2. Primitivos de Llenado de Area

La tarea de los primitivos de llenado se puede separar en dos partes:
1. la decisión de que pixeles llenar (esto depende de la forma de la primitiva), y
2. la decisión mas sencilla de cual valor utilizar para el relleno.

En general, determinar que pixeles llenar consiste de tomar líneas de rastreo sucesivas que intersectan la primitiva y llenar en intervalos (*spans*) de pixeles adyacentes que están dentro de la primitiva de izquierda a derecha.

Para llenar un rectángulo con un color sólido, se asigna a cada pixel sobre una misma línea de rastreo desde el borde izquierdo al borde derecho el mismo valor de pixel; o sea llenamos cada intervalo de x_{min} a x_{max}.
Se aprovecha de varios tipos de coherencias no solamente para convertir primitivas de 2D, pero también de 3D.

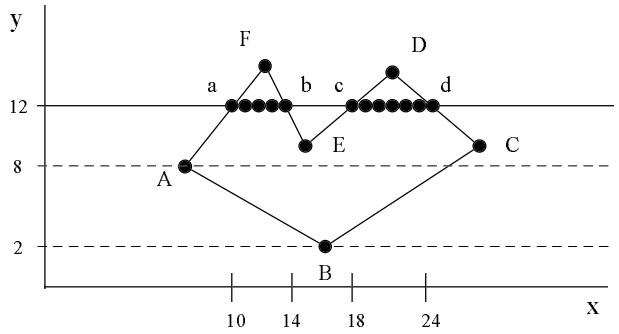
- Los intervalos explotan la **coherencia espacial (spatial coherence)** de una primitiva: el hecho que las primitivas a menudo no cambian de pixel en pixel dentro de un intervalo o de línea de rastreo a línea de rastreo. Se explota esta coherencia buscando solo aquellos pixeles donde ocurren cambios.
- Para una primitiva trazada de forma sólida, se asigna el mismo valor a todos los pixeles en un mismo intervalo, proporcionando **coherencia de intervalo (span coherence)**.
- Un rectángulo trazado de forma sólida también muestra una fuerte **coherencia de línea de rastreo (scan-line coherence)** ya que líneas de rastreo consecutivas que intersectan el rectángulo son idéntica; mas tarde se usa también **coherencia de arista (edge coherence)** para los lados de polígonos generales.

La capacidad de manejar múltiples pixeles en un intervalo de forma idéntica es especialmente importante porque se debe escribir el frame buffer una palabra a la vez para minimizar el numero de accesos a memoria.

Conceptos del algoritmo de relleno de áreas

El algoritmo presentado a continuación considera polígonos cóncavos al igual que convexos, incluyendo aquellos que puedan tener huecos internos o intersectados por si mismos.

La siguiente figura ilustra el procedimiento de la línea de rastreo para el llenado sólido de polígono.



Para cada línea de rastreo que cruza un polígono el algoritmo de llenado de áreas debe:
- localizar los puntos de intersección de la línea de rastreo con las aristas del polígono,
- definir los intervalos de llenado (**spans**) para cada línea de rastreo.

Esto se repetiría para todas las líneas de rastreo que intersectan el polígono.

Algoritmo de punto medio para lineas aplicado a relleno de areas

Una forma de derivar las intersecciones (extremos) de los intervalos es utilizar el algoritmo de punto medio para conversión de líneas para cada arista y mantener una tabla de extremos de intervalos para cada una de las línea de rastreo. En el ejemplo, los dos intervalos están definidos por las cuatro posiciones de intersección de pixel con las fronteras de polígono x=10 a x=14.5 y de x=18.5 a x=24.

