Capítulo II.- Algoritmos Gráficos en 2 Dimensiones.

Un paquete gráfico raster aproxima las primitivas gráficas matemáticas (ideales), descritas en términos de un plano cartesiano, a un conjunto de pixeles de una determinada intensidad de gris o color. Estos pixeles son almacenados como mapas de bits en un buffer de memoria gráfica.

II.1.- Discretización de líneas

Un algoritmo de discretización de líneas calcula las coordenadas de los pixeles que están sobre o cerca de una línea ideal infinitamente delgada sobrepuesta a una trama bidimensional.

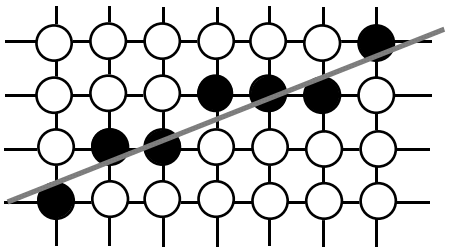


Figura II.1. Relación de píxeles con línea "ideal"

En principio nos gustaría que la secuencia de píxeles estuviera lo más cerca posible de la línea ideal y que además fuera lo más recta posible.

Consideremos por el momento líneas de 1 píxel de grosor que tengan exactamente un pixel de dos niveles en cada columna (en cada fila patra líneas de pendiente mayor o igual que ±1).

Para visualizar la geometría supondremos que mostraremos un pixel como un punto circular centrado en la posición (x, y) de la trama.

II.1.1.- Algoritmo incremental básico

La estrategia más sencilla para discretizar líneas es calcular la pendiente m como $\Delta y/\Delta x$ e incrementar x en 1 a partir del punto más a la izquierda. Así se calculará :

$$y_i = m * x_i + B; \ \forall x_i$$

Así se obtendrá pares de puntos $(x_i, round(y_i))$ donde $round\ (y_i) = floor\ (0{,}5 + y_i)$.

En la práctica esta estrategia podría ilustrarse de la siguiente forma:

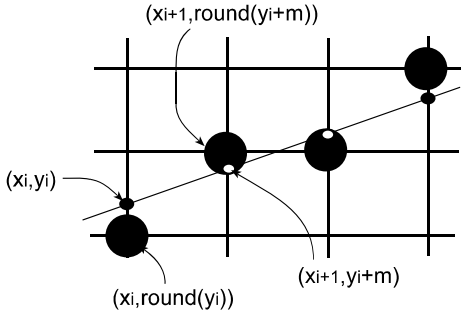


Figura II.2. Aplicación de algoritmo incremental básico

Esta estrategia, aunque es sencilla no es muy eficiente ya que requiere realizar una multiplicación y una suma en punto flotante y además invocar a la función *floor* por cada iteración (punto dibujado).

Observemos que

$$y_{i+1} = m * x_{i+1} + B = m * (x_i + \Delta x) + B = y_i + m * \Delta x$$

y como Δx es 1, entonces

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

Resumiendo, si $x_{i+1} = x_i + 1$, entonces $y_{i+1} = y_i + m$.

Si $|m| > 1$, un incremento en x crea un incremento en y mayor que 1. Por lo tanto, se debe invertir los papeles de x e y, es decir,

aumentar x en $\Delta x = \Delta y / m = 1/m$ y asignarle un incremento unitario a y.

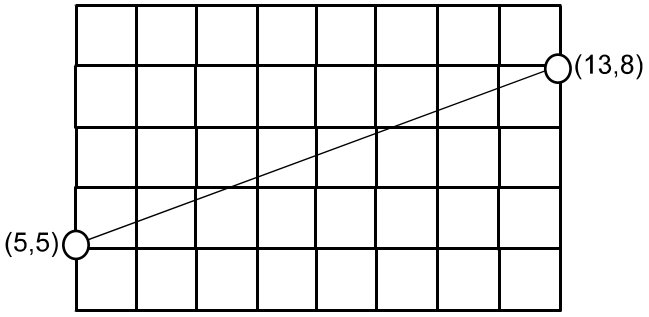
Otros casos especiales corresponden a la discretización de líneas con pendiente ∞ y 0, los cuales deben tratarse en forma particular, notar que el primer caso se da cuando Δx es 0 y el segundo cuando Δy es 0.

