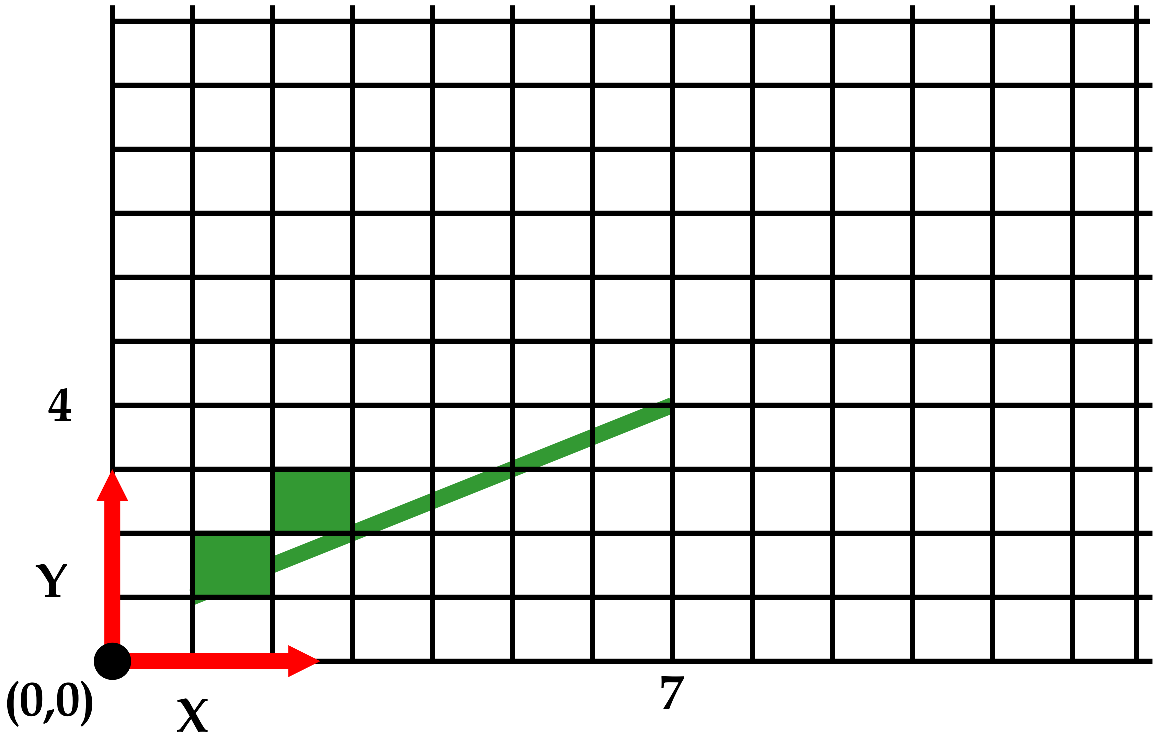
- m = dy/dx — m=(4-1)/(7-1)=3/6=1/2=0.5
- b = y1 – x1·m; — b=1 – 1*0.5 = 0.5
- Para cada *x* desde *x1* hasta *x2* (incrementos unitarios) — x = 1
  - *y = x·m + b* — y=1*0.5 + 0.5 = 1
  - dibuja_pixel(*x, round(y)*) — dibuja_pixel(1,1)
- finpara



Algoritmos Líneas

Algoritmo de fuerza bruta

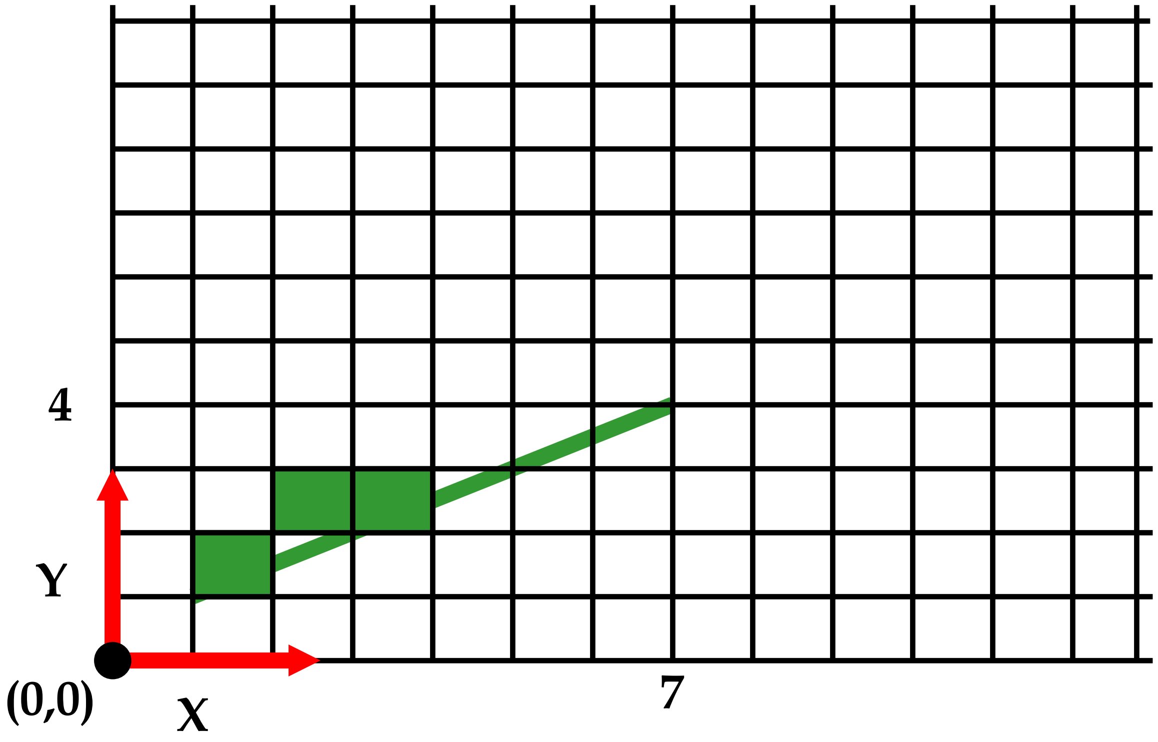
- m = dy/dx ← $m=(4-1)/(7-1)=3/6=1/2=0.5$
- b = y1 – x1·m; ← $b=1 - 1*0.5 = 0.5$
- Para cada *x* desde *x1* hasta *x2* (incrementos unitarios) ← $x = 2$
  - $y = x \cdot m + b$ ← $y=2*0.5 + 0.5 = 1,5$
  - dibuja_pixel(*x*, *round*(*y*)) ← dibuja_pixel(2,2)
- finpara



Algoritmos Líneas

Algoritmo de fuerza bruta

- m = dy/dx ← m=(4-1)/(7-1)=3/6=1/2=0.5
- b = y1 – x1·m; ← b=1 – 1*0.5 = 0.5
- Para cada *x* desde *x1* hasta *x2* (incrementos unitarios) ← x = 3
  - *y = x·m + b* ← y=3*0.5 + 0.5 = 2
  - dibuja_pixel(*x, round(y)*) ← dibuja_pixel(3,2)
- finpara

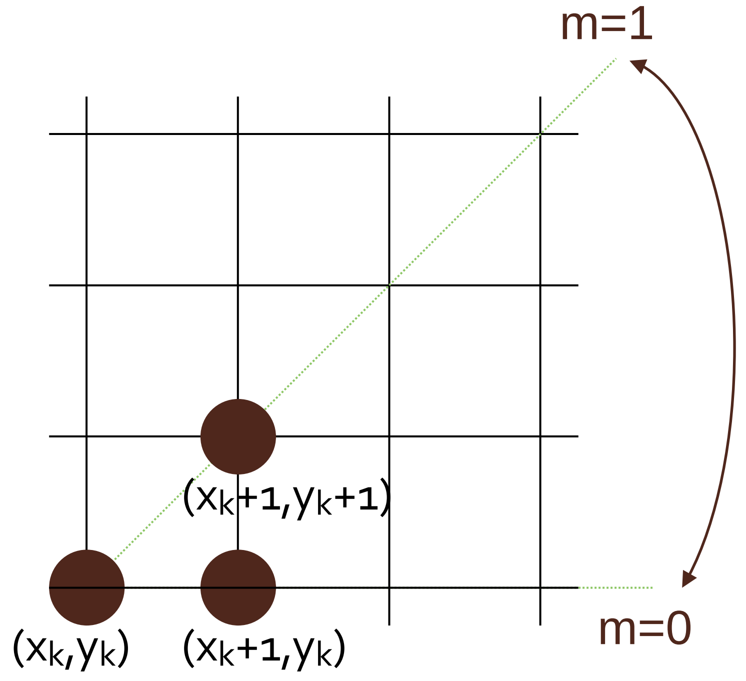


Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

- Utiliza únicamente operaciones con enteros
- La ventaja de este algoritmo radica en que es fácilmente aplicable a cualquier tipo de elemento geométrico