Esta estrategia puede produce algunos pixeles extremos que estén fuera del polígono; ya que fueron escogidos por el algoritmo de conversión de rastreo porque están mas cerca de la arista, sin importar de que lado de la arista se encuentran. El algoritmo de punto medio no tiene la noción de interior o exterior. No quisiéramos trazar los pixeles del lado externo de una arista compartida, ya que se introducirían en las regiones de polígonos vecinos, y esto se vería raro si los polígonos tuvieran diferentes colores. Es por lo tanto preferible dibujar solo aquellos pixeles que son estrictamente interiores a la región, incluso cuando un pixel exterior este mas cerca de la arista. De tal manera, un polígono no se introduce (incluso por un solo pixel) en las regiones definidas por otras primitivas. Por lo tanto se debe ajustar el algoritmo de conversión de línea de forma correspondiente.

Algoritmo básico de relleno de áreas

Como con el algoritmo original de punto medio, se usa un algoritmo incremental para calcular los extremos de un intervalo en una línea de rastreo de aquellos en la línea de rastreo previa sin tener que computar de forma analítica las intersecciones de una línea de rastreo con cada arista del polígono.
Los intervalos se pueden llenar por medio de un proceso consistiendo de tres pasos:

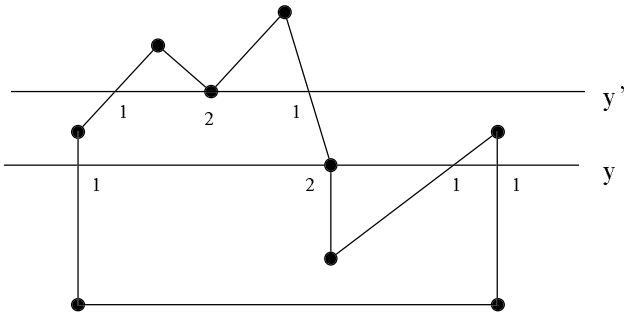
1. Encontrar las intersecciones de la línea de rastreo con todas las aristas del polígono.
2. Ordenar las intersecciones de forma incremental para coordenadas de x.
3. Llenar los pixeles entre pares de intersecciones que están en el interior del polígono, usando una regla de no-paridad para determinar que un punto esta dentro de una región: La paridad es originalmente par, y cada vez que se encuentra una intersección, se invierte la paridad. Se dibuja solo cuando la paridad es non, no se dibuja si es par.

Los pasos 1 y 2, encontrar intersecciones y ordenarlas, se verán en la siguiente sección.
El paso 3, llenado de intervalos, se expande a continuación.
Para la lista de intervalos ordenado del ejemplo (10,14.5,18.5,24), el paso 3 requiere cuatro elaboraciones:

3.1 Dada una intersección con un valor arbitrario fraccional de *x*, como determinamos cual de los dos pixeles a los lados de la intersección es interior ?
3.2 Como manejamos el caso especial de intersecciones en coordenadas de pixel enteras ?
3.3 Como manejamos el caso especial en 3.2 para vértices compartidos ?
3.4 Como manejamos el caso especial en 3.2 donde los vértices definen una línea horizontal (máximo o mínimo) ?

Manejo de vértices

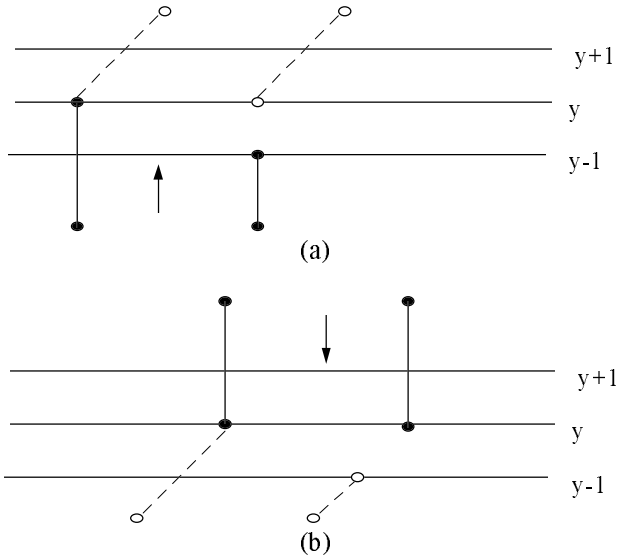
Una línea de rastreo que pasa a través de un vértice intersecta dos aristas del polígono en esa posición, agregando dos puntos a la lista de intersecciones para la línea de rastreo. En la siguiente figura se presentan dos líneas de rastreo en las posiciones *y'* y *y* que intersectan los extremos de la arista.



