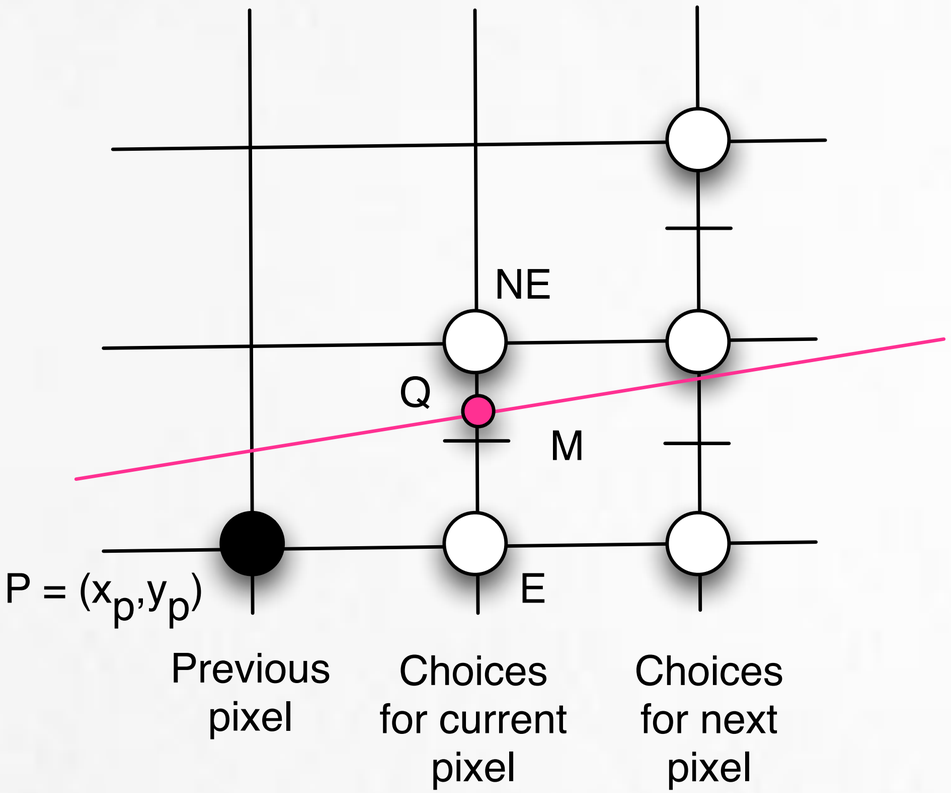
ALGORITMOS DE GRÁFICAS EN RASTER II

Computación Gráfica

Midpoint Line Algorithm

- Bresenham (1965)
- Utiliza solamente aritmética de enteros evitando funciones de redondeo.
- Permite el cálculo de (x_{i+1}, y_{i+1}) de manera incremental.
- Se extiende a aritmética de punto flotante.
- Se extiende a círculos.
- No se extiende fácilmente a cónicas arbitrarias.
- Modificación: midpoint line algorithm (Pitteway,1967), Van Aken (1984).
- Para líneas y círculos el algoritmo de Van Aken se reduce a la formulación de Bresenham.
- Minimiza el error (distancia) a la primitiva original.

Midpoint Line Algorithm



- Suponemos que la pendiente está entre 0 y 1 (las demás se manejan por reflexión sobre los ejes principales)
- Llamamos al punto extremo inferior izquierdo (x_0,y_0) y el superior derecho (x_1,y_1).
- Suponemos haber elegido al pixel **P** en (x_p,y_p) y debemos elegir entre el pixel **E** (east pixel) y el **NE** (northeast pixel).
- Sea **Q** el punto de intersección entre la línea deseada y la línea de la rejilla $x=x_p+1$.
- Si **M**, el punto medio entre NE y E, está abajo de la línea, elegimos E, si está arriba elegimos NE.
- El error será siempre menor o igual a 1/2.