  - Restricción
    - La inclinación de la recta m se ha de encontrar dentro del intervalo [0,1]
    - Esta restricción permite afirmar que si (x_k, y_k) es el píxel dibujado en la etapa *k* del algoritmo, el siguiente píxel que se encontrará más cerca de la recta será (x_k+1, y_k) o (x_k+1, y_k+1)

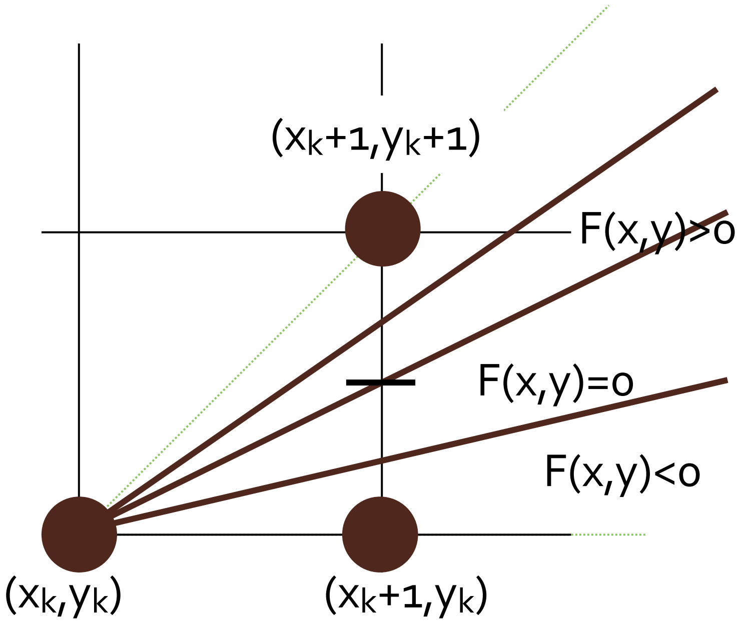


Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

- A partir de la ecuación de la recta (y=mx+b), se puede obtener una función implícita F(x, y) $\Longrightarrow$ $F(x,y) = dy \cdot x - dx \cdot y + b \cdot dx$
- Siendo $m = dy/dx$.
  - Si F(x,y)=0 entonces el punto (x,y) está en la recta
  - si F(x,y)>0 el punto está debajo de la recta y,
  - si F(x,y)<0 el punto está encima de la recta real
- Para saber el punto a elegir, hay que calcular $F(x_k + 1, y_k + 1/2)$ y analizar el signo de la función
- Al valor de decisión F en la iteración k será:

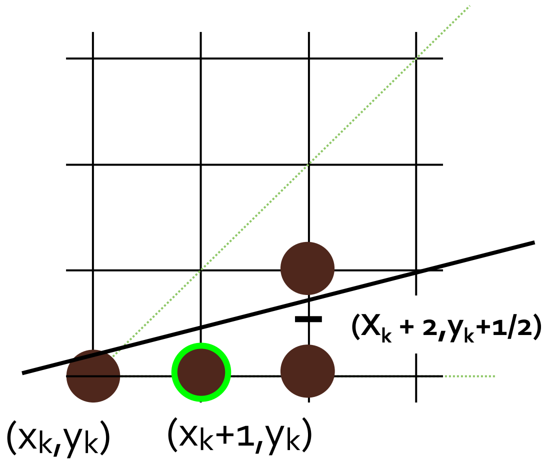
$$d_k = F(x_k + 1, y_k + \frac{1}{2}) = dy \cdot (x_k + 1) - dx \cdot (y_k + \frac{1}{2}) + b \cdot dx$$



Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

$$d_k = F(x_k + 1, y_k + \frac{1}{2}) = dy \cdot (x_k + 1) - dx \cdot (y_k + \frac{1}{2}) + b \cdot dx$$



- Si en la iteración *k* el punto elegido es (x_k+1, y_k), el valor de decisión en la iteración *k+1* será:

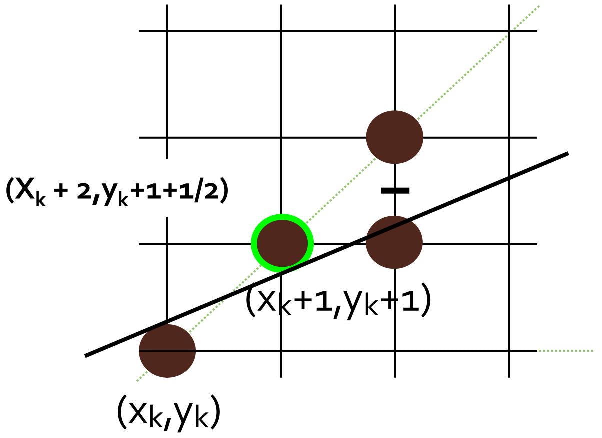
$$d_{k+1} = F(x_k + 2, y_k + \frac{1}{2}) = dy \cdot (x_k + 2) - dx \cdot (y_k + \frac{1}{2}) + b \cdot dx$$

  - Si resto d_{k+1}- d_k se deduce que:

$$d_{k+1} = d_k + dy$$

- Si, en cambio, en la iteración k el punto elegido fue $(x_k + 1, y_k + 1)$, el valor en k+1 será:

$$d_{k+1} = F\left(x_k + 2, y_k + \frac{3}{2}\right) = dy \cdot (x_k + 2) - dx \cdot (y_k + \frac{3}{2}) + b \cdot dx$$



  - Si resto d_{k+1}- d_k se deduce que:

$$d_{k+1} = d_k + dy - dx$$

Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

- El primer parámetro de decisión, sería el parámetro de decisión para el punto inicial (x0, y0) de la recta:

$$d_0 = F(x_0 + 1, y_0 + \frac{1}{2}) = dy \cdot (x_0 + 1) - dx \cdot (y_0 + \frac{1}{2}) + b \cdot dx$$

$$d_0 = dy \cdot x_0 - dx \cdot y_0 + b \cdot dx + dy - \frac{dx}{2}$$

$$d_0 = F(x_0, y_0) + dy - \frac{dx}{2}$$

- Pero dado que (x0, y0) es un punto de la recta, F(x0, y0) = 0. Por tanto,

  d0=dy - dx/2

- Para eliminar la fracción en el parámetro de decisión en el instante inicial, se puede multiplicar la función F por 2 y utilizar la función resultante F' para el cálculo de los parámetros de decisión (el signo no cambia).

Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

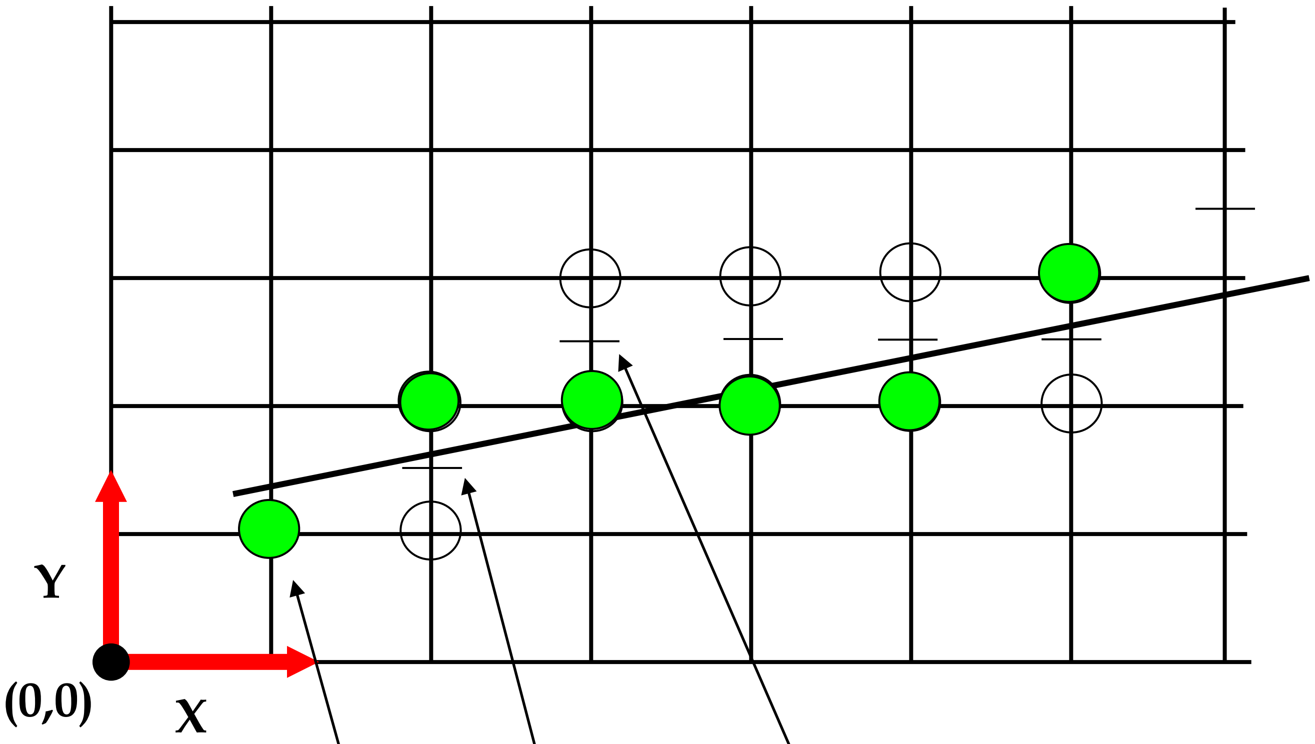
- El algoritmo del punto medio para una recta de extremos (x0,y0)-(x1,y1) tendría de la siguiente forma:

```
Introducir los puntos extremos de una recta
(x0, y0) = punto más a la izquierda de la recta
y=y0
d=2dy - dx
Para cada x entre x0 y x1 (incrementos unitarios)
    dibuja_pixel(x, y)
    Si d < 0
        d= d + 2dy
    sino
        y=y + 1
        d=d + 2dy - 2dx
    finSi
finPara
```

