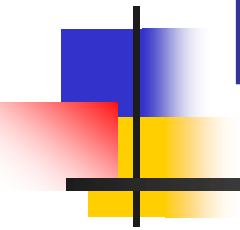
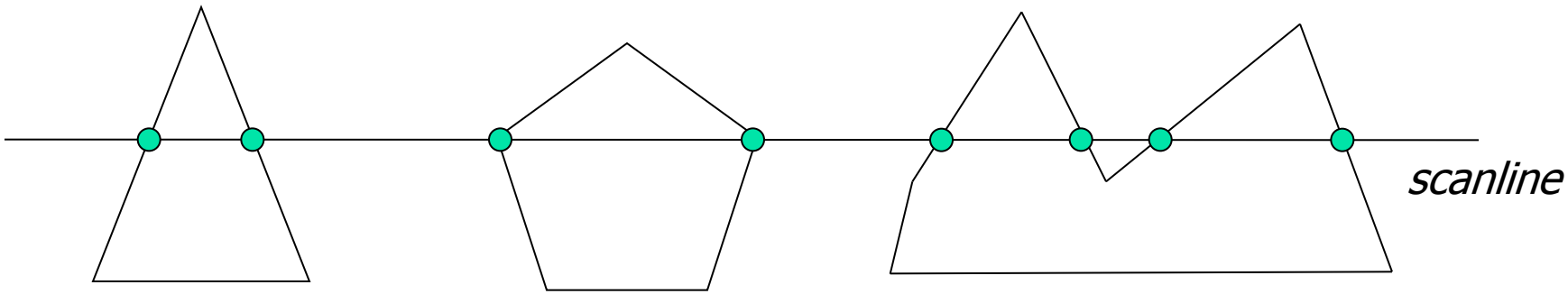


Relleno de Polígonos

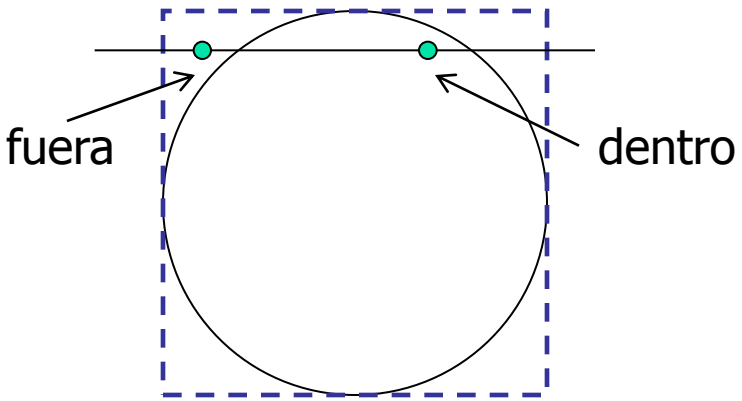
Relleno de Polígonos

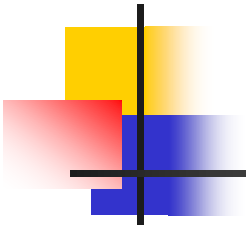
- El caso más sencillo de relleno es el del triángulo
- Luego le sigue en complejidad el de polígonos convexos de N-lados
- Finalmente le sigue el relleno de polígonos cóncavos



Relleno de Polígonos

- Método de fuerza bruta (*naive approach*)
  - Calcular caja contenedora del objeto
  - Hacer un barrido interno de la caja, comprobando para c/pixel si está dentro o no del polígono





Relleno de Polígonos

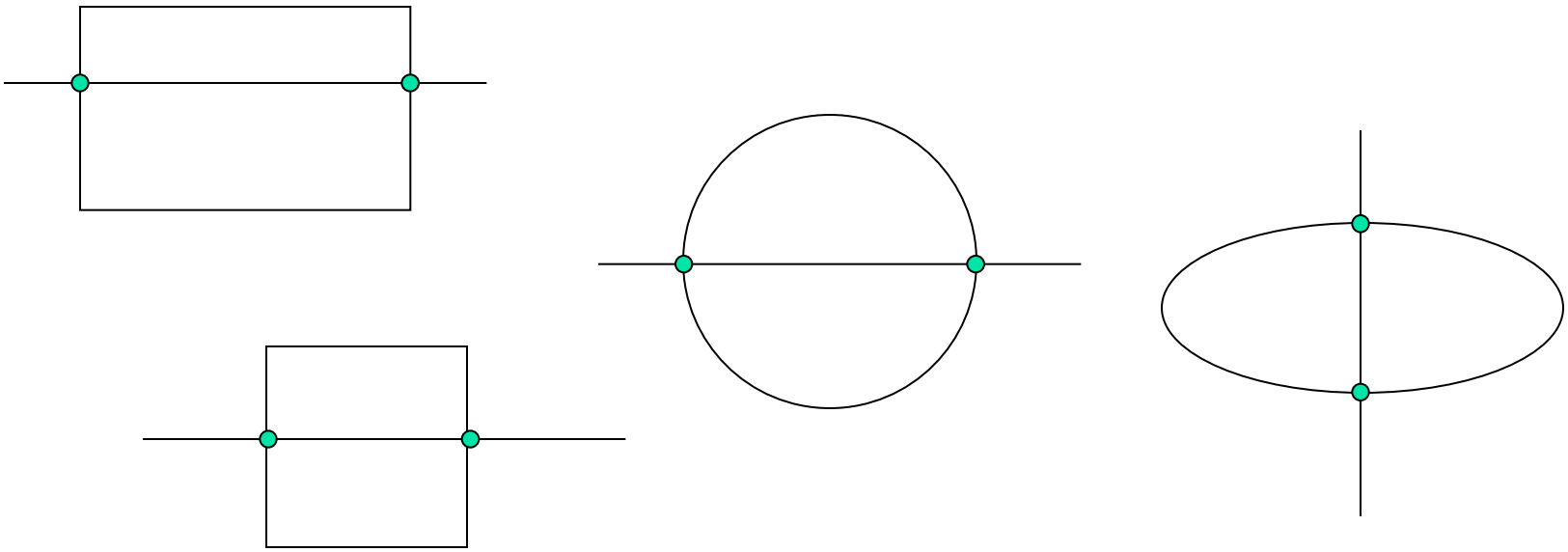
- Método de fuerza bruta
  - Con polígonos simétricos basta con hacer un barrido en una sección y replicar los demás píxeles
  - Requiere aritmética punto-flotante, lo que lo hace impreciso y costoso