Midpoint Line Algorithm

- Determinar si el punto medio está arriba o abajo de la línea.
- Representar la línea con una función implícita con coeficientes a, b y c: $F(x,y) = ax + by + c = 0$.
- Si $d_y = y_1 - y_0$ y $d_x = x_1 - x_0$ podemos escribir $y = (d_y/d_x)x + B$.
- $F(x,y) = d_y \cdot x - d_x \cdot y + B \cdot d_x = 0$
- $a=d_y$, $b=-d_x$, $c= B \cdot d_x$ en la forma implícita.
- Se puede verificar que $F(x,y)$ es cero sobre la línea, positiva bajo la línea y negativa para puntos sobre la línea.

Midpoint Line Algorithm

- Necesitamos solamente calcular $F(M) = F\left(x_p + 1, y_p + \frac{1}{2}\right)$ y verificar el signo.
- Definimos una variable de decisión $d=F(x_p+1, y_p+1/2)$
- Si $d>0$, elegimos el pixel NE.
- Si $d<0$, elegimos el pixel E.
- Si $d=0$, podemos elegir cualquiera, por convención elegimos E.
- La siguiente posición de M y el valor de d dependerá del pixel elegido en la última iteración.

Midpoint Line Algorithm

- Si elegimos **E**, incrementamos **M** un paso en la dirección **x**:

$$d_{new} = F\left(x_p + 2, y_p + \frac{1}{2}\right) = a\left(x_p + 2\right) + b\left(y_p + \frac{1}{2}\right) + c.$$

pero

$$d_{old} = a\left(x_p + 1\right) + b\left(y_p + \frac{1}{2}\right) + c.$$

y restando $\mathbf{d}_{\mathbf{old}}$ de $\mathbf{d}_{\mathbf{new}}$ escribimos:

$$d_{new} = d_{old} + a = d_{old} + dy$$

Midpoint Line Algorithm

- Si elegimos **NE**, incrementamos **M** un paso en la dirección **x** y un paso en la dirección de **y**:

$$d_{new} = F\left(x_p + 2, y_p + \frac{3}{2}\right) = a\left(x_p + 2\right) + b\left(y_p + \frac{3}{2}\right) + c.$$

y restando $\mathbf{d_{old}}$ de $\mathbf{d_{new}}$ escribimos:

$$d_{new} = d_{old} + a + b = d_{old} + dy - dx$$

- Nótese que esta versión del algoritmo funciona para pendientes entre 0 y 1.

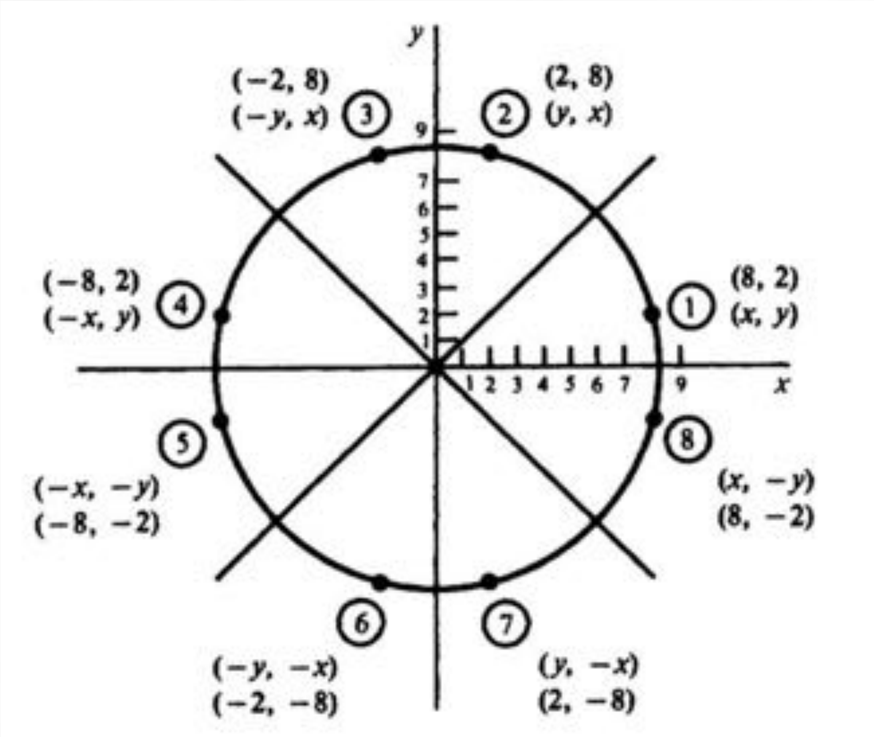
Midpoint Line Algorithm

```
void MidpointLine( int x0, int y0, int x1, int y1, int value )
{
  int dx = x1-x0;
  int dy = y1-y0;
  int d = 2*dy-dx;           /* valor inicial de d */
  int incrE = 2*dy;          /* incremento para E */
  int incrNE = 2*(dy-dx);    /* incremento para NE */
  int x = x0;
  int y = y0;

  WritePixel(x,y,value);     /* the start pixel */

  while( x < x1 ){
    if( d <= 0 ){          /* choose E */
        d += incrE;
        x++;
    } else {               /* choose NE */
        d += incrNE;
        x++;
        y++;
    }
    WritePixel(x,y,value);    /* the selected pixel */
  } /* while */
} /*MidpointLine */
```