El siguiente procedimiento muestra sólo el caso de discretización de líneas con pendiente $|m| \leq 1$.

```
procedure linea (x0, y0, x1, y1, valor : integer);
var x : integer;
    dx, dy, y, m : real;
begin
    dy := y1 - y0;
    dx := x1 - x0;
    m := dy / dx;
    y := y0;
    for x := x0 to x1 do begin
          escribirPixel(x, int(floor(y+0,5)),valor);
          y := y + m;
          end;
end;
```

Ejercicios 1 :

1) Aplique el algoritmo incremental para obtener los valores de píxeles en el siguientes caso:



2) Implemente en forma completa el algoritmo incremental para discretización de líneas.

II.1.2.- Algoritmo de punto medio

El algoritmo anterior utiliza aritmética de punto flotante, lo que hace lo hace tener cierta desventaja. Bresenhan desarrollo un algoritmo que sólo emplea aritmética entera, es decir, que calcula (x_{i+1}, y_{i+1}) a partir de (x_i, y_i).

Para ilustrarlo supondremos que la pendiente de la línea está entre 0 y 1 (las líneas de otras pendientes se pueden obtener por reflexión). Al punto extremo inferior le denominaremos (x_o,y_o) y al superior derecho (x_1,y_1).

Consideremos la situación ilustrada en la siguiente figura, en la cual, según la formulación de Bresenhan, se calcula la diferencia entre las distancias *E* y *NE* a *Q* y se usa el signo de la diferencia para seleccionar la mejor aproximación de la línea al píxel cuya distancia desde *Q* sea la menor.

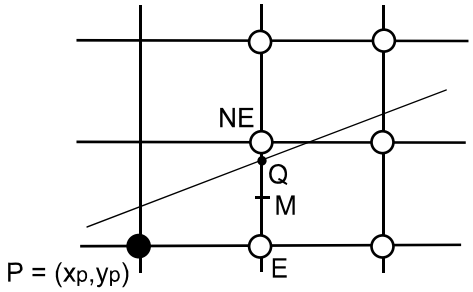


Figura II.3. Aplicación de algoritmo de punto medio para dicretización de línea

En la formulación del algoritmo de punto medio se observa a que lado de la línea se encuentra el punto medio *M*, es fácil ver que si *M* está por encima de la línea, el pixel *E* es el más cercano y si el punto medio *M* está por debajo de la línea el punto *NE* será el elegido. Así es como en el ejemplo el punto elegido será el *NE*.

¿Cómo se calcula a qué lado debe quedar el píxel?. Supongamos que representa la línea por

$$F(x, y) = a*x + b*y + c$$

y consideremos nuevamente

$$\Delta x = x_1 - x_0$$

$$\Delta y = y_1 - y_0$$

con esto la ecuación de la línea queda

$$y = \Delta y/\Delta x * x + B$$

ambas ecuaciones interceptan su pendiente, por lo tanto, podemos escribir

$$F(x, y) = \Delta y * x - \Delta x * y + B * \Delta x$$

donde $a = \Delta y$; $b = -\Delta x$; y, $c = B * \Delta x$.

No es necesario demostrar aquí que $F(x, y)$ es 0 en la línea, es mayor que 0 para los puntos por debajo de la línea y es negativa para los puntos sobre la línea. Para aplicar el criterio del punto medio sólo se debe calcular $F(M) = F(x_p+1, y_p+½)$ y calcular el signo.

Como nuestro criterio se basa en el valor de la función en el punto $(x_p+1, y_p+½)$, por definición

$$d = a*(x_p+1) + b*(y_p + ½) + c$$

que corresponde a la ecuación original evaluada en el siguiente punto medio. Si $d>0$ se elige el pixel NE; si $d<0$ se elige E y si $d=0$ se puede elegir cualquiera, aunque en este último caso debe quedar establecido cual se elegirá.

Los valores de M y d para el siguiente punto dependen de la elección de NE o E.

Si se elige E, M se incrementa una unidad en la dirección x, por lo tanto

$$d_{nuevo} = F(x_p+2, y_p+1/2) = a*(x_p+2) + b*(y_p + ½) + c$$

pero

$$d_{viejo} = a*(x_p+1) + b*(y_p + ½) + c$$

Al restar d_{viejo} a d_{nuevo} se obtiene la diferencia incremental

$$d_{\text{nuevo}} = d_{\text{viejo}} + a$$

El incremento que se usa después de elegir *E* se denomina ΔE y su valor es $\Delta E = a = \Delta y$.

En otras palabras, con sólo sumar ΔE a la variable de decisión actual podemos obtener, en forma incremental, el valor de la variable de decisión en el siguiente paso, y sin tener que calcular *F(M)*.