Algoritmo ScanLine

- También llamado *scan conversion algorithm*
- Detectar la intersección de los *scanlines* del dispositivo con los bordes de la primitiva
- Por cada *scanline*, rellenar el *span* de píxeles entre cada par de intersecciones

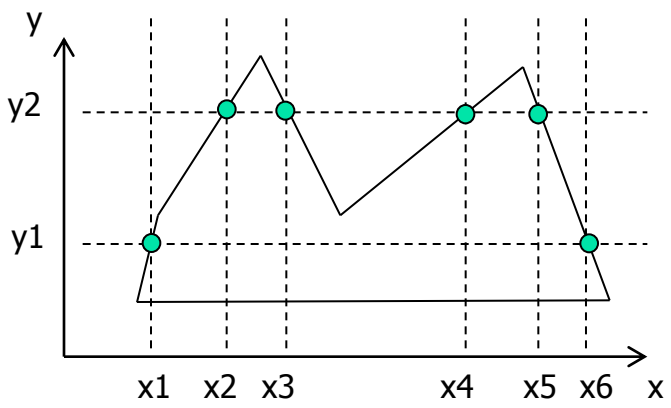
Algoritmo ScanLine

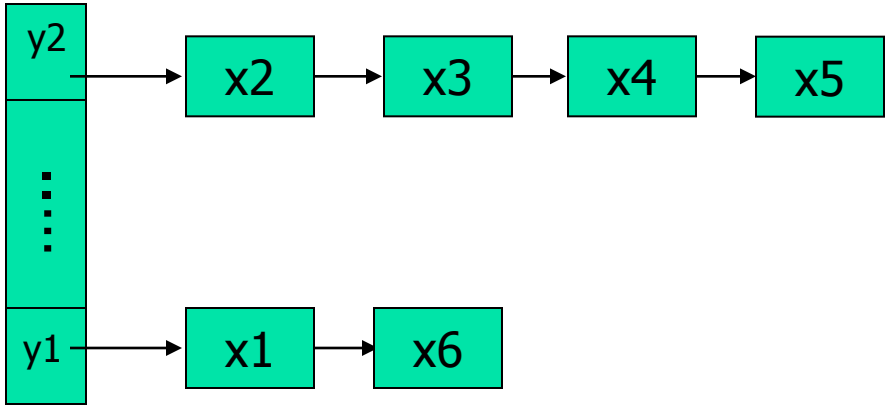
- Cuando el polígono es convexo, como las primitivas que vimos aquí, solo hay dos intersecciones por *scanline*, en cualquier dirección



Algoritmo ScanLine

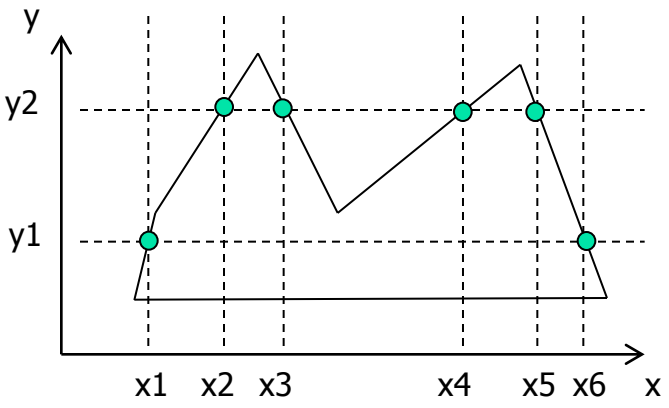
- Encontrar las intersecciones de los *scanlines* con el polígono
- Almacenar las intersecciones en alguna estructura de datos ET (*Edge Table*), de forma ordenada, ascendentemente en *y* y en *x*, en *buckets*
- Rellenar los *spans* usando la estructura