Conversión de scan de un círculo



- Cualquier algoritmo para generar un círculo puede aprovechar de la simetría de la figura.
- Por cada punto calculado, se pueden graficar 8 reflejando en ejes a 45 grados.

$p_1 = (x, y)$ $\quad$ $p_5 = (-x,-y)$

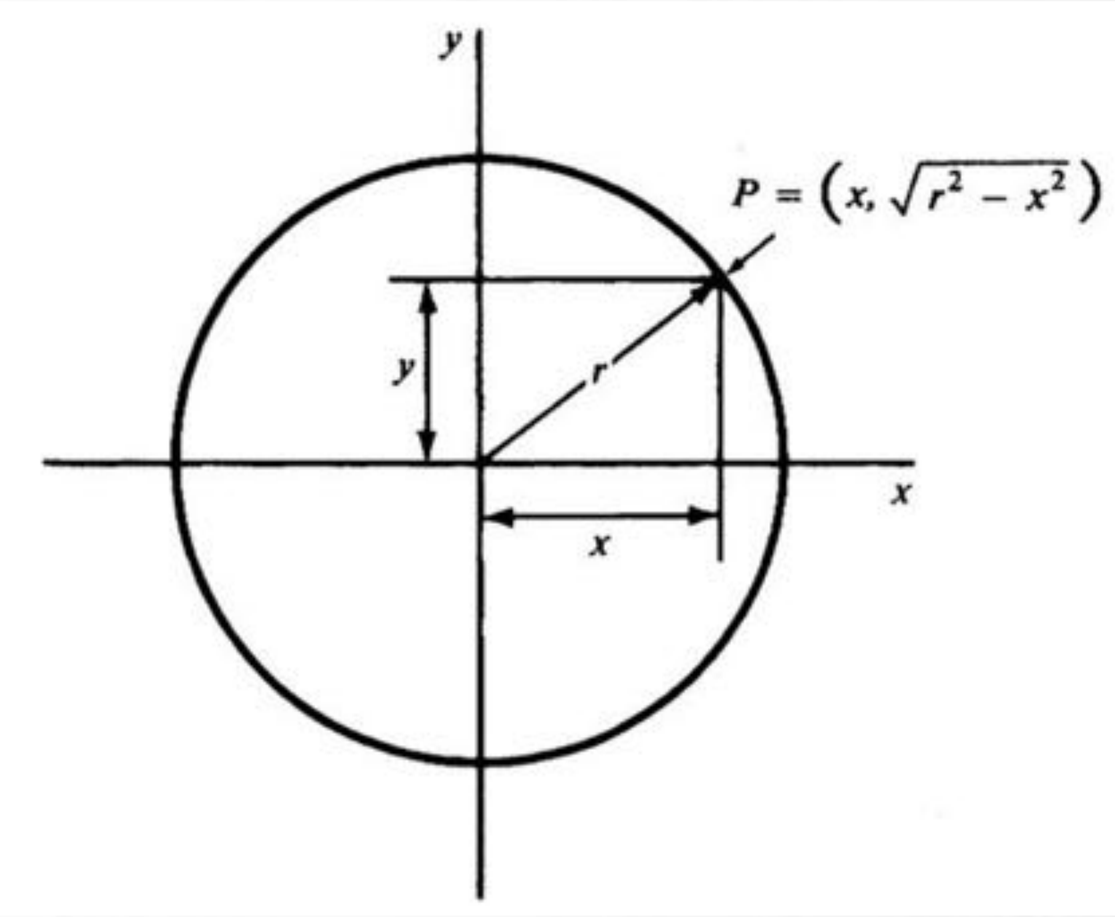
$p_2 = (y, x)$ $\quad$ $p_6 = (-y,-x)$

$p_3 = (-y, x)$ $\quad$ $p_7 = (y,-x)$

$p_4 = (-x, y)$ $\quad$ $p_8 = (x,-y)$