Algoritmos Líneas

Algoritmo del punto medio (Bresenham)

Vídeo: https://media.upv.es/player/?id=d98ae620-fc0b-11ea-9ede-d1ad8f82e7cd



$P(x_k,y_k)$

$M_k=(x_k+1,y_k+1/2)$ $M_{k+1}=(x_k+2,y_k+3/2)$

Algoritmos Circunferencias
Algoritmo de fuerza bruta

- Se utiliza la ecuación de la circunferencia:

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

- Si el circulo está sobre el origen de coordenadas y es de radio r:

$$y = \pm\sqrt{r^2 - x^2}$$

```
Para cada x desde -r hasta r
(incrementos unitarios)
```

$$y1 = +\sqrt{r^2 - x^2}$$

$$y2 = -\sqrt{r^2 - x^2}$$

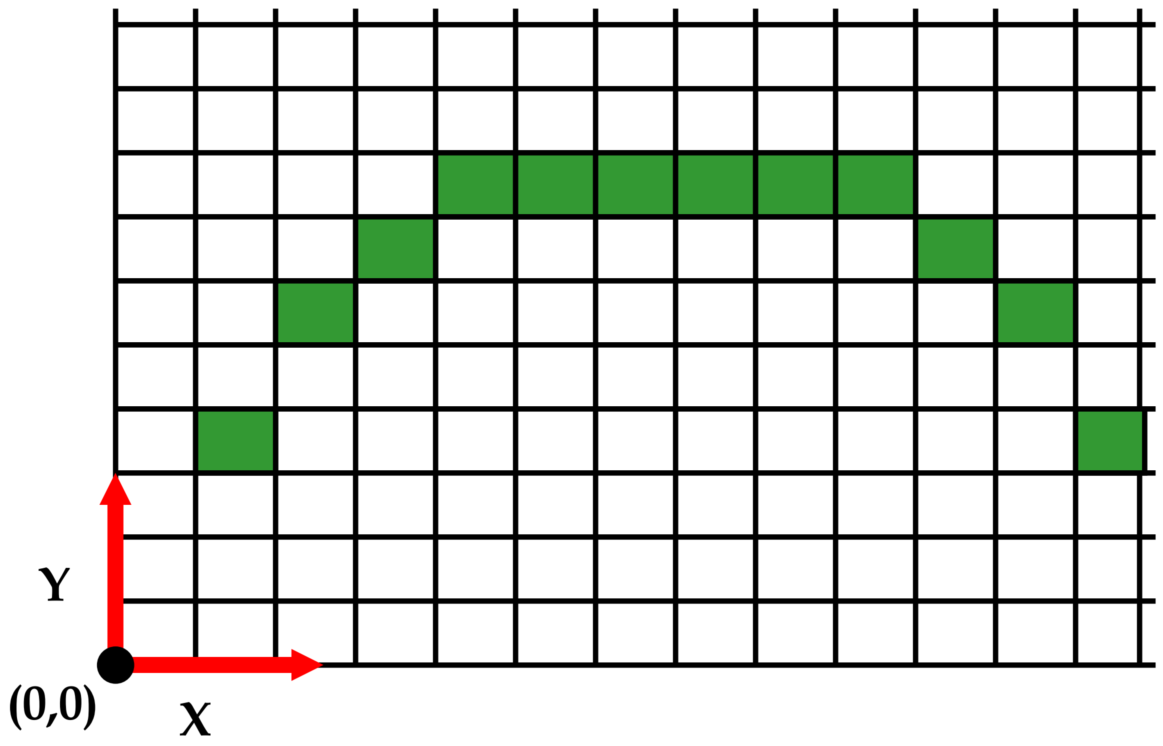
```
    dibuja_pixel(x, round(y1))
    dibuja_pixel(x, round(y2))
finPara
```

- Problemas
  - Realiza operaciones en coma flotante (raíz cuadrada y *round()*)
  - La distancia entre los puntos de la circunferencia no es homogénea
  - Se pueden utilizar coordenadas polares (distribución homogénea de puntos) y trazar líneas entre la secuencia de puntos generados

Algoritmos Circunferencias

Algoritmo de fuerza bruta

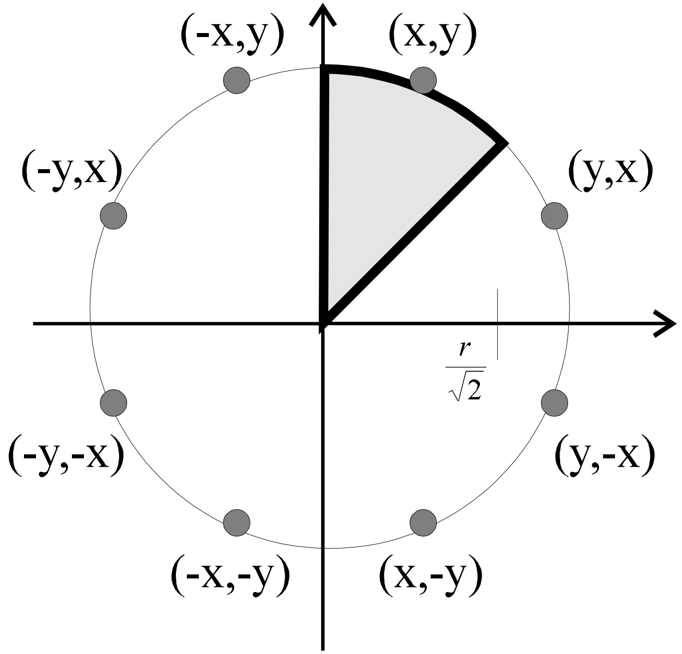
- La distancia entre los puntos de la circunferencia no es homogénea:



Algoritmos Circunferencias

Simetría de ocho puntos

- Se puede mejorar el proceso de dibujo del punto anterior mediante la utilización de la simetría del círculo
- Dado un punto (x, y) de un círculo centrado en el origen de coordenadas, a partir de dicho punto se pueden calcular los otros siete puntos (sólo se tendrán que calcular los primeros 45º sombreados del círculo para dibujarlo completamente)



Algoritmos Circunferencias

Simetría de ocho puntos

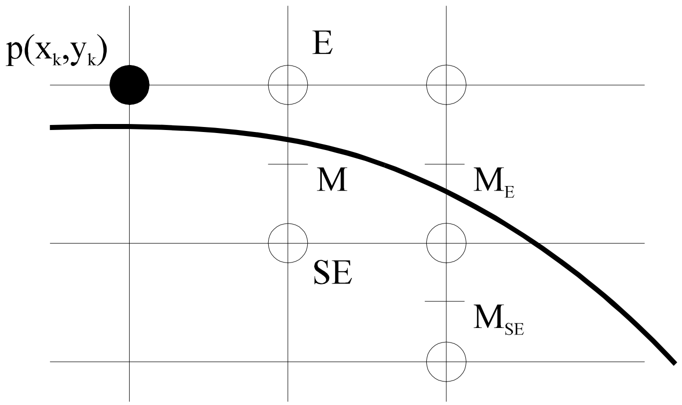
- La incorporación en el algoritmo de fuerza bruta, consistirá en que la x varíe entre 0 y $r/\sqrt{2}$
- Sólo se calculan las y positivas, y en lugar de llamar al procedimiento *dibuja_pixel()* se llama a *puntos_círculo():*

```
puntos_círculo(x, y : enteros)
    dibuja_pixel(x, y)
    dibuja_pixel(y, x)
    dibuja_pixel(y, -x)
    dibuja_pixel(x, -y)
    dibuja_pixel(-x, -y)
    dibuja_pixel(-y, -x)
    dibuja_pixel(-y, x)
    dibuja_pixel(-x, y)
finProcedimiento
```