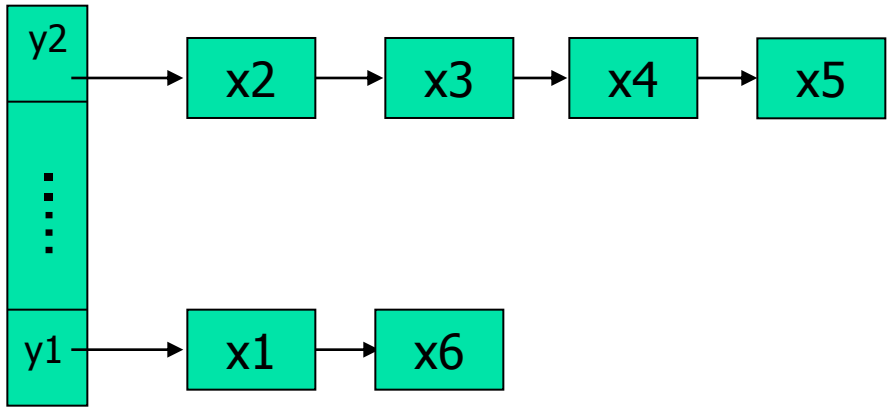




Algoritmo ScanLine

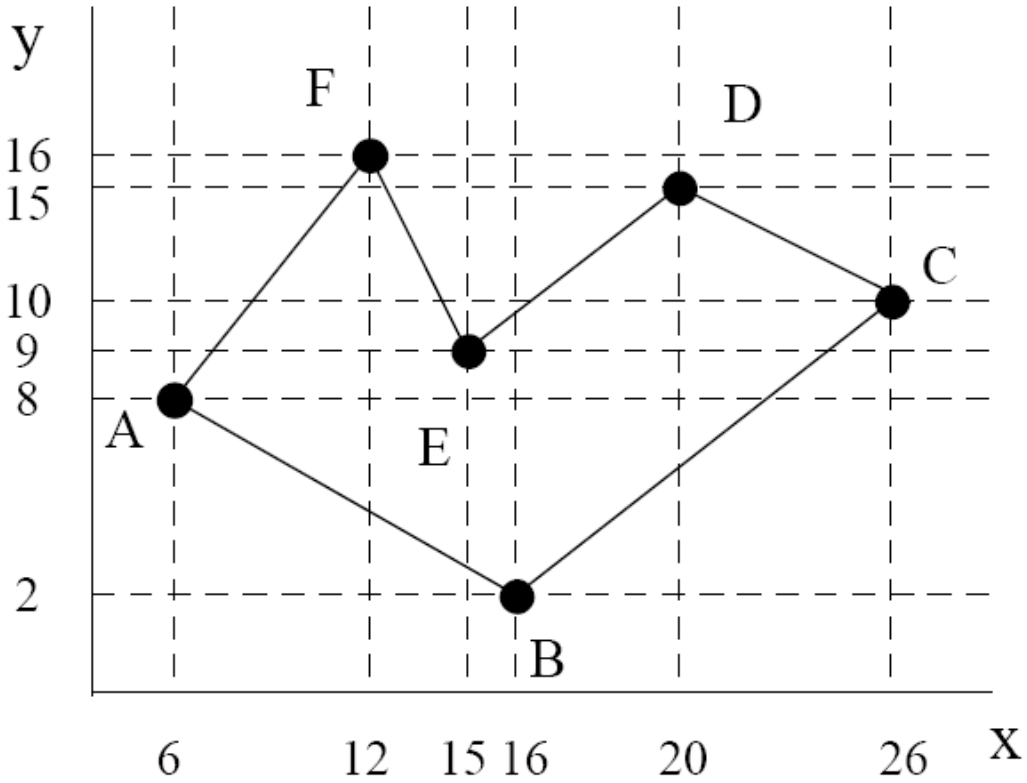
- Utilizar algún criterio de *paridad* para saber cuando un intervalo debe ser rellenado o no
  - Ej: (x2, x3) debe ser rellenado; (x3 ,x4) no ; (x4, x5) si debe ser rellenado





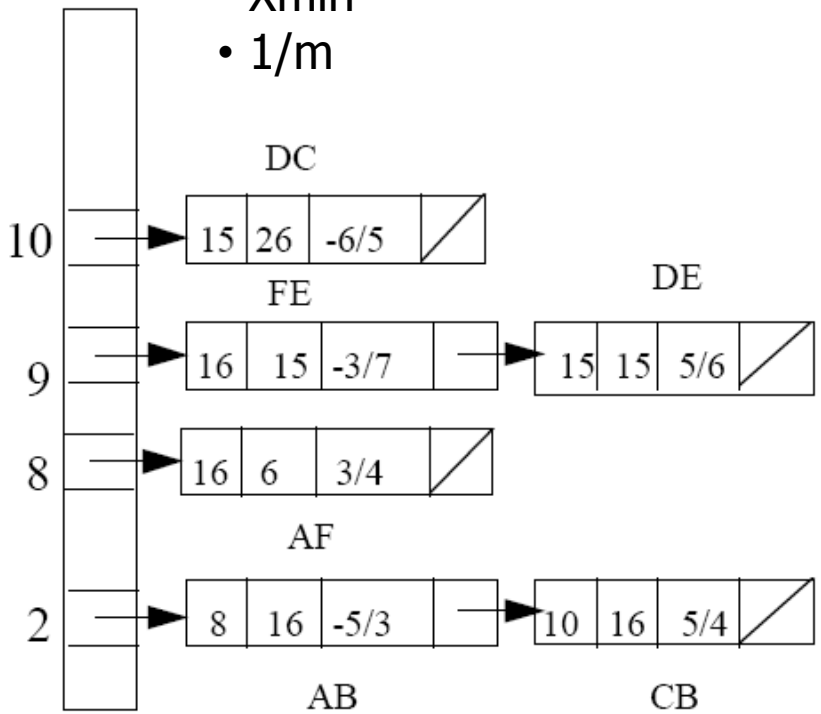
Algoritmo ScanLine

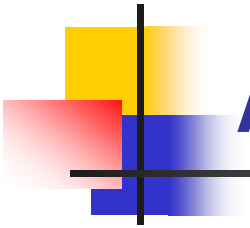
- Un ejemplo más concreto …



Cada *bucket* contiene:

- Ymax
- Xmin
- 1/m



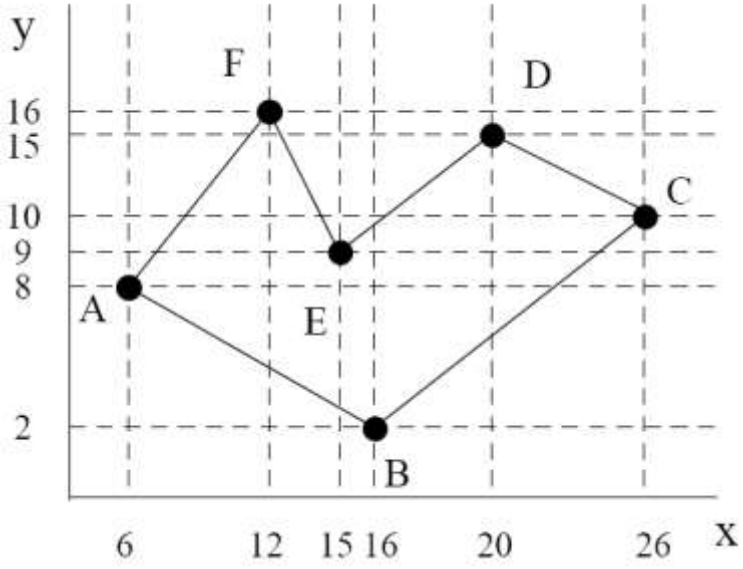


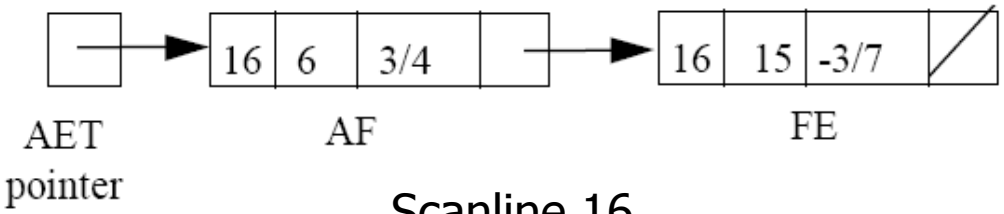
Algoritmo ScanLine

- El método de fuerza bruta para calcular las intersecciones sería con ecuaciones punto-flotante, probando cada *scanline* contra los bordes de la primitiva
- En lugar de ello, las intersecciones se pueden calcular de forma incremental, de forma similar al algoritmo del punto medio
- Para hacer esto eficientemente, se puede utilizar una *tabla de bordes activos* (AET – *Active Edge Table*), que no es mas que otra lista de *buckets*
- Es importante no salirse de los bordes de la primitiva, debido a problemas de redondeo de las coordenadas