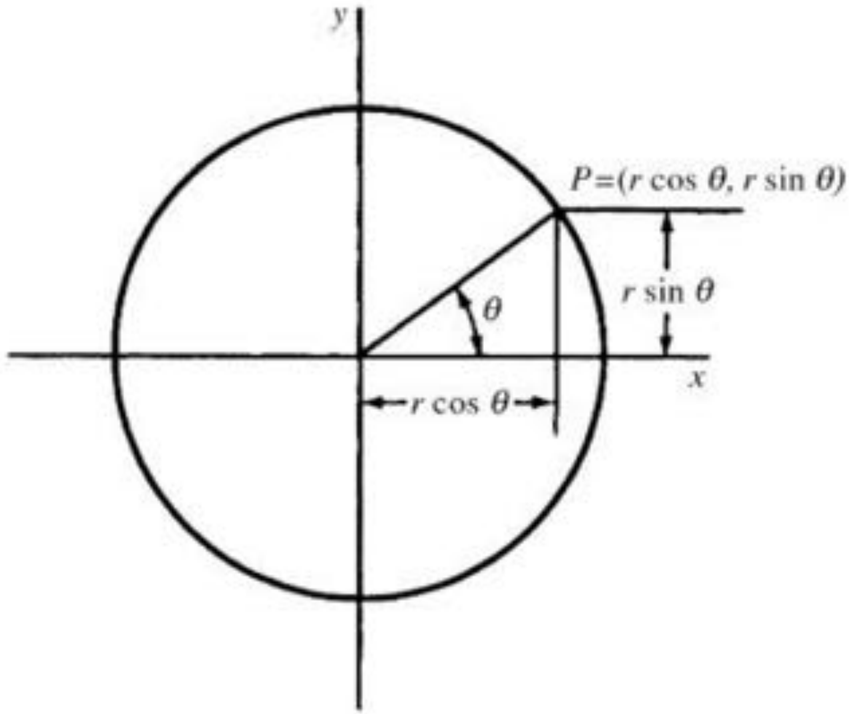
Conversión de scan de un círculo

- Definir matemáticamente un círculo centrado en el origen:
  - Polinomio de segundo orden: $y^2 = r^2 - x^2$.
  - Cada coordenada x en el sector de 90° a 45° se encuentra avanzando $r/\sqrt{2}$.
  - Cada coordenada y se encuentra evaluando $\sqrt{(r^2 - x^2)}$ para cada paso de x.
  - Ineficiente porque par cada punto hay cuadrados y raíces.