La línea de rastreo *y* intersecta cinco aristas del polígono. No obstante, la línea de rastreo *y'* intersecta un numero par de aristas, aunque también pasa a través de un vértice. Los puntos de intersección a lo largo de la línea de rastreo *y'* identifican en forma correcta los intervalos interiores. Pero con la línea de rastreo *y*, se necesita realizar cierto procesamiento adicional para determinar los puntos interiores correctos.

La diferencia topología entre la línea de rastreo *y* y la línea de rastreo *y'* se identifica al notar la posición de los bordes de intersección en relación con la línea de rastreo. Para la línea de rastreo *y*, los bordes de intersección que comparten un vértice están en lados opuestos de la línea de rastreo. Para la línea de rastreo *y'*, las dos aristas de intersección se encuentran arriba de la línea de rastreo. Así, los vértices que requieren procesamiento adicional son aquellos que tiene aristas que se unen en lados opuestos de la línea de rastreo. Se puede identificar estos vértices al rastrear alrededor de la frontera del polígono ya sea en el sentido del reloj o en dirección inversa y observar los cambios relativos en las coordenadas de *y* del vértice conforme se desplaza de una arista a la siguiente. Si los valores de *y* del extremo de dos aristas consecutivas aumentan o decrecen en forma monótona, se debe contar el vértice medio como un punto de intersección particular para cualquier línea de rastreo que pasa a través de ese vértice. De otro modo, el vértice común representa un extremo local (mínimo o máximo) en la frontera del polígono y las dos intersecciones de la arista con la línea de rastreo que pasa a través de ese vértice puede agregarse a la lista de intersección.

Una manera de responder la pregunta de si se debe considerar un vértice como una o dos intersecciones, es al reducir algunas aristas del polígono para dividir los vértices que deben considerarse como una intersección.
Se puede procesar aristas no horizontales alrededor de la frontera del polígono en el orden especifico, sea en el sentido del reloj o en dirección contraria. Conforme se procesa cada arista, se puede realizar una verificación para determinar si esa arista y la siguiente arista no horizontal tiene valores de *y* del extremo creciente o decrecientes de forma monótona. Si tal es el caso, la arista inferior se puede reducir para asegurarse de que se genere solo un punto de intersección para la línea de rastreo que pasa a través del vértice común y una de las dos aristas.
En la siguiente figura se muestra la reducción de una arista:



Cuando las coordenadas de *y* del extremo de las dos aristas se incrementan, el valor de *y* del extremo superior para la arista actual se reduce a razón de 1 (ver (a)).
Cuando los valores de *y* se reducen de manera monótona (ver (b)), se reduce la coordenada de *y* del extremo superior de la arista que sigue a la actual.

Refinación del algoritmo

El manejo de los casos se resume a continuación:

3.1 Si nos aproximamos a una intersección fraccional hacia la derecha, y estamos dentro del polígono, se redondea hacia abajo (trunca) la coordenada de *x* de la intersección para definir el pixel interior; si estamos fuera del polígono, se redondea hacia arriba para estar adentro.