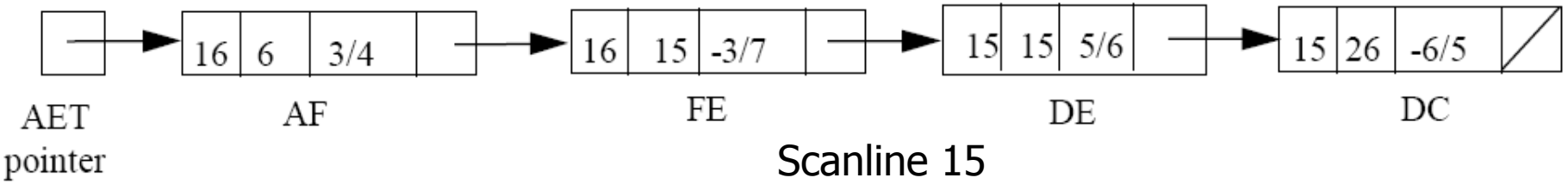
Algoritmo ScanLine

- Para los scanlines 15 y 16 del ejemplo anterior ...





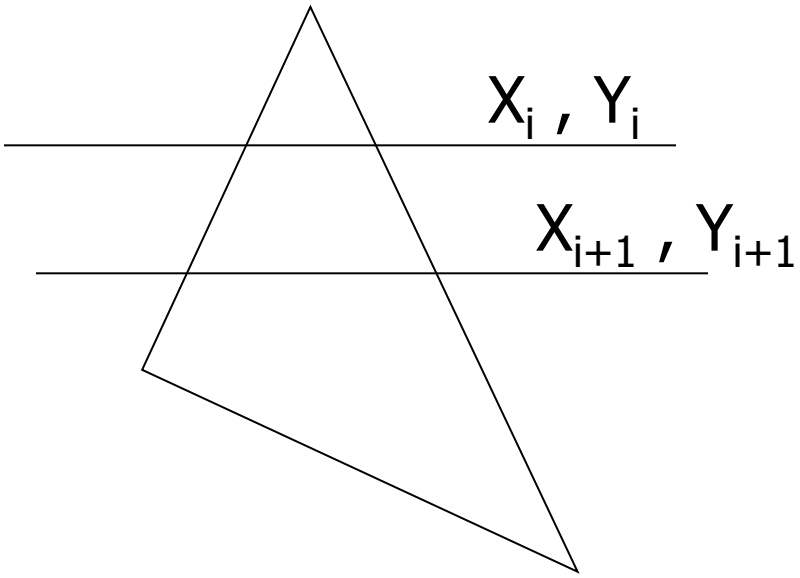
Scanline 16



Scanline 15

Algoritmo ScanLine

- Para el calculo de las intersecciones se utiliza un algoritmo incremental



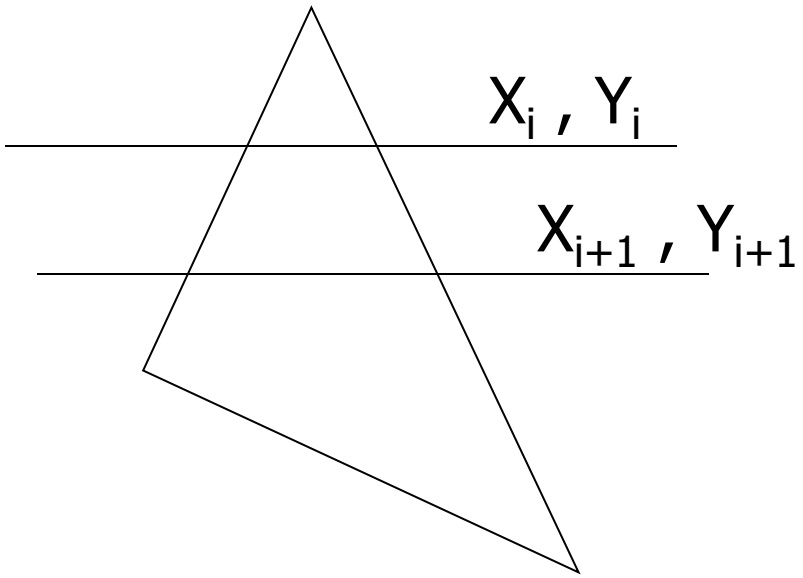
$$Y_{i+1}=Y_i+1$$
$$X_{i+1} = X_i + 1/m = X_i + dx/dy$$

Para m > 1

- 1/m ocasiona que X pueda tener una parte fraccional …..

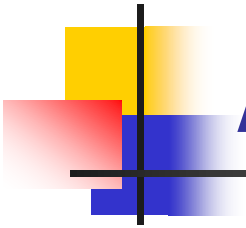
Relleno de Polígonos

- Para rellenar un polígono entre el intervalo real (Xmin, Xmax) para un *scanline* especifico, se despliegan los píxeles entre *ceiling(Xmin*) y el *floor(Xmax*)



$$Y_{i+1}=Y_i+1$$
$$X_{i+1} = X_i + 1/m = X_i + dx/dy$$

Para m > 1



Algoritmo ScanLine

- Se puede seguir las siguientes reglas para llevar el calculo de X a aritmética entera
  - Cuando la parte fraccional de X es cero: se dibuja el pixel (x,y) que está en la línea
  - Cuando la parte fraccional de X es mayor que cero pero menor que uno: se redondea hacia un X mayor para obtener el pixel que esta estrictamente dentro de la línea (hacia la derecha).
  - Cuando la parte fraccional de X se hace mayor que uno: se incrementa X en uno y se resta uno a la parte fraccional. Se aplica a la nueva especificación de X la consideración de fracciones menores a uno pero mayores a cero.