Si se elige *NE*, *M* se incrementa en una unidad, tanto en la dirección *x* como en la dirección *y*, luego

$$d_{\text{nuevo}} = F(x_p+2, y_p+3/2) = a*(x_p+2) + b*(y_p+3/2) + c$$

Aplicando el procedimiento de resta para obtener la diferencia incremental

$$d_{nuevo} = d_{\text{viejo}} + a + b$$

El incremento que sumamos a d después de elegir *NE* se llama ΔNE y corresponde a $\Delta NE = a + b = \Delta y - \Delta x$.

Resumiendo en el algoritmo del punto medio, en cada paso el método escoge entre dos pixeles basándose en el signo de una variable de decisión. Esta variable de decisión se actualiza sumándole ΔE o ΔNE.

Como el primer píxel esta en (x_o,y_o) se puede calcular directamente eligiendo *E* o *NE*. El primer punto medio está en

$$(x_o+1, y_o+½)$$

y

$$F(x_o+1, y_o+½) = a*(x_o+1) + b*(y_o+½) + c$$

$$= a*x_o + b*y_o + c + a + b/2$$

$$= F(x_o, y_o) + a + b/2$$

sin embargo, (x_o, y_o) es un punto de la línea, luego $F(x_o, y_o) = o$, entonces

$$d_{inicial} = a + b/2 = \Delta y - \Delta x/2.$$

Para eliminar la división por 2 se redefine F multiplicándola por 2, lo cual no afecta al signo de la variable de decisión.

Finalmente, el algoritmo de punto medio para la discretización de líneas cuyas pendientes están entre 0 y 1 es:

```
procedure linea (x0, y0, x1, y1, valor : integer);
var dx, dy, incr_E, incr_NE, d, x, y: integer;
begin
    dy := y1 - y0;
    dx := x1 - x0;
    d := 2*dy - dx;
    incr_E := dy * 2;
    incr_NE := (dy - dx) * 2;
    x := x0;
    y := y0;
    escribirPixel(x, y, valor);
    while (x < x1) do begin
          if (d <= 0) then begin
                d := d + incr_E;
                x := x + 1;
          end else begin
                d := d + incr_NE;
                x := x + 1;
                y := y + 1;
          end;
          escribirPixel(x, int(floor(y+0,5)),valor);
    end;
end;
```

Ejercicios 2

1) Aplicar el algoritmo de punto medio al primer problema del ejercicio 1.

2) Escribir el algoritmo de punto medio para el caso de general (todas las pendientes).

II.1.3.- Aspectos adicionales

- Una línea entre p_0 y p_1 debe marcar los mismos píxeles que una línea entre p_1 y p_0.

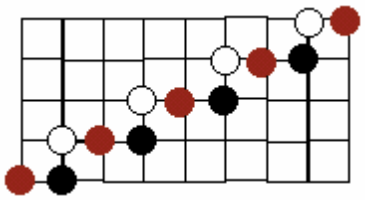


Figura II.4. Aplicación de pixeles en sentido p_0-p_1 y p_1-p_0

- Problemas de las líneas que se inician en las aristas de un triángulo de recorte.

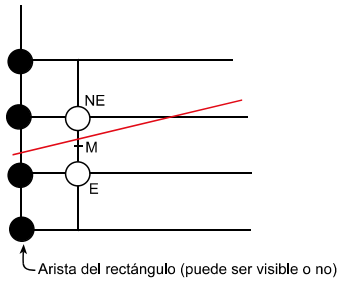


Figura II.5. Problemas de recorte de líneas

- Variación de la intensidad de la línea como función de la pendiente.

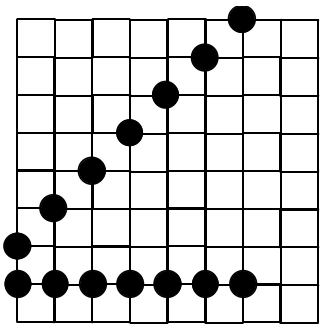


Figura II.6. Variación de intensidad como función de la pendiente

II.2.- Discretización de círculos

La ecuación de un círculo centrado en el origen es:

$$x^2 + y^2 = R^2$$

por lo cual, una primera aproximación para discretizar un círculo es resolver la ecuación

$$y = \pm(R^2 - x^2)^{1/2};\ \forall x \in Z,\ x \text{ entre } 0 \text{ y } R$$ [1]

También se pueden usar las ecuaciones ($R \cos \phi, R \operatorname{sen} \phi$) con ϕ creciendo desde $0°$ a $90°$, con esto se reduce el espaciado, pero tampoco es tan eficiente, ya que usa aritmética de punto flotante y llamadas a funciones.