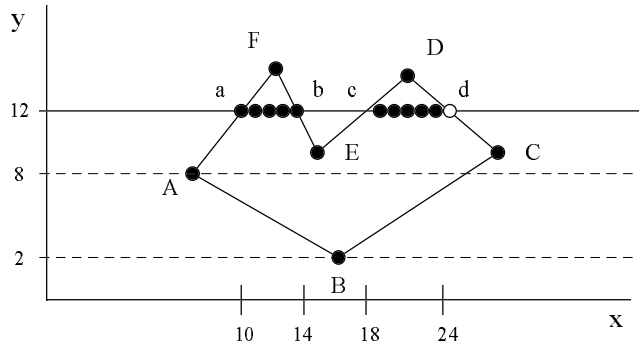
3.2 Se aplicando el siguiente criterio para evitar conflictos entre aristas compartidas de rectángulos: Si el pixel de mas a la izquierda en un intervalo tiene una coordenada x entera, se define como interior; si el pixel de mas a la derecha tiene una coordenada x entera, se define como exterior.

3.3 Se cuenta el vértice y_{min} de un arista en el calculo de paridad pero no el vértice y_{max}; por lo tanto, un vértice y_{max} se dibuja solo si equivale al vértice y_{min} del arista adyacente. Por ejemplo, el vértice A, en la figura anterior, se cuenta una vez en el calculo de paridad ya que es el vértice y_{min} para la arista FA, pero el vértice y_{max} para la arista AB.

3.4 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, como se vera mas adelante, si no se cuentan los vértices de las aristas, ya que no son ni y_{min} ni y_{max}.

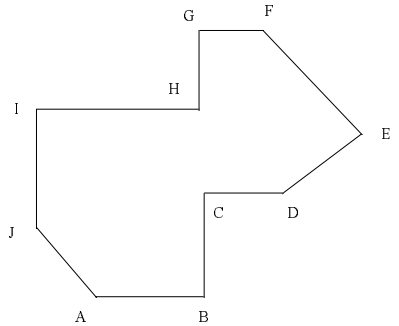
Si se aplica estas reglas al ejemplo anterior, el cual no intersecta ningún vértice, se llenaría de la siguiente forma:

- Para la línea de rastreo 12, se llenan los pixeles desde el punto a (10,12), pixel 10, hasta el primer pixel a la izquierda del punto b (14.5,12), pixel 14, y del primer pixel a la derecha del punto c (18.5,12), pixel 19, hasta 1 pixel a la izquierda del punto d (24,12), pixel 23.
- Para la línea de rastreo 8, el vértice A cuenta una vez ya que es el y_{min} del arista FA pero el y_{max} del arista de AB; esto causa paridad non, por lo cual dibujamos el intervalo de allí a un pixel a la izquierda de la intersección con CB, donde la paridad se vuelve par y el intervalo se termina.
- Para la línea de rastreo 2, se intersecta el vértice B; los aristas AB y BC ambos tienen sus vértices y_{min} en B, por lo cual se cuenta doble y se deja la paridad par. Este vértice actúa como intervalo nulo: se entra al vértice, se dibuja el pixel, y se sale del vértice. Aunque tal mínimo local dibuja un solo pixel, no se dibuja un pixel en un máximo local, tal como la intersección con el vértice F, compartida por las aristas FA y EF. Ambos vértices son y_{max} y por lo tanto no afectan la paridad, que se queda par.



2.1 Aristas Horizontales

Se maneja apropiadamente las aristas horizontales al no contar sus vértices.
Consideremos la siguiente figura:

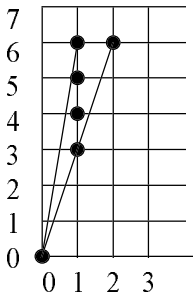


En el arista inferior AB, el vértice A es un vértice y_{min} para el arista JA, y AB no contribuye. Por lo tanto, la paridad es non y se dibuja el intervalo AB. El arista vertical BC tiene su y_{min} en B, pero otra vez, AB no contribuye. La paridad se hace par, y el intervalo se termina. En el vértice J, el arista IJ tiene un vértice y_{min} pero el arista JA no lo tiene, por lo cual la paridad se hace non y el intervalo se dibuja hasta el arista BC. El intervalo que comienza en el arista IJ e intersecta C no ve ningún cambio en C ya que C es un vértice y_{max} para BC, por lo cual el intervalo continua a través del arista inferior CD; sin embargo, en D, el arista DE tiene un vértice y_{min}, por lo cual la paridad se regresa a par y el intervalo termina. En I, el arista IJ tiene un vértice y_{max} y el arista HI no contribuye tampoco, por lo cual la paridad se mantiene par y el arista superior IH no se dibuja. Sin embargo, en H, el arista GH tiene un vértice y_{min}, la paridad se vuelve non, y el intervalo se dibuja de H hasta el pixel a la izquierda de la intersección con el arista EF. Finalmente, no hay un vértice y_{min} en G, ni en F, por lo cual el arista superior FG no se dibuja.

Este algoritmo trata con vértices compartidos en un polígono, con aristas compartidos por dos polígonos adyacente, y con aristas horizontales. Permite polígonos que se intersectan a si mismos. No obstante, no trabaja perfectamente, ya que omite pixeles. Peor aun, no puede evitar completamente escribir pixeles compartidos múltiples veces sin mantener una historia: e.g. aristas compartidos por mas de dos polígonos o un vértice y_{min} compartido por dos triángulos disjuntos.

2.2 Slivers

Existe otro problema con el algoritmo de conversión de línea, uno que no se resuelve de forma tan satisfactoria como el caso de las aristas horizontales: Polígonos con aristas que están bastante cerca crean en forma conjunta un **sliver -** un área poligonal tan fina que su interior no contiene un intervalo distintivo para cada línea de rastreo. Por ejemplo, un triángulo de (0,0) a (1,6) a (2,6) a (0,0), como se muestra en la siguiente figura:



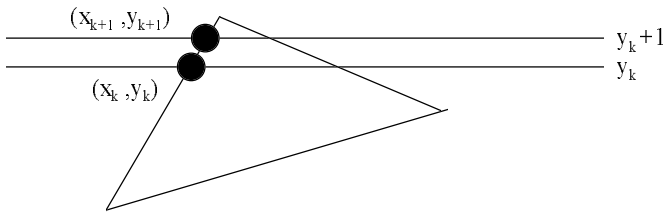
Dada la regla que se dibujan solo pixeles que están en el interior o en una arista izquierda o inferior, habrá muchas líneas de rastreo con un solo pixel o sin ningún pixel. El problema de tener pixeles *faltantes* es aun otro ejemplo del problema de *aliasing*, o sea, de representar una señal continua con aproximaciones discretas. Si tuviéramos múltiples bits por pixel, pudiéramos utilizar técnicas de antialiasing como se vera mas adelante. Antialiasing consistiría de suavizar la regla de *dibujar solo en pixeles que están en el interior o en una arista izquierda o inferior*, para permitir que pixeles de borde e incluso del exterior tengan valores de intensidad que varíen según su función de distancia del pixel central y la primitiva; múltiples primitivas pueden contribuir al valor de un pixel.

2.3 Coherencia de Arista y el Algoritmo de Conversión de Línea

El paso 1, según se señalo anteriormente, es determinar las intersecciones de arista. Se debe evitar métodos lentos, en especial de fuerza bruta de verificar cada arista de un polígono para su intersección con cada nueva línea de rastreo. Los cálculos que se realizan en la conversión de rastreo y en otros algoritmos de gráfica aprovechan, por lo regular, las diversas propiedades de **coherencia** de una escena que se debe desplegar. Se expresa mediante el termino coherencia que las propiedades de una parte de una escena se relacionan de algún modo con otras partes de la escena en forma tal que la relación puede servir para reducir el procesamiento. Los métodos de coherencia a menudo implican cálculos incrementales aplicados a lo largo de una línea de rastreo particular o entre líneas de rastreo sucesivas.

Algoritmo básico

El calculo de intersecciones se puede establecer por medio de cálculos de coordenadas en incrementos a lo largo de cualquier arista y aprovechar el hecho que la pendiente de la arista es constante de una línea de rastreo a la siguiente (**coherencia de arista**). La siguiente figura muestra dos líneas de rastreo sucesivas que cruzan una arista izquierda de un polígono.