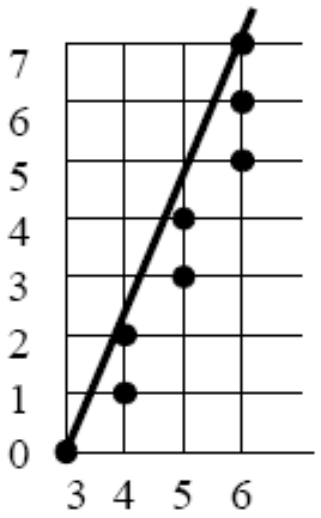
Algoritmo ScanLine

- Ejemplo: si el Xmin de la linea es 0 y la pendiente es 7/3 → dx = 3 , dy =7

x	pixel
0	0
3/7	1
6/7	1
9/7 = 1 2/7	2
1 5/7	2
1 8/7 = 2 1/7	3
2 4/7	3
2 7/7 = 3	3

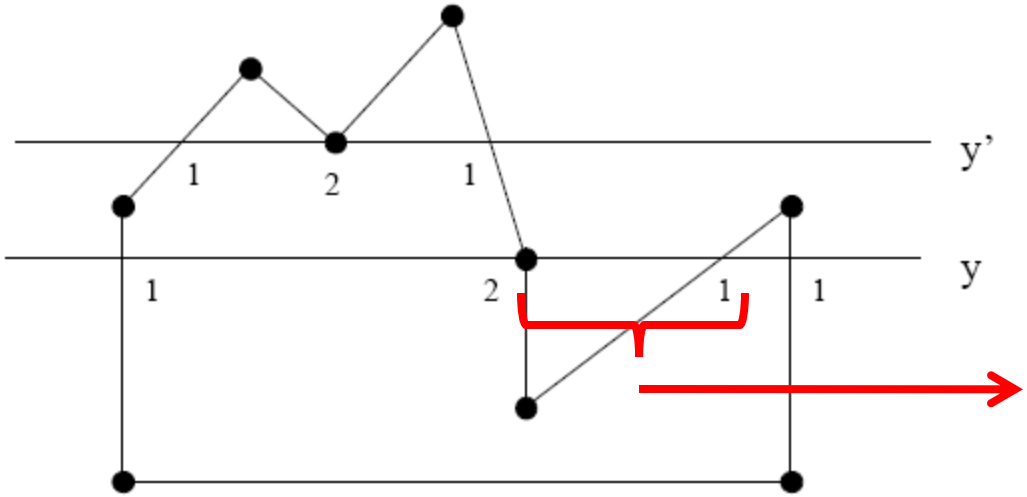
$Y_{i+1}=Y_i+1$

$X_{i+1} = X_i + 1/m = X_i + dx/dy$



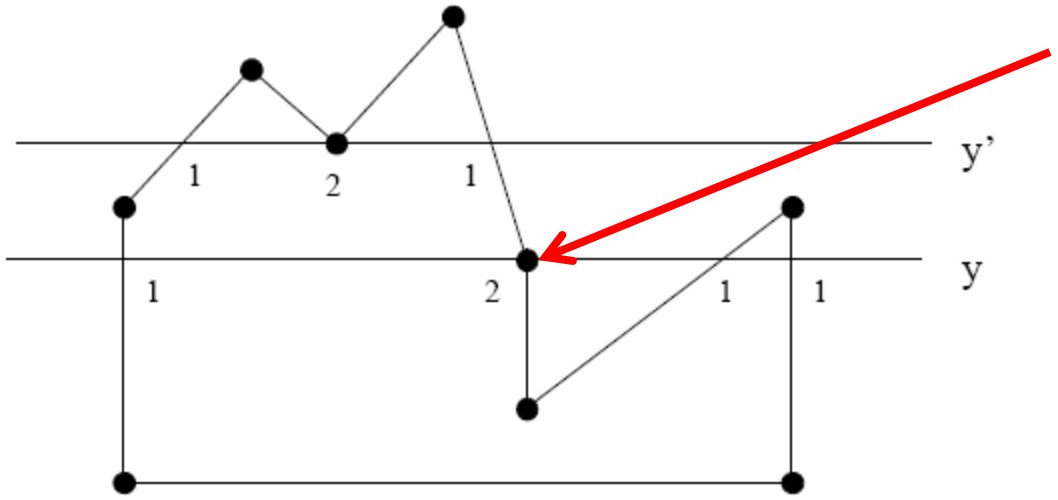
Algoritmo ScanLine

- Manejo de vértices
  - Una línea de rastreo que pasa a través de un vértice intersecta dos aristas del polígono en esa posición, agregando dos puntos a la lista de intersecciones para la línea de rastreo.
  - Esto puede causar problemas en la paridad
- Las líneas horizontales también pueden ser un problema, porque producen múltiples intersecciones



Algoritmo ScanLine

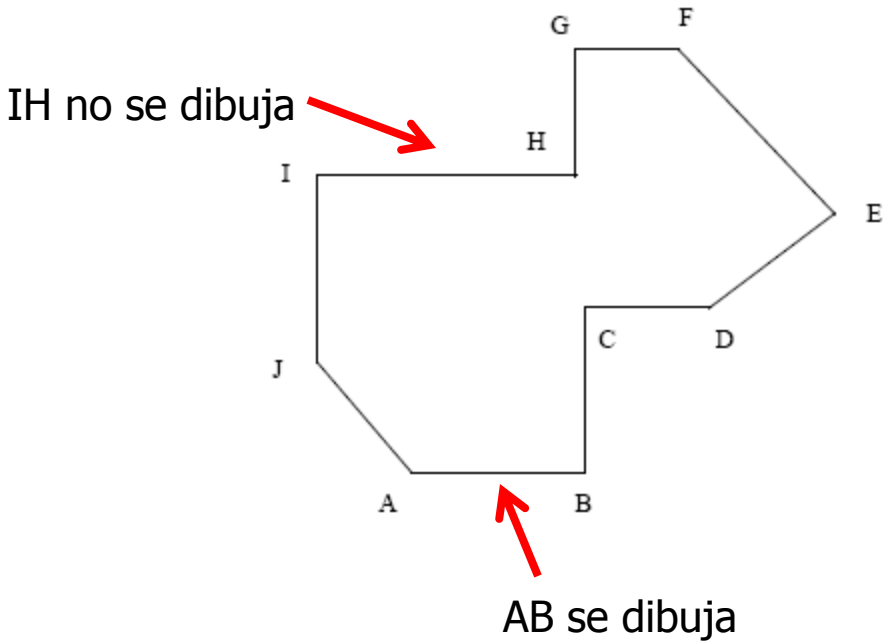
- ¿ Solución ? Aplicar la siguiente política
  - Se cuenta el vértice ymin de una arista en el cálculo de paridad pero no el vértice ymax; por lo tanto, un vértice ymax se dibuja solo si equivale al vértice ymin de la arista adyacente.