- Con esta modificación se eliminan las discontinuidades del algoritmo de fuerza bruta, pero se siguen haciendo cálculos en coma flotante

Algoritmos Circunferencias
Algoritmo del Punto Medio

- Utiliza aritmética entera
- Se basa en la simetría del círculo (ocho puntos)
- Dadas las características del segundo octante, si el último punto que se ha convertido al raster es (x_k, y_k), el siguiente será (x_k+1, y_k) o (x_k+1, y_k-1)

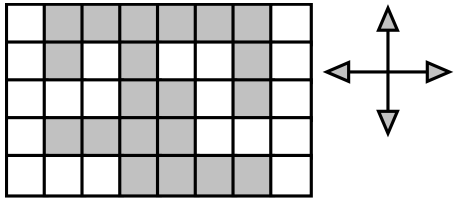


- Para determinar cuál de los dos puntos se encuentra más cerca del círculo real, se mira el punto medio entre ambos
  - Si se encuentra por encima de la curva del círculo, se elige $(x_k + 1, y_k - 1)$
  - Si está por debajo, el punto a elegir será $(x_k + 1, y_k)$
- Para determinar la posición del punto medio se utiliza *F(x, y)*, función deducida a partir de la ecuación del círculo:

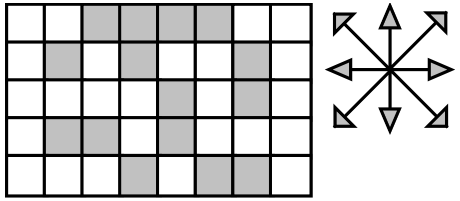
$$F(x,y) = x^2 + y^2 - r^2$$

Algoritmos regiones

- Región: conjunto de píxeles conectados
- Los píxeles de una misma región se definen por:
  - El tipo de conectividad:
    - Regiones 4-Conectadas
    - Regiones 8-Conectadas

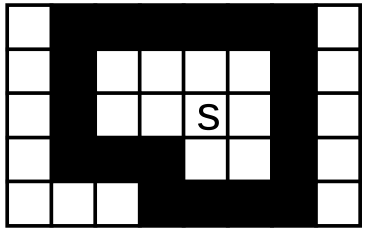


4-CONECTADA

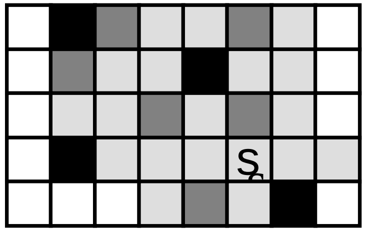


8-CONECTADA

  - El color
    - Definida por su interior: píxeles de un mismo color
    - Definida por su frontera: píxeles de un color distinto al de la frontera



FRONTERA



INTERIOR

Algoritmos regiones

- Precisan de un punto inicial de la región denominado semilla
- Algoritmo para el rellenado de una región 4-conectada definida por su interior

```
Procedimiento rellenadoInterior4(x, y, : entero; viejo, nuevo: color);
Empezar
    Si obtener_pixel(x, y) = viejo entonces
            dibujar_pixel(x, y, nuevo);
            rellenadoInterior4(x, y-1, viejo, nuevo);
            rellenadoInterior4(x, y+1, viejo, nuevo);
            rellenadoInterior4(x-1, y, viejo, nuevo);
            rellenadoInterior4(x+1, y, viejo, nuevo);
    finSi
finProcedimiento
```

Algoritmos regiones

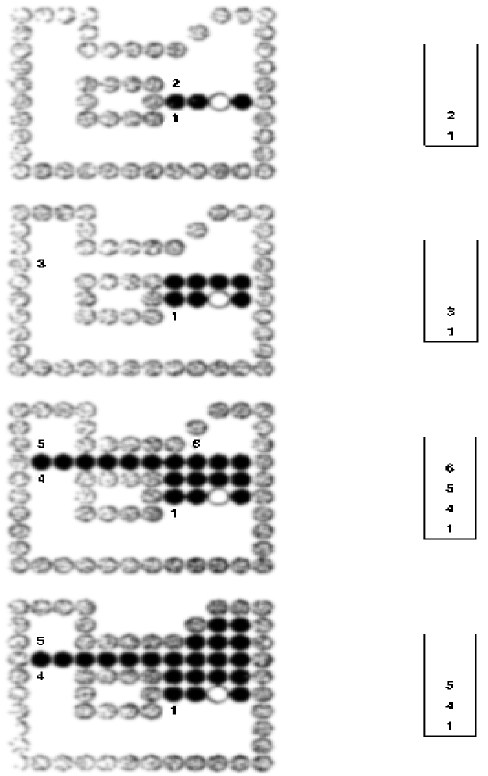
- Algoritmo para el rellenado de una región 4-conectada definida su frontera

```
Procedimiento rellenadoFrontera4(x, y, : entero; frontera, nuevo: color);
Var    c: color;
Empezar
        c ← obtener_pixel(x, y);
        Si (c<>frontera) y (c<>nuevo) entonces
                dibujar_pixel(x, y, nuevo);
                rellenadoFrontera4(x, y-1, frontera, nuevo);
                rellenadoFrontera4(x, y+1, frontera, nuevo);
                rellenadoFrontera4(x-1, y, frontera, nuevo);
                rellenadoFrontera4(x+1, y, frontera, nuevo);
        finSi
finProcedimiento
```

Algoritmos regiones

- Algoritmo iterativo (área 4-conexa) por frontera

> *P = pixel semilla*
>
> *Apilar P en la pila de semillas*
>
> *Mientras la pila de semillas no esté vacía*
>
> *S = cabeza pila semillas (se desapila)*
>
> *Se rellena el tramo de píxeles de S*
>
> *Se apilan los extremos izquierdos de los tramos conectados por abajo con los píxeles del tramo actual que no estén al nuevo color*
>
> *Se apilan los extremos izquierdos de los tramos conectados por arriba con los píxeles del tramo actual que no estén al nuevo color*
>
> *finMientras*



- Vídeo: https://media.upv.es/player/?id=fb072250-fc0b-11ea-9ede-d1ad8f82e7cd

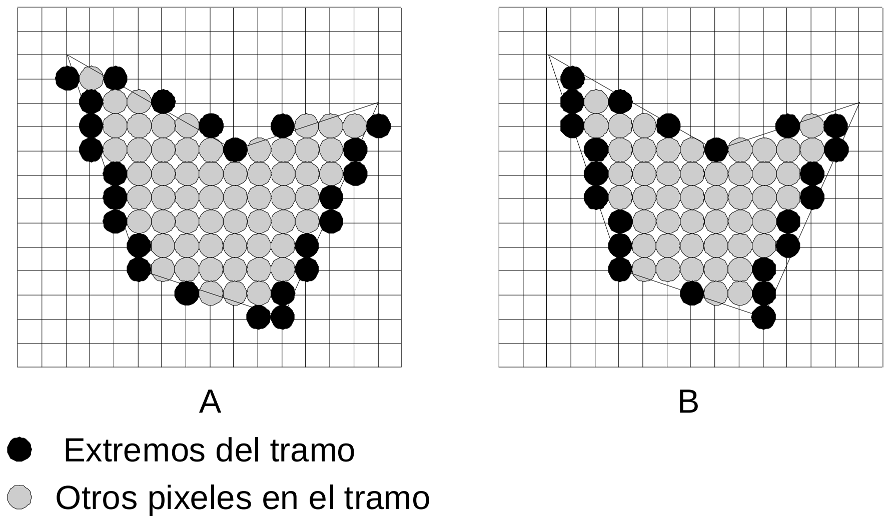
Algoritmos Polígonos

- Dadas las coordenadas y atributos del polígono
  - Determinar qué píxeles debemos rellenar
  - Decidir el color de los píxeles (patrones)

- Dos posibilidades
  - Trazar frontera mediante algoritmo de conversión de rectas y utilizar relleno por frontera
  - Algoritmos específicos para trazo de polígonos con relleno (Scan-line)