La pendiente de esta arista del polígono puede expresarse en términos de las coordenadas de intersección de la línea de rastreo:

(58) $m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$

Puesto que el cambio en las coordenadas de y entre las dos líneas de rastreo es solo

(59) $y_{k+1} - y_k = 1$

el valor de la intersección de x_{k+1} en la línea de rastreo superior puede determinarse a partir del valor x_k de la intersección de x en la línea de rastreo anterior como

(60) $x_{k+1} = x_k + 1/m$

Así, cada intersección de x sucesiva puede calcularse al sumar la inversa de la pendiente y redondear al entero mas próximo.

Consideremos líneas con una pendiente $m>1$ para aristas izquierdas (aristas derechas y otras pendientes se manejan de forma similar).
En el punto extremo (x_{min}, y_{min}) se requiere dibujar un pixel.
Según se incrementa y, la coordenada de x del punto en la línea ideal se incrementara con valores de $1/m$ a lo largo de una arista, donde la pendiente se da con

$$m = (y_{max} - y_{min})/(x_{max} - x_{min})$$

siendo la pendiente m es equivalente a la división de dos enteros:

$$m = \Delta y/\Delta x$$

cuando Δy y Δx son las diferencias entre las coordenadas x y y del extremo de la arista. Por lo tanto, los cálculos incrementales de las intersecciones de x a lo largo de una arista para líneas de rastreo sucesivas pueden expresarse como

(62) $x_{k+1} = x_k + \Delta x / \Delta y$

Este incremento resultara en que x tenga una parte entera y otra fraccional, la cual puede expresarse como una fracción con denominador $\Delta y = y_{max} - y_{min}$. Según se itera este proceso, la parte fraccional sobrepasara el numero entero, y la parte entera tendrá que incrementarse.

Por ejemplo, si la pendiente es 7/3 ($\Delta x/\Delta y$ = 3/7), y x_{min} es 0, entonces la secuencia de valores de x será la siguiente:

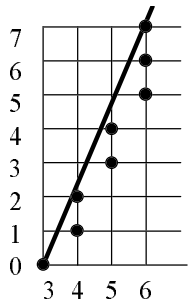
x
0
3/7
6/7
9/7 = 1 2/7
1 5/7
1 8/7 = 2 1/7
2 4/7
2 7/7 = 3

Se consideran los siguientes casos:

- Cuando la parte fraccional de x es cero: se dibuja el pixel (x,y) que esta sobre la línea
- Cuando la parte fraccional de x es mayor que cero pero menor que uno: se redondea hacia arriba (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 por uno y se resta uno a la parte fraccional. Se aplica a la nueva especificacion de x la consideracion de fracciones menores a uno pero mayores a cero.

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

Esto se puede ver en la siguiente figura:



Algoritmo de linea sin fracciones

Se puede evitar el uso de fracciones siguiendo solo el numerador de la fracción y observando que la parte fraccional es mayor a 1 cuando el numerador es mayor que el denominador. Al utilizar esta ecuación, se puede llevar a cabo una evaluación entera de las intersecciones de x, empezando a contar desde 0 y aumentando la cuenta por el valor de Δx cada vez que se mueve a una línea de rastreo. Se consideran nuevamente los siguientes casos:

- Si el valor de la cuenta es igual a 0, y se mantiene el valor actual de x.
- Si el valor de la cuenta es mayor que 0 pero menor que Δy, se incrementa la intersección actual de x a razón de 1.

- Si el valor de la cuenta es mayor o igual que Δy, se reduce la cuenta a razón del valor Δy y se incrementa la intersección actual de x a razón de 1, y se aplican las consideraciones anteriores.