Con esta política este vértice solo se cuenta una vez, solucionando el problema

Algoritmo ScanLine

- Con esta misma política, en el caso de aristas horizontales, el efecto deseado es que las aristas de abajo se tracen pero no las de arriba. Esto ocurrirá de forma automática, si no se cuentan los vértices de estas aristas, ya que no son ni ymin, ni ymax



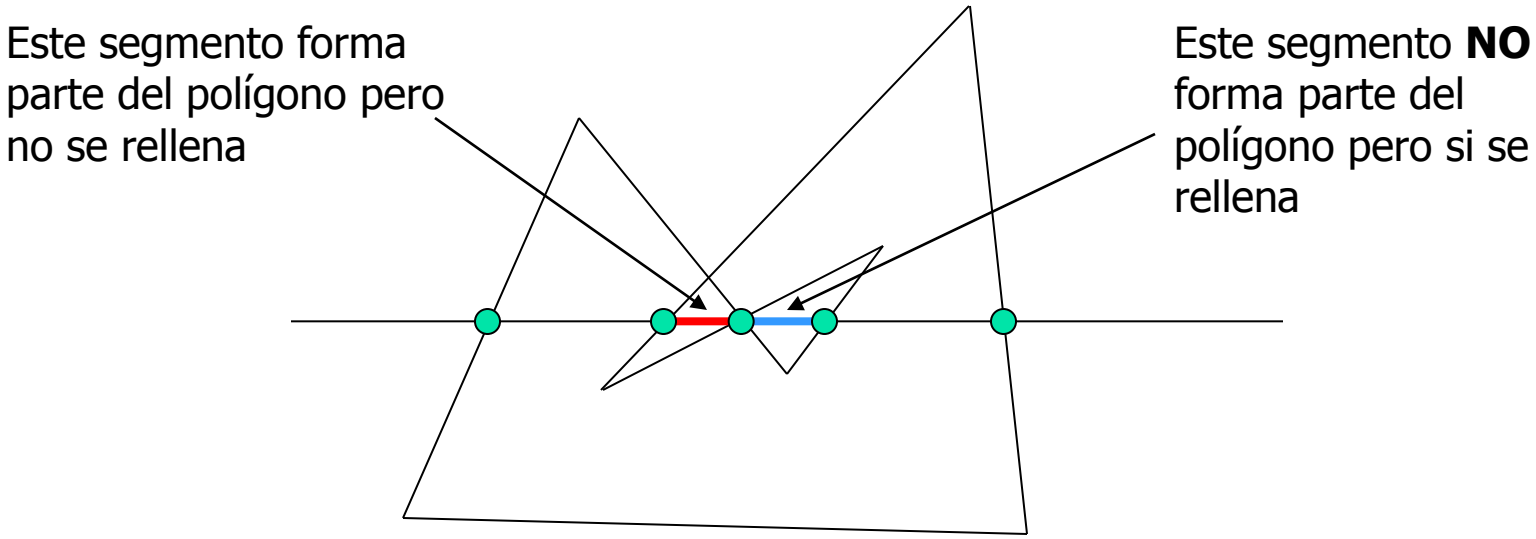
Algoritmo ScanLine

- El algoritmo queda

1. Asignar la primera línea de rastreo *y* a la coordenada *y* mas pequeña que tenga una entrada en el ET, i.e. *y* para el primer bucket no vacío.
2. Inicializar AET para que este vacío.
3. Repetir hasta que AET y ET estén vacíos:
   3.1 Mover de ET al AET los buckets de aquellas aristas cuyos *ymin* = *y* (aristas entrantes), luego ordenar el AET en *x*
   3.2 Pintar los pixel deseados en la línea de rastreo, usando los *x* de AET
   3.3 Quitar del AET aquellas entradas cuyos *ymax* = *y* (aristas no involucradas en la línea de rastreo siguiente).
   3.4 Incrementar *y* en 1 (a la coordenada de la siguiente línea de rastreo).
   3.5 Para cada arista no-vertical en el AET, actualizar los *x* para el nuevo *y*.

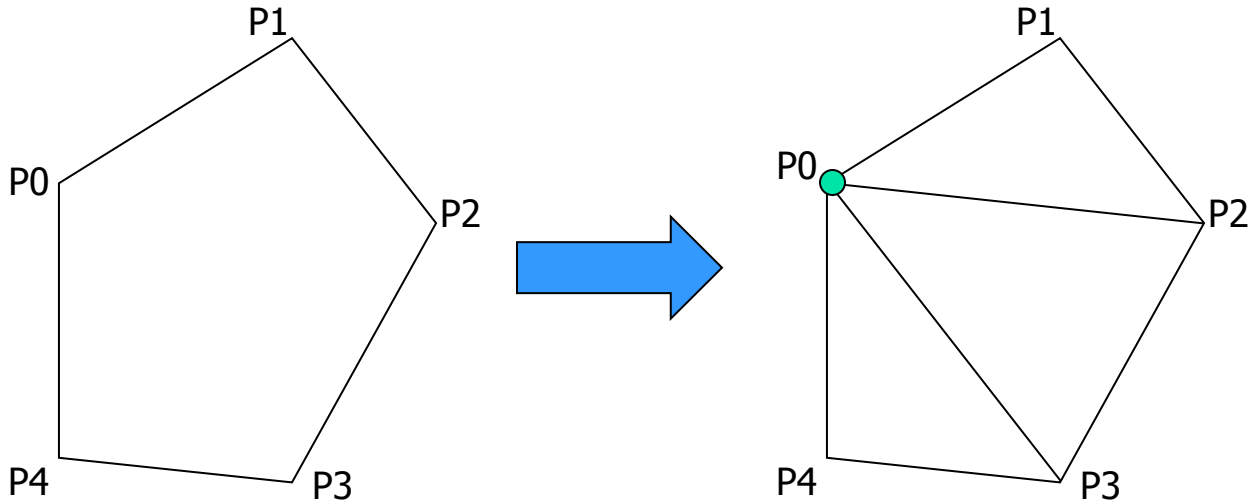
Algoritmo ScanLine

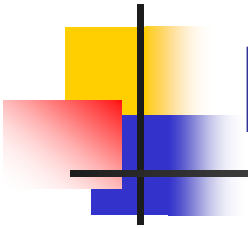
- Este método funciona para polígonos convexos o cóncavos
- Pero no funciona si el polígono se intersecta a si mismo, porque el algoritmo se confunde
- Dicho caso no se tomará en cuenta en este curso