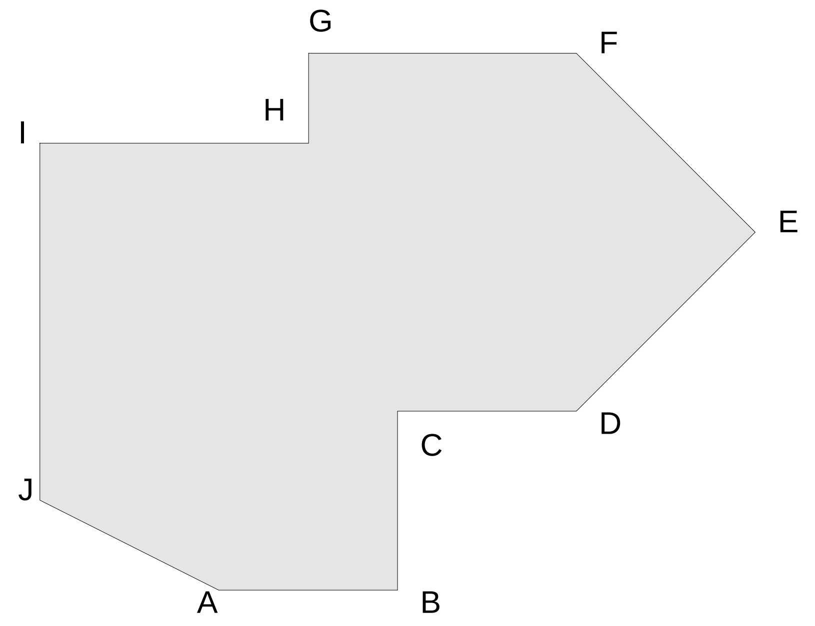
Algoritmos Polígonos

- **Algoritmo**
  - Cálculo de las intersecciones de la LDR con todas las aristas
  - Ordenación de intersecciones por *x* creciente
  - Rellenar pares de intersecciones utilizando
- **Utilización del algoritmo del punto medio en las aristas**
  - **Problema**: algunos píxeles de los extremos caen fuera del polígono (Figura A)



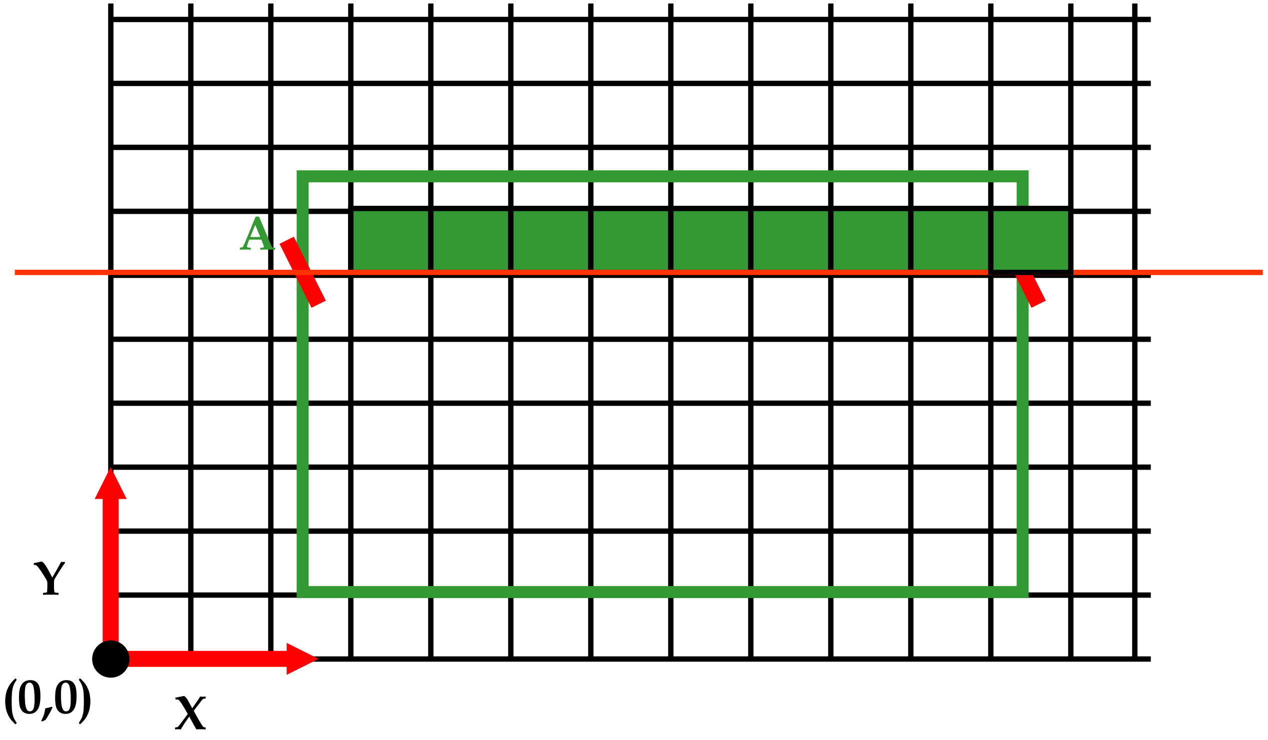
Algoritmos Polígonos

- **Rellenado de pares de intersecciones**
  - Dada una intersección con un valor real de x ¿qué pixel seleccionamos?
  - ¿Qué ocurre si el valor de x en la intersección es entero?
  - ¿Qué ocurre si la intersección es un vértice?
  - ¿Qué ocurre con las aristas horizontales?



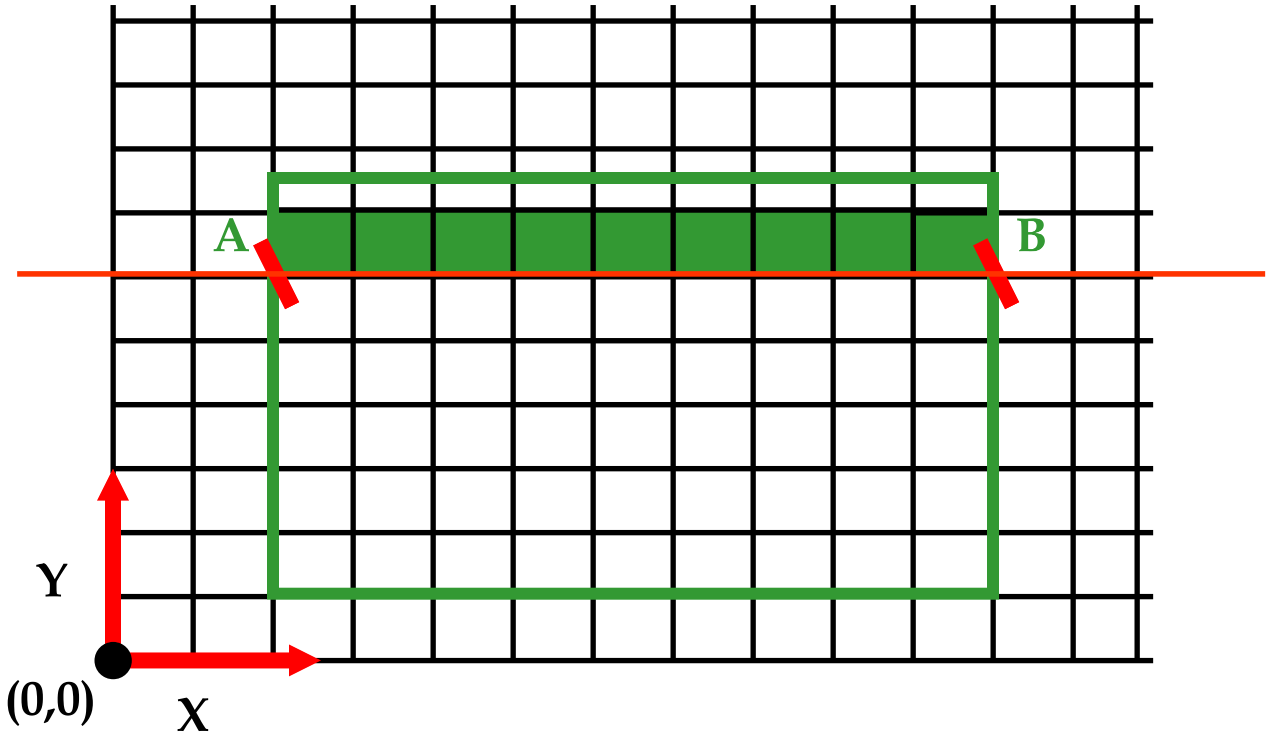
Algoritmos Polígonos

- **Dada una intersección con un valor real de *x* ¿qué píxel seleccionamos?:**
  - Si es la intersección de la izquierda redondeamos por arriba, sino por abajo