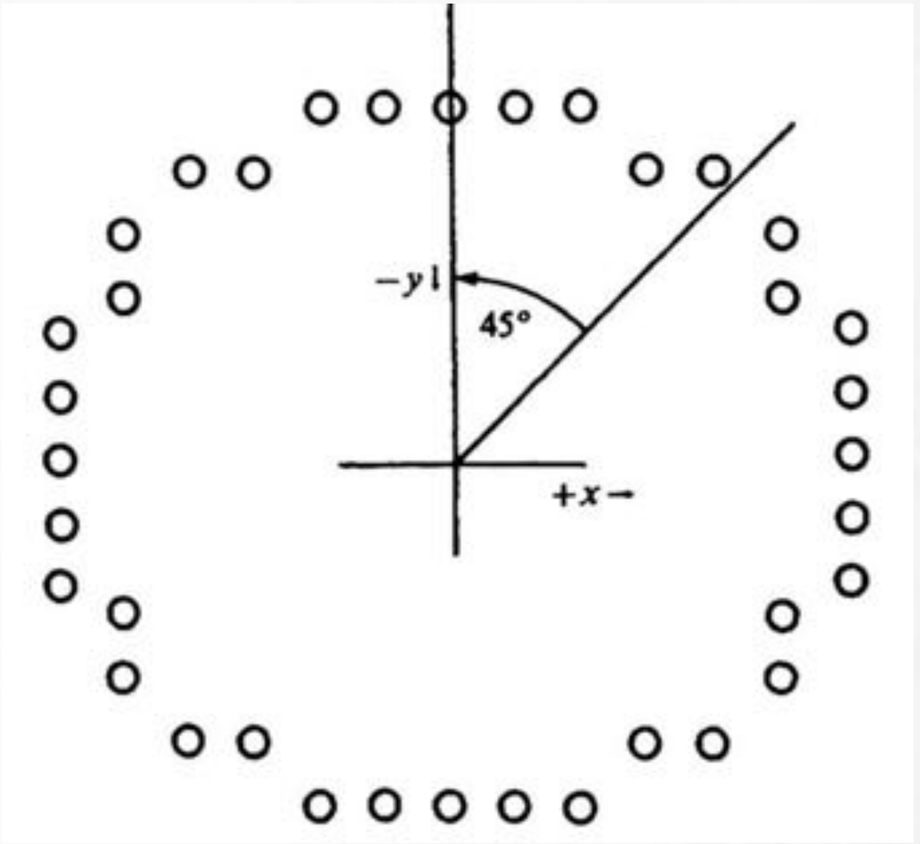
Conversión de scan de un círculo

- Definir matemáticamente un círculo centrado en el origen:
  - Usando funciones trigonométricas.
  - $x = r\cos\theta$, $y=r\sin\theta$.
  - Avanzamos θ de θ a $\pi/4$ y calculamos cada valor de x y y.
  - Aún más costoso que el método anterior.



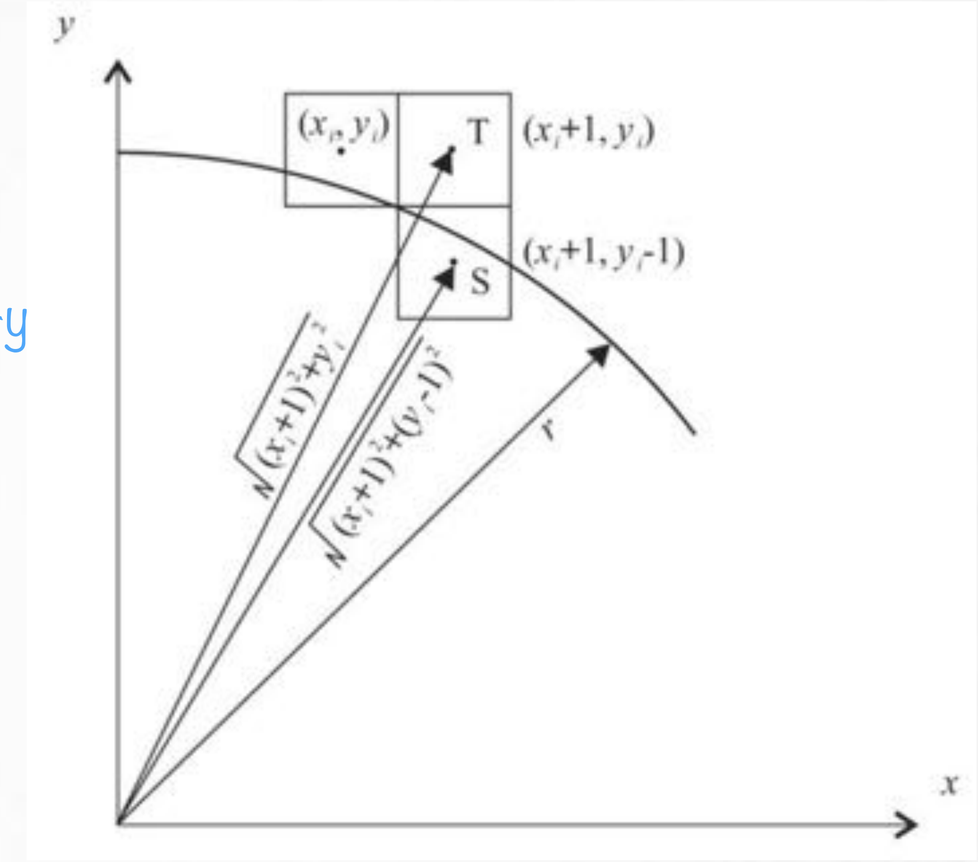
Algoritmo de Bresenham para círculos

- Aprovechar la simetría del círculo.
- Generar puntos solamente en un sector de 45 grados (de 90 a 45, esto es, en las direcciones +x y -y).