Como ejemplo de calculo incremental con enteros, tomemos el arista anterior con una pendiente $m = 7/3$.
En la línea de rastreo inicial, se establece la cuenta como 0 y su incremento como 3.
Conforme se mueve a las tres líneas de rastreo siguientes a lo largo de esta arista, se asignan a la cuenta, en forma sucesiva, los valores 3, 6, y 9.
Los primeros dos incrementos son menores al incrmento de y, por lo cual se incrementa x por uno.
En la tercera línea de rastreo sobre la línea de rastreo inicial, la cuenta tiene ahora un valor mayor que 7.
Así incrementamos la coordenada de la intersección de x a razón de 1 y volvemos a establecer la cuenta como el valor 9 - 7 = 2.
La siguiente tabla muestra los valores encontrados en el procedimiento:

x	$n=n+\Delta x$	y_k	x_k
0	0	0	0
0	3	1	1
0	6	2	1
1	9-7=2	3	2
1	5	4	2
2	8-7=1	5	3
2	4	6	3
2	7-7=0	7	3

Nótese que el cálculo anterior es para arista izquierda con pendiente positiva. Se desarrolla consideraciones similares para las demás aristas y de acuerdo a las diferentes pendientes (positivas o negativas).

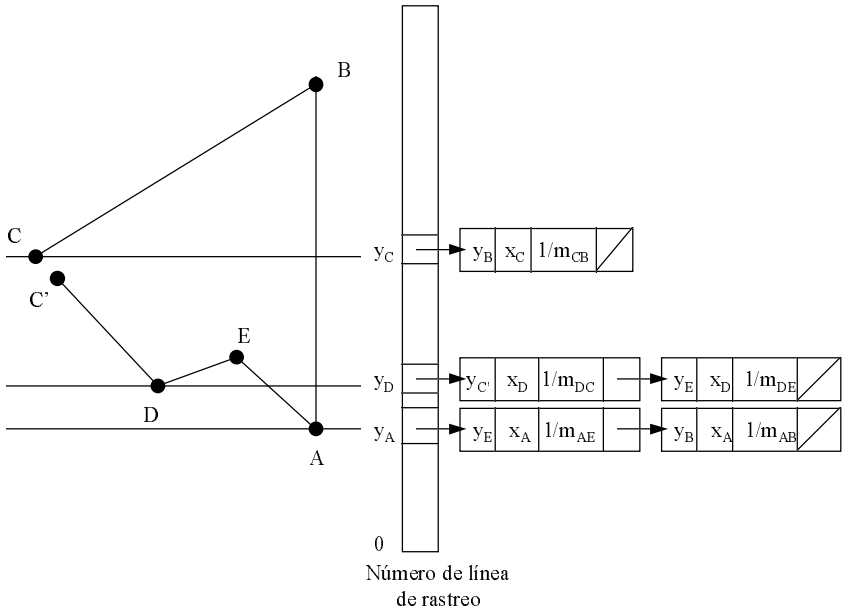
Algoritmo final de relleno

Ahora desarrollamos un **algoritmo de conversión de línea de rastreo** que aprovecha la coherencia de esta línea y, para cada línea de rastreo.

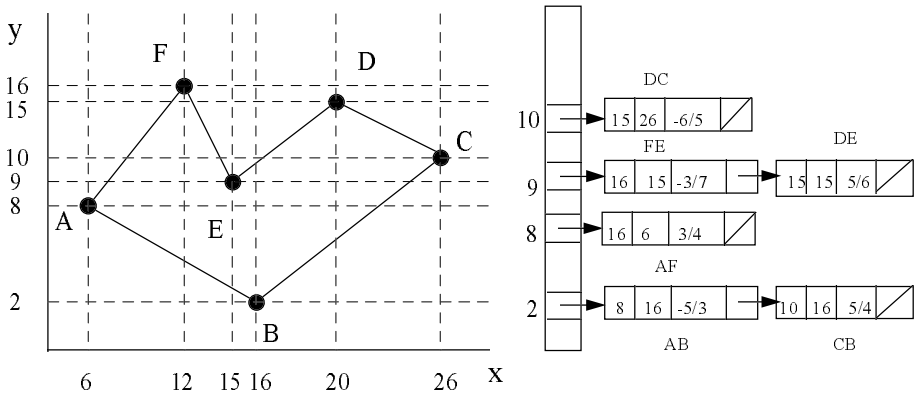
Se crea inicialmente una **tabla de aristas** (**ET - edge table**) (también conocida como **tabla de arista clasificada** (**ordered edge table**) conteniendo todas las aristas. El ET tipicamente se construye usando un "buckle sort" con la misma cantidad de "buckets" que líneas de rastreo en la pantalla. La organizacion del ET es de acuerdo a las siguientes consideraciones:

- Ordenada según su coordenada de y_{min}.
- Dentro de cada bucket, las aristas se guardan en orden de coordenada x creciente del extremo inferior.
- Cada entrada en el ET contiene la coordenada máxima y_{max} del arista, la coordenada x del vértice inferior (x_{min}) y el incremento (la pendiente inversa) de x usado para ir de una línea de rastreo a otra, $1/m$.
- En la tabla de arista clasificada solo se incluyen las aristas no horizontales.

En la siguiente figura se presenta un polígono y la tabla ET asociada:

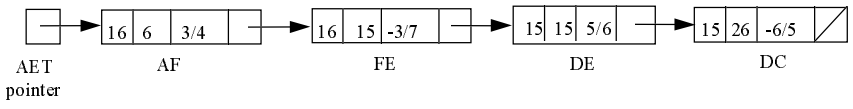


Por ejemplo, para nuestro polígono anterior, la tabla ET se muestra en la siguiente figura:

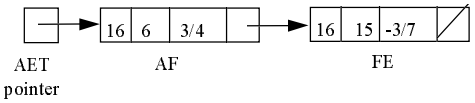


Además del ET, se mantiene un conjunto de aristas que intersectan la linea de rastreo actual en una estructura de datos llamada la **tabla de aristas activa** (**AET - active edge table**).
Los aristas del AET se ordenan según sus valores de intersección de x para así poder llenar los intervalos en pares de valores de intersección (redondeados).
Según nos movemos a la línea de rastreo siguiente en $y + 1$, el AET se actualiza.

La siguiente figura muestra el AET para las líneas de rastreo 15 (a) y 16 (b):



(a)



(b)

(En una implementación real se debe agregar una bandera indicando si es una arista izquierda o derecha.)

Una vez formado el ET, se completan los siguientes pasos de procesamiento para el algoritmo de conversión de línea:

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 el bucket y a AET aquellos aristas cuyos $y_{min} = y$ (aristas entrantes), luego clasificar el AET en x (se facilita porque ET esta preclasificado).
   - 3.2 Llenar los valores de pixel deseados en la línea de rastreo y usando pares de coordenadas x del AET.
   - 3.3 Quitar del AET aquellas entradas cuyos $y_{max} = y$ (aristas no involucrados en la línea de rastreo siguiente).
   - 3.4 Incrementar y por 1 (a la coordenada de la siguiente línea de rastreo).
   - 3.5 Para cada arista no-vertical aun en el AET, actualizar x para el nuevo y.

Se puede computar todos los intervalos en una paso, luego llenar los intervalos en un segundo paso, o computar un intervalo y llenarlo cuando se complete.
La implementación de los cálculos de intersección de arista se puede facilitar también al almacenar los valores Δx y Δy en la tabla de arista clasificada.

Los triángulos y trapezoides se pueden tratar como casos especiales de polígonos, ya que solo tienen dos aristas para cada línea de rastreo (ya que aristas horizontales se convierten de forma explícita).
Y como un polígono puede convertirse en un conjunto de triángulos compartiendo vértices y aristas, se puede convertir un polígono general primero descomponiéndolo en triángulos y luego convirtiendo los triángulos.