Algoritmos Polígonos

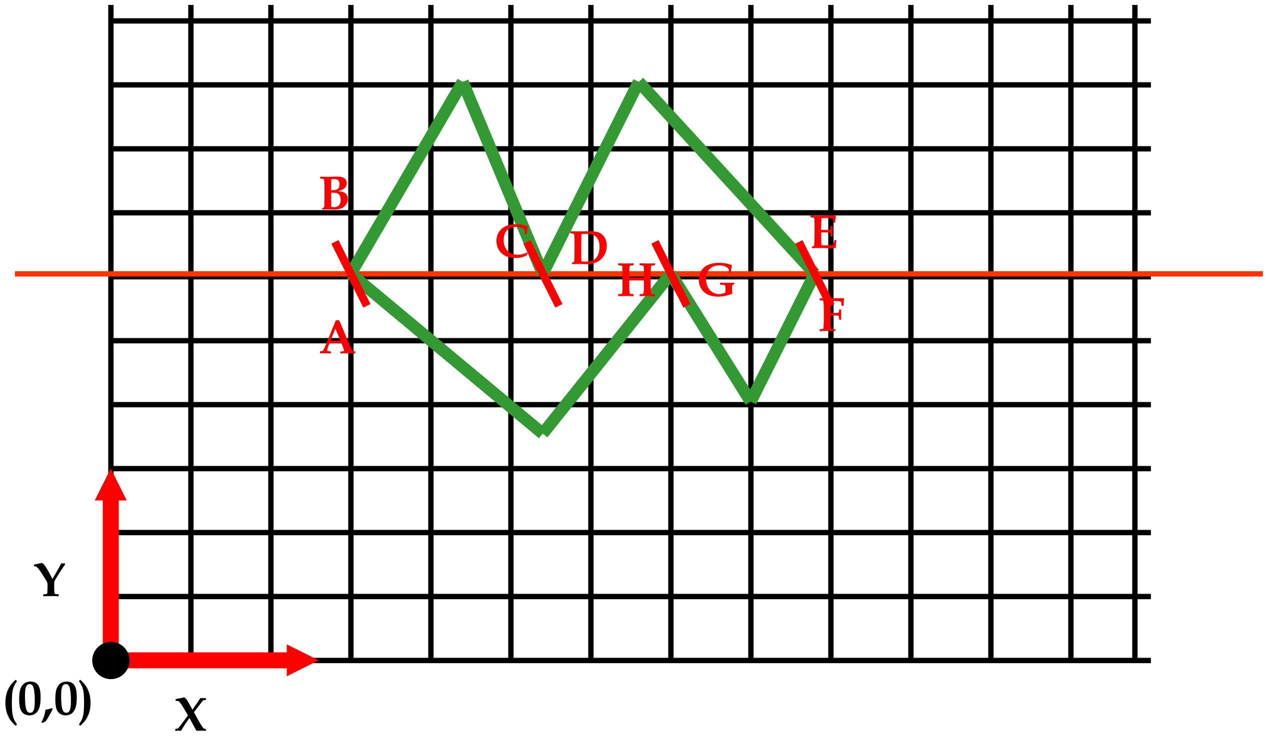
- **¿Qué ocurre si el valor de *x* en la intersección es entero?**
  - Si es la intersección de la izquierda escogemos ese píxel, sino el anterior



Algoritmos Polígonos

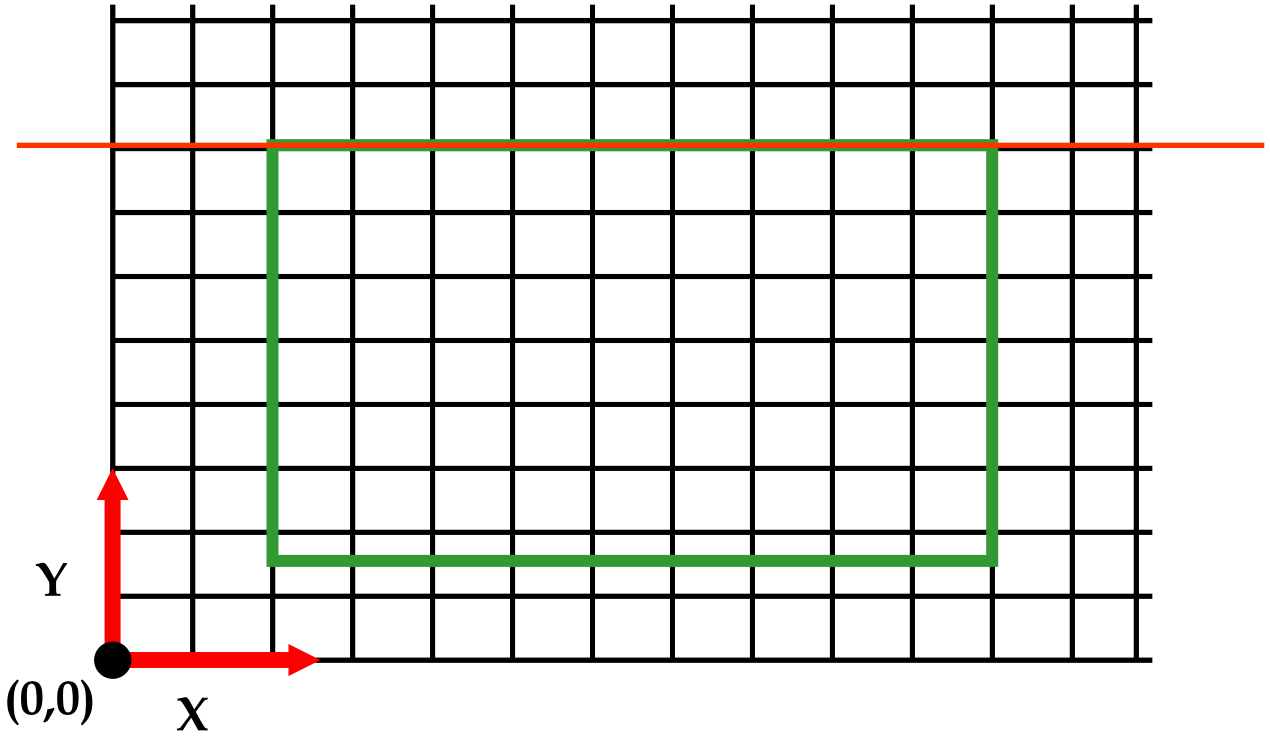
- **¿Qué ocurre si la intersección es un vértice?**
  - Para calcular tramos elegimos el vértice con Ymin de cada arista

(B, C), (D, E)



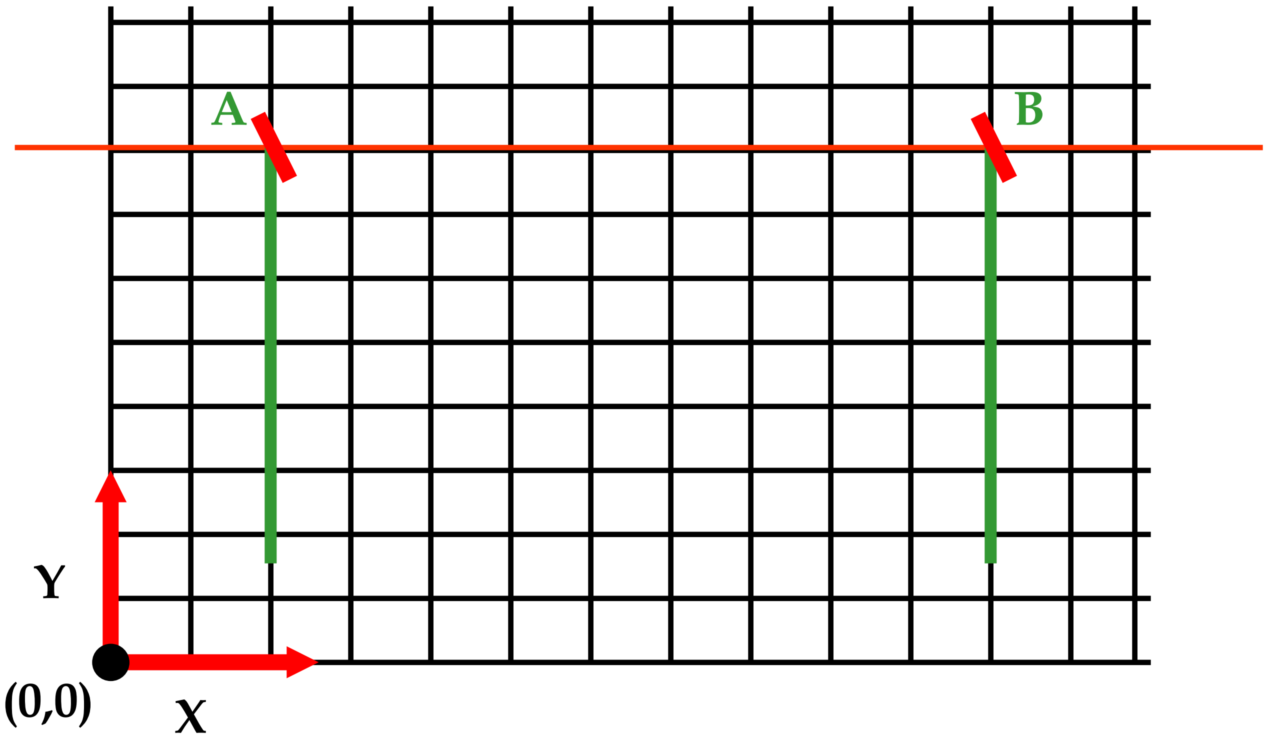
Algoritmos Polígonos

- **¿Qué ocurre con las aristas horizontales ?**
  - No se tienen en cuenta al calcular las intersecciones