Algoritmo de Bresenham para círculos

- Dos posibles acciones:
  - moverse una unidad en dirección x.
  - moverse una unidad en dirección x y una unidad en dirección -y
  - (x_i,y_i) - coordenada del último pixel elegido al entrar al paso i.
  - D(T) = distancia del origen al pixel T al cuadrado menos la distancia del origen al círculo real al cuadrado.
  - D(S) = distancia del origen al pixel S al cuadrado, menos la distancia del origen al círculo real al cuadrado.



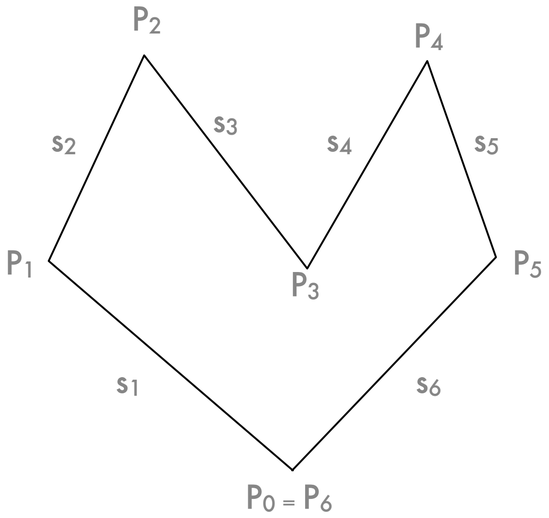
ALGORITMOS DE RELLENADO DE POLÍGONOS

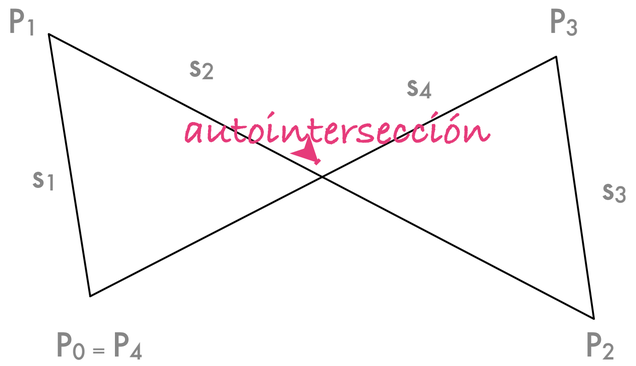
Computación Gráfica

Polígono en un plano

- Un polígono Π en un plano es una serie de segmentos $\Pi=(s_1,s_2,\dots,s_n)$ con $n\in\mathbb{N}^*$ y para $i=1,\dots,n$ el segmento si es de la forma $s_i=[P_{i-1},P_i]$ donde P_i es un punto del plano para $i=0,\dots,n$, con $P_n=P_0$.
- Los puntos P_i, con $i=0,..,n$ se llaman **vértices** del polígono Π.
- Los segmentos s_i se llaman **arístas** del polígono Π.
- Para $i=2,\dots,n$, el origen P_{i-1} del segmento s_i es igual al extremo del segmento s_{i-1}.
- El origen P_0 del segmento s_1 es igual al extremo P_n del segmento s_n.