II.2.1.- Aprovechamiento de la simetría

Se puede aprovechar la simetría del círculo para sólo calcular uno de los 8 segmentos de $45°$ y luego con éste determinar los demás octantes. La siguiente figura y procedimiento resumen el proceso:

[1] *Con esto se dibuja 1/4 de círculo, los demás cuadrantes se pueden obtener por simetría.*

```
procedure puntos_circulo (x, y:real; valor:integer);
begin
    escribir_pixel(x, y, valor);
    escribir_pixel(y, x, valor);
    escribir_pixel(y, -x, valor);
    escribir_pixel(x, -y, valor);
    escribir_pixel(-x, -y, valor);
    escribir_pixel(-y, -x, valor);
    escribir_pixel(-y, x, valor);
    escribir_pixel(-x, y, valor);
end;
```

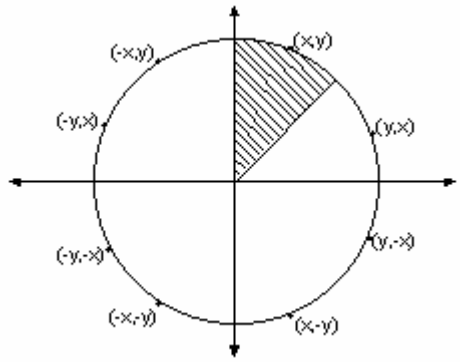


Figura II.7. Simetría en círculos

II.2.2.- Algoritmo de Círculo de Punto Medio

Se considerará la discretización del segmento de círculo de 45º correspondiente a $x = 0$ hasta $x = y = R/2^{1/2}$, los demás puntos se pueden dibujar con la función *puntos_circulo*.

Sean P, el pixel elegido previamente en (x_p, y_p); E y SE los píxeles candidatos (ver figura).

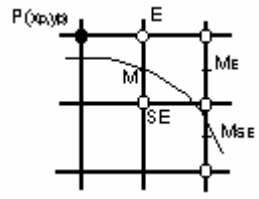


Figura II.8. Aplicación de algoritmo de punto medio para discretización de círculo

La ecuación de una circunferencia es

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

y $F(x, y) = 0$, si el punto (x, y) se encuentra en el círculo; $F(x, y) < 0$, si (x, y) está dentro del círculo; y, $F(x, y) > 0$, si (x, y) está fuera del círculo.

Se define la variable de decisión d en el punto medio entre E y SE.

$$d = F(x_p + 1, y_p - ½) = (x_p + 1)^2 + (y_p - ½)^2 - R^2$$

Si $d < 0$ se escoge E y la variable de decisión d_{nueva} será:

$$d_{nueva} = F(x_p + 2, y_p - ½) = (x_p + 2)^2 + (y_p - ½)^2 - R^2$$

lo cual da que la diferencia incremental es

$$d_{nuevo} = d + 2 * x_p + 3$$

Por lo tanto, se define el incremento por :

$$\Delta E = 2 * x_p + 3$$

Si $d > 0$ se elige SE y la variable de decisión d_{nuevo} será:

$$d_{nuevo} = F(x_p + 2, y_p - 3/2) = (x_p + 2)^2 + (y_p - 3/2)^2 - R^2$$

como

$$d_{nuevo} = d + 2 * x_p - 2 * y_p + 5$$

Por lo tanto, el incremento está dado por:

$$\Delta SE = 2 * x_p - 2 * y_p + 5$$

Es decir, en este caso los valores de ΔE y ΔSE son variables, dependen de los puntos seleccionados (x_p, y_p) en la iteración previa.

El cálculo de la condición inicial debe considerar que el punto de partida del círculo está en $(0, R)$, por lo tanto, el siguiente punto medio estará en $(1, R - ½)$ y

$$F(1, R - ½) = 1 + R^2 - R + ¼ - R^2 = 5/4 - R$$

Por lo tanto, el programa quedará

```
procedure circulo_punto_medio(radio, valor : integer);
var
    x, y : integer;
    d : real;
begin
    x := 0; y:= radio;
    d := 5.0 / 4 - radio;
    puntos_circulo (x, y, valor);
    while ( y > x) do begin
          if ( d < 0) then begin
                d := d + x * 2.0 + 3;
                x := x +1;
          end else begin
                d := d + (x - y) * 2.0