División en Triángulos

- Otra forma de rellenar un polígono es dividirlo en triángulos, y reducir el problema al relleno de secuencias de triángulos
- La subdivisión en triángulos de un polígono convexo es sencilla
- Se escoge un punto cualquiera y se construyen triángulos desde este punto hasta cada par de vértices consecutivos que no incluyan este punto
- Este método no funciona para polígonos cóncavos



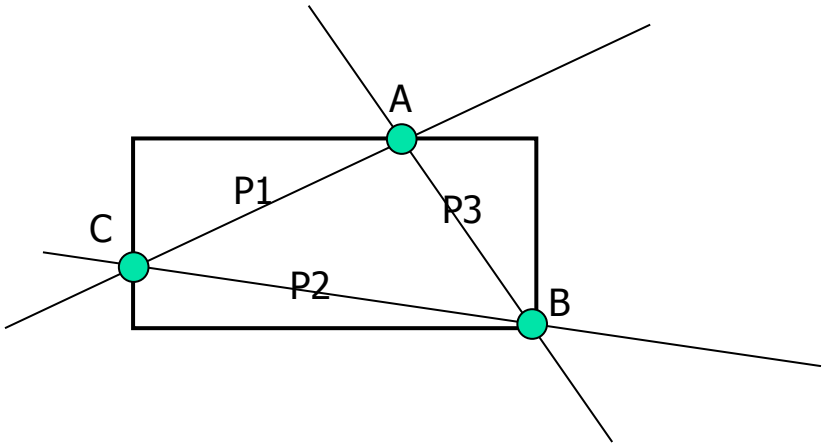


División en Triángulos

- El siguiente método funciona para polígonos cóncavos o convexos
- La idea general es formar triángulos y probar si ningún vértice del polígono queda dentro de un triángulo
- Se comienza desde el triángulo más a la izquierda
  - Encontrar vértice más a la izquierda (con el *x* más pequeño), y llamarlo *A*
  - Formar un posible triángulo entre *A* y dos vértices adyacentes *B* y *C*
  - Comprobar que ningún otro vértice queda dentro de *ABC*
  - Si todos los otros puntos quedan fuera de ABC, se "separa" el triángulo del polígono y se procede con el próximo vértice más a la izquierda
  - Si un punto esta dentro del triángulo, se forma un nuevo triángulo de prueba con A y el punto **interno** más a la izquierda

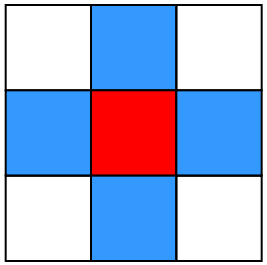
División en Triángulos

- Para comprobar que un punto está dentro de un triángulo *ABC* se utiliza la prueba estándar de comprobar la posición del punto con respecto a los tres planos *P1*, *P2* y *P3* definidos por los lados del triángulo
- Sin embargo, para acelerar la comprobación se puede comprobar la posición del punto con respecto a la caja contenedora del triángulo
- Si el punto pasa esta primera prueba entonces se comprueba contra los lados del triángulo

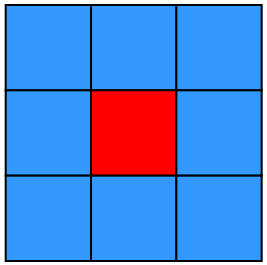


Relleno por inundación

- Otra solución es el *Flood-Fill Algorithm* (FFA) o Algoritmo de Relleno por Inundación
  - Se comienza desde un píxel dentro del polígono, al cual se le asigna el color de relleno
  - El color se propaga desde este hacia sus píxeles vecinos, y de estos a sus vecinos y así sucesivamente, simulando una inundación dentro del polígono
  - La propagación continúa hasta encontrar los píxeles del borde, o simplemente encontrar píxeles de un color diferente al color de la semilla inicial
  - Se pueden usar dos patrones distintos para hacer la propagación



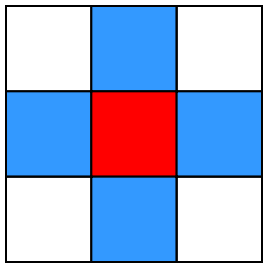
4-Fill



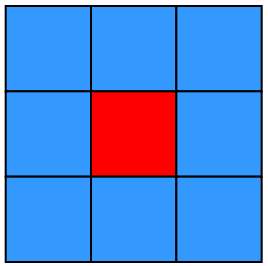
8-Fill

Relleno por inundación

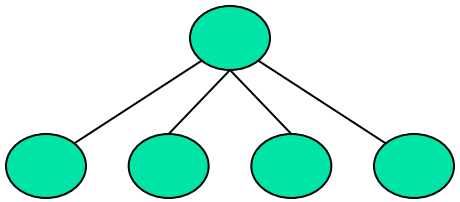
- El *Flood-Fill Algorithm* (FFA) o Algoritmo de Relleno por Inundación es un algoritmo inherentemente recursivo
  - ¿ Es posible una implementación iterativa ?
  - ¿ Ventajas y desventajas de escoger una vecindad 4-Filll o 8-Fill ?