Polígono en un plano

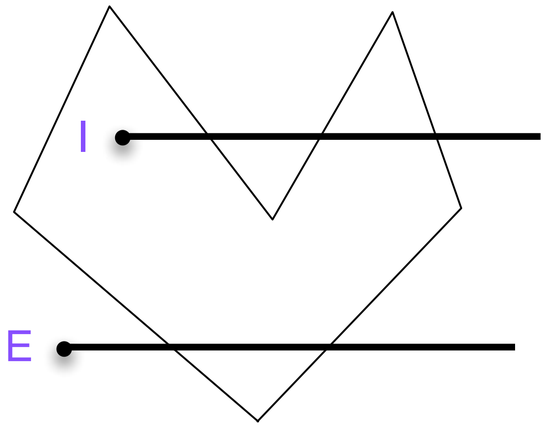


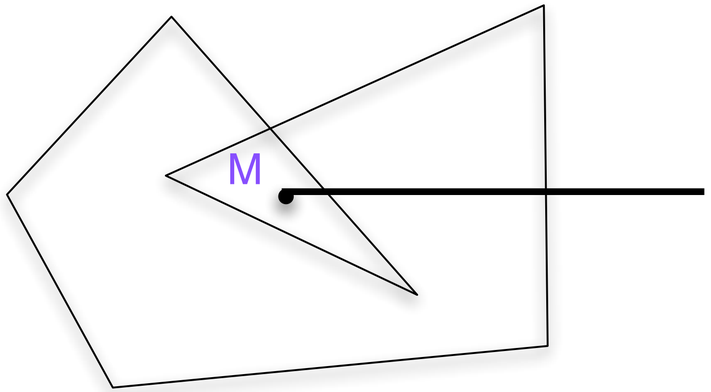


- Decimos que un polígono se auto-intersecta si dos segmentos que componen al polígono y que son no-consecutivos se intersectan.

Puntos interiores y exteriores

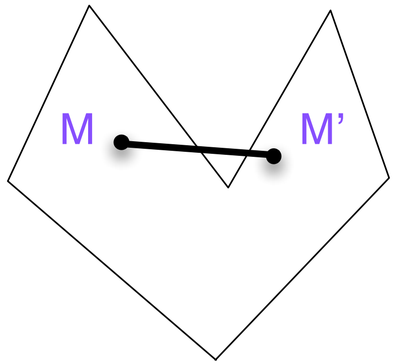
- Sea $\Pi=(s_1,s_2,...,s_n)$ un polígono y sea M un punto en el plano que no forma parte de ninguna de las aristas del polígono Π.
  - Consideramos una semirecta Δ que sale de M y no contiene a ninguno de los vértices del polígono Π.
  - Decimos que el punto M es interior al polígono Π si la semirecta Δ intersecta un número impar de segmentos del polígono Π.
  - Decimos que el punto M es exterior al polígono Π si la semirecta Δ intersecta un número par de segmentos del polígono Π.

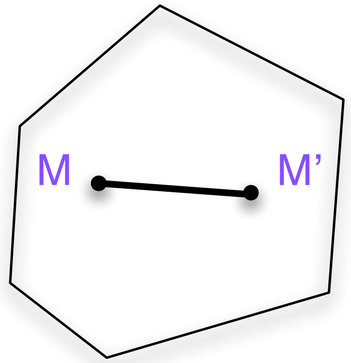




Polígonos convexos

- Un polígono se llama convexo si para todos los puntos M y M' interiores al polígono Π, el segmento [M,M'] está enteramente compuestos de puntos interiores al polígono Π.