Algoritmos Polígonos

- **¿Qué ocurre con las aristas horizontales ?**
  - No se tienen en cuenta al calcular las intersecciones



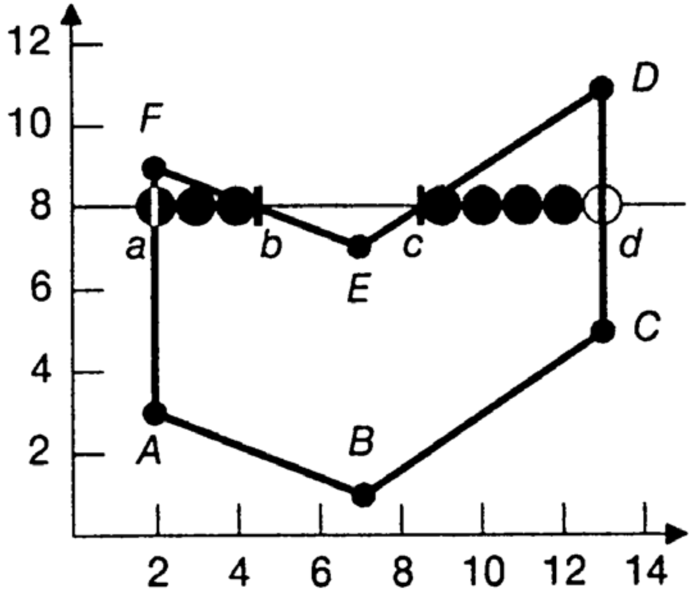
Algoritmos Polígonos

- Resumen:

  - Coordenada X de la intersección
    - Real
      - interior-exterior?: redondeo por abajo
      - exterior-interior?: redondeo por arriba
    - Entera
      - Arista horizontal: se descarta
      - En un vértice
        - Y_{min}?: se conserva
        - Y_{max}?: se descarta
      - No está en un vértice
        - Extremo izquierda?: interior
        - Extremo derecha?: exterior

Algoritmos Polígonos
Algoritmo de línea de rastreo

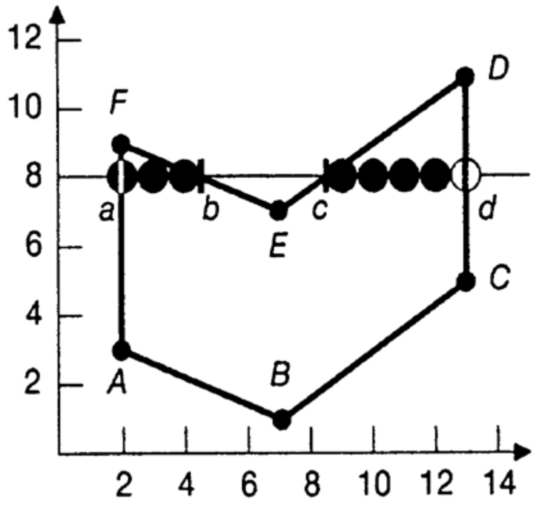
- Rellena áreas cuyo límite se define mediante un polígono
  - Pasos del algoritmo
    - Trazar líneas de rastreo y calcular las intersecciones con el polígono
    - Ordenar las intersecciones por orden creciente de las *x*
    - Agrupar las intersecciones por pares consecutivos
    - Rellenar los píxeles que caen entre las intersecciones

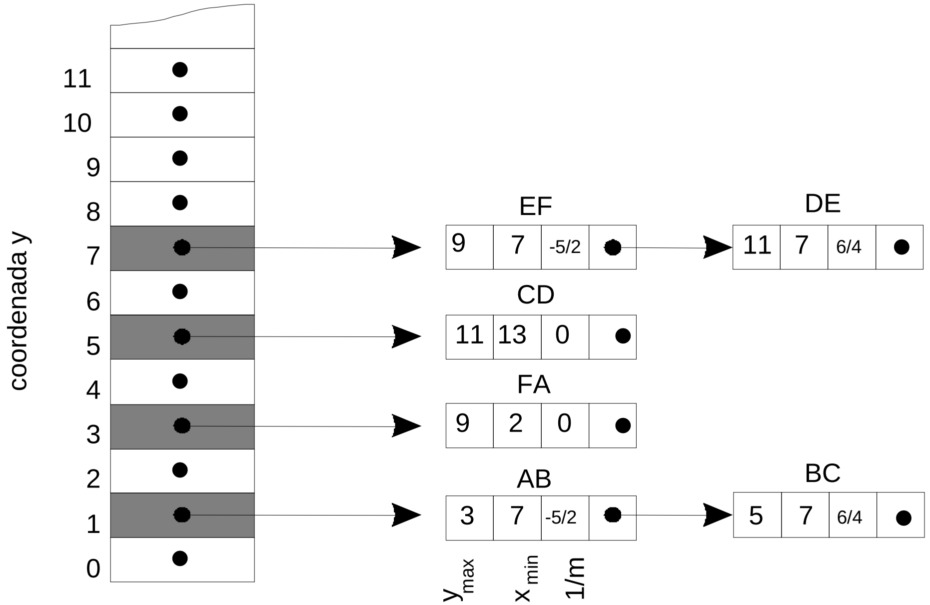


Algoritmos Polígonos

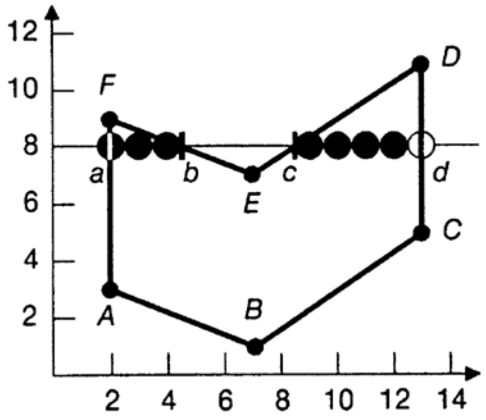
Algoritmo de lista de aristas activas

- POR CADA ARISTA: *Y* máxima de la arista, *X* correspondiente a la *Y* mínima de la arista, Incremento de *X* entre dos LDR (1/m)
- Se almacena en la lista asociada a la primera LDR que corta la arista