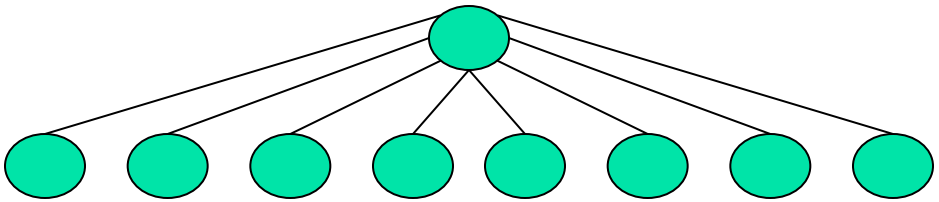


4-Fill



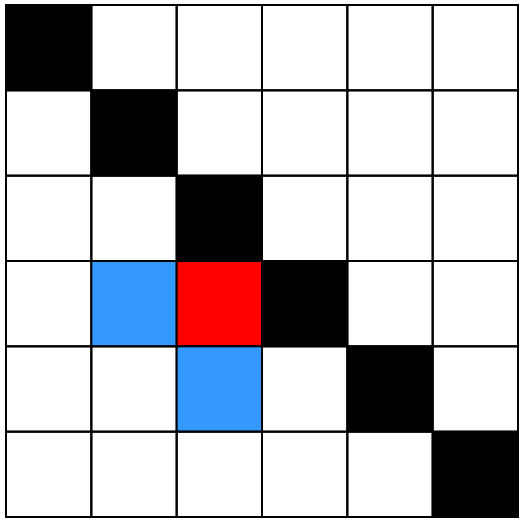
8-Fill



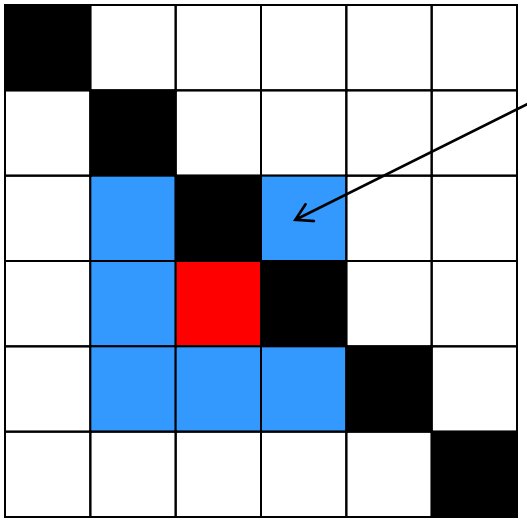


Relleno por inundación

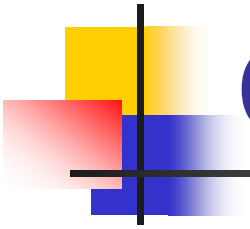
- El *Flood-Fill Algorithm* (FFA) o Algoritmo de Relleno por Inundación es un algoritmo inherentemente recursivo
  - ¿ Ventajas y desventajas de escoger una vecindad 4-Filll o 8-Fill ?



4-Fill



8-Fill



Comparación

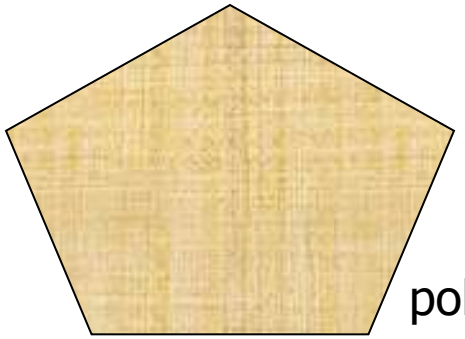
Inundación	ScanLine
Muy Simple	Más complejo
Algoritmo discreto en espacio de despliegue (*screen*)	Algoritmos discreto en espacio de despliegue (*screen*) o de objeto
Requiere llamada al sistema GetPixelVal()	Independiente del dispositivo
Requiere una semilla	No requiere una semilla
Requiere una pila amplia	Requiere una pila pequeña
Común en paquetes para pintar imágenes (*paint bucket*)	Usada en el render de imágenes

Relleno con patrones

- Alternativamente el relleno puede realizarse siguiendo un patrón (*pattern filling*)
- El patrón puede ser un matriz de NxN con un patrón
- El patrón se distribuye a lo largo del polígono, usando un apuntador a columna y un apuntador a fila del patrón
- Para pintar el píxel (x,y) entonces se utiliza

PaintPixel(x, y, Patron[rowptr,colptr])

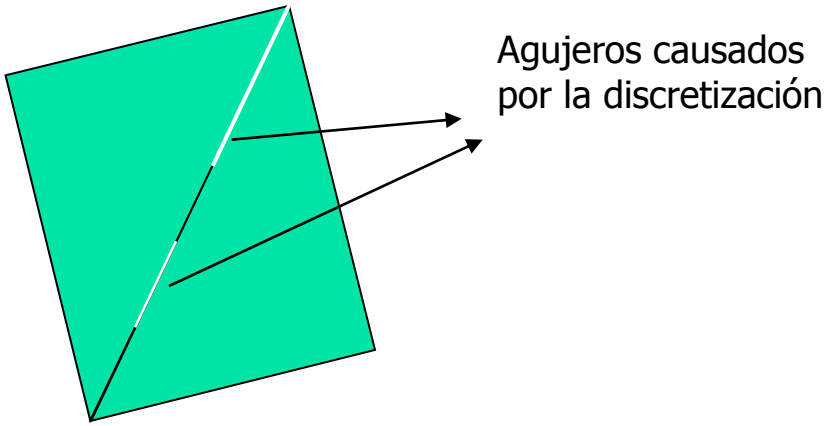
patrón



polígono

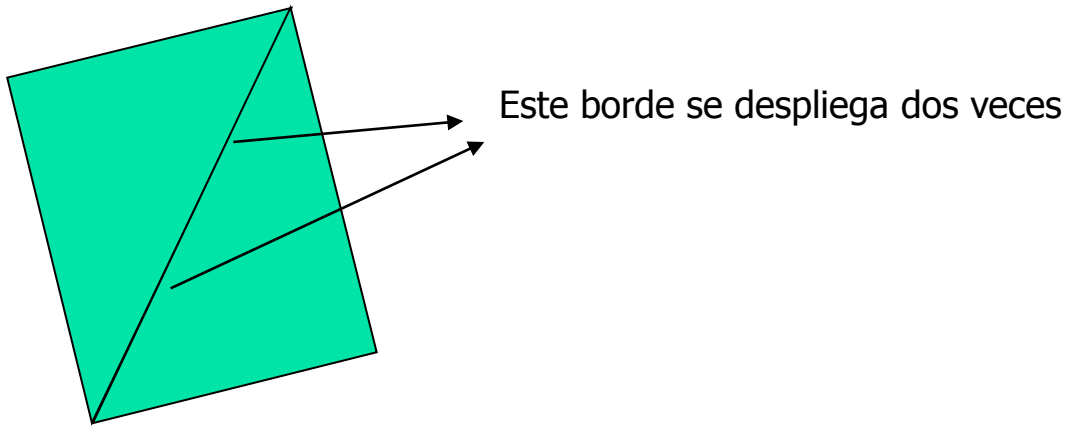
Relleno de Polígonos

- Posibles problemas
  - Dibujar bordes compartidos
  - En algunos casos, la discretización de las coordenadas puede ocasionar artefactos
  - Ejemplo: Series de triángulos. ¿Solución?.



Relleno de Polígonos

- Posibles problemas
  - Dibujar bordes compartidos
  - Redespliegue de los pixeles de los bordes
  - Ejemplo: Series de triángulos. ¿Solución?.



Relleno de Polígonos

- Posibles problemas
  - Astillas (*Slivers*).
  - Se encienden muy pocos pixeles → *aliasing*
  - ¿Solución?