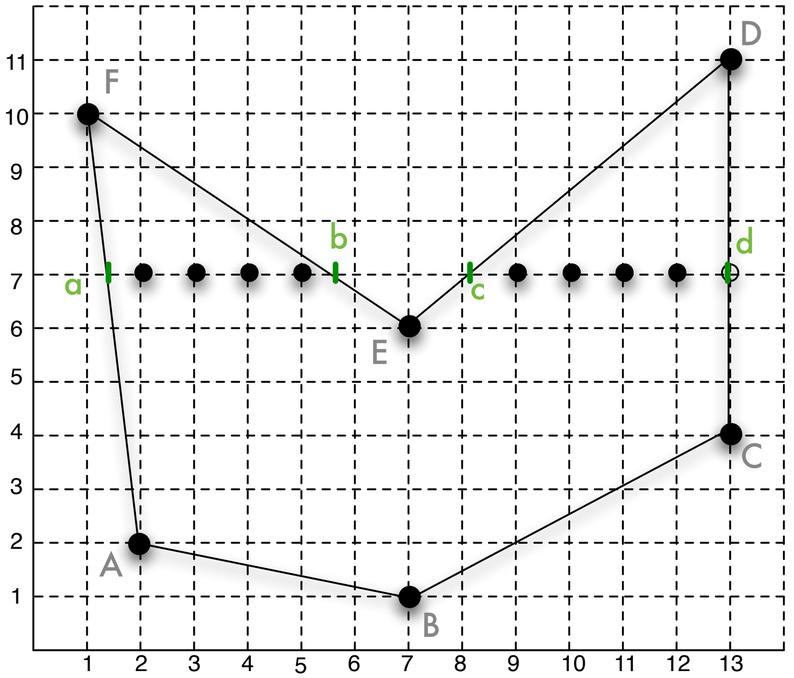




Principio de rellenado de polígonos

- Consideremos un polígono Π del plano, donde los vértices tienen coordenadas enteras.
- El problema de rellenado consiste en encontrar un algoritmo que permita recorrer todos los puntos de coordenadas enteras del polígono que sean interiores.

- Calcular los segmentos horizontales incluídos en el polígono.
- Determinar por cada horizontal cuáles son los pixels al interior del polígono para desplegarlos con su color apropiado.
- Haciendo eso de manera iterativa para todas las horizontales se rellenará el polígono.

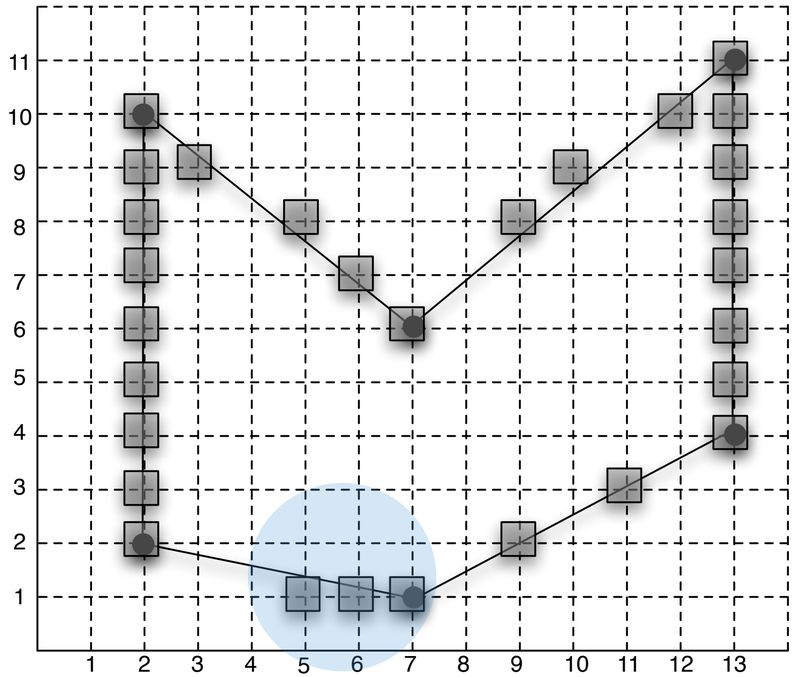


Rellenado de polígonos

- El rellenado, para cada línea horizontal se descompone en 3 etapas:
  - Encontrar las extremidades de los segmentos horizontales incluídos en el polígono: intersecciones de la línea horizontal con las aristas del polígono.
  - Ordenar los extremos obtenidos en orden creciente de las coordenadas *x*.
  - Desplegar todos los pixels al interior del polígono entre pares de extremidades. Utilizar regla de paridad para determinar si un pixel se encuentra al interior o al exterior de un polígono.

Recorrido de aristas

- ¿Cómo encontrar los extremos sobre cada horizontal?
  - rellenar cada arista con el algoritmo del punto medio.



- Algunos pixels están al exterior del polígono.
- Problema si dibujamos varios polígonos diferentes con diferentes colores.
- Podría mostrar polígonos disjuntos como juntos.
- No vamos a utilizar el algoritmo del punto medio.
- Algoritmo incremental: elegir los pixels a la derecha de la arista ideal si se trata de una arista izquierda y viceversa.
- Memorizamos los pixels extremo generados por este algoritmo en una tabla.