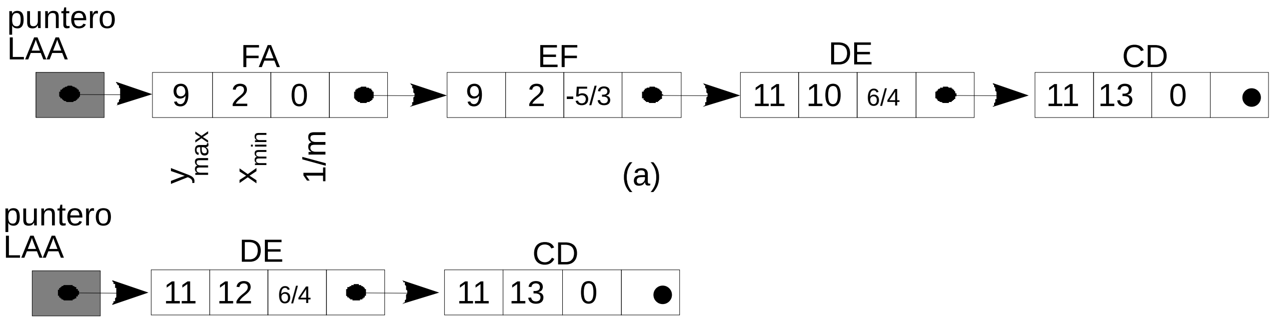




Algoritmos Polígonos
Algoritmo de lista de aristas activas

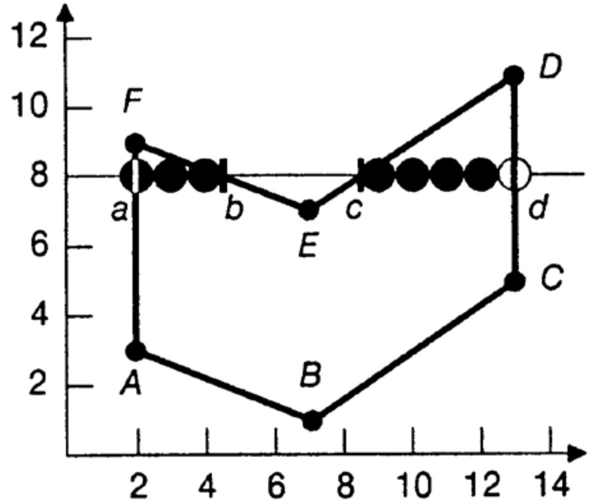


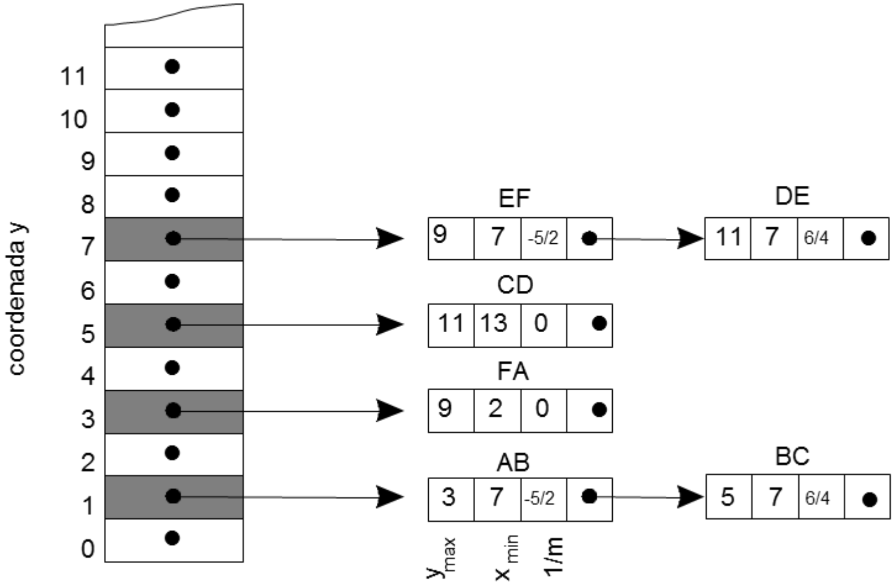
- *Y* igual a la primera LDR significativa
- LAA vacía
- Repetir hasta que la LA y la LAA queden vacías
  - Lista *Y* a la LAA. Ordenar la LAA por X
  - Eliminar de la LAA aquellas entradas cuya $Ymax = Y$
  - Rellenar parejas de intersecciones
  - Incrementar *Y* en 1
  - Actualizar el valor de *X* para cada arista no vertical que quede en la LAA



(b)

LDR	LISTA ARISTAS ACTIVAS				PÍXELES
1	AB X=7	BC X=7			(7,1)-(7,1)
2	AB X=4.5	BC X=8.5			(5,2)-(8,2)
3	AF X=2	BC X=10			(2,3)-(9,3)
4	AF X=2	BC X=11.5			(2,4)-(11,4)
5	AF X=2	CD X=13			(2,5)-(12,5)
6	AF X=2	CD X=13			(2,6)-(12,6)
7	AF X=2	EF X=7	ED X=7	CD X=13	(2,7)-(6,7) (7,7)-(12,7)
8	AF X=2	EF X=4.5	ED X=8.5	CD X=13	(2,8)-(4,8) (9,8)-(12,8)
9	ED X=10	CD X=13			(10,9)-(12,9)
10	ED X=11.5	CD X=13			(12,10)-(12,10)





Algoritmos Polígonos

Algoritmo de lista de aristas activas

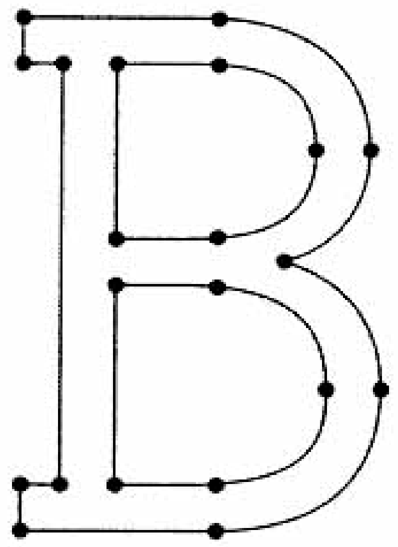
- En este screencast se explica el algoritmo de línea de rastreo para conversión de polígonos:
  - Rellenado de polígonos
  - Intersección con un valor real de X
  - Intersección con un valor entero de X
  - Intersección en un vértice
  - Aristas horizontales
  - Algoritmo de aristas activas
  - Ejercicio
- http://hdl.handle.net/10251/105189

Algoritmos Texto

RASTERIZACIÓN DE CARACTERES.

- Existen dos técnicas básicas a la hora de definir caracteres: *fuentes tipo bitmap* y *fuentes tipo vectorial* (o *outline font*).

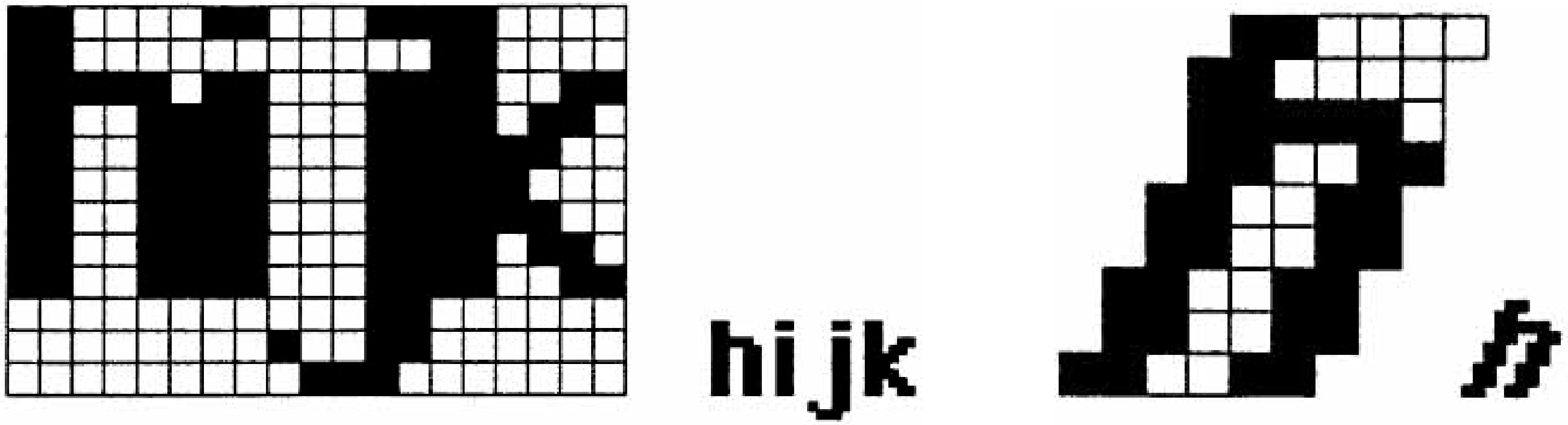
1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0



Algoritmos Texto

Rasterización de caracteres: tipo bitmap

- Las letras de una fuente de texto se definen en una matriz de bits (*bitmap*)
- Para cada tamaño y tipo (negrita, itálica) hay una matriz



- **Ventaja:** se visualizan rápidamente: para escribir un carácter basta con recortarlo de la matriz de bits y pegarlo en el dispositivo
- **Inconveniente:** sufren aliasing, especialmente en itálica

Algoritmos Texto

Rasterización de caracteres: tipo vectorial

- Las letras de una fuente se definen mediante un conjunto de curvas y rectas que representan el contorno de los caracteres
- **Ventaja**: Una fuente vectorial sirve para un rango de tamaños, sin problemas de aliasing.
- **Inconvenientes**:
  - El dibujo de una letra requiere su conversión al ráster en función de su tamaño y tipo (negrita, itálica, etc.)
  - Es más ineficiente.
- Queremos una **solución intermedia**: Representación vectorial para almacenamiento y representación bitmap durante su utilización (después de cargar la fuente).

Ejemplos: TrueType (Apple), Type-1 (Adobe), FreeType (Linux)

Bibliografía

- D. Hearn, M. Baker. Computer Graphics with OpenGL. Pearson Prentice Hall, 4ª edición.
  - Capítulos 4,5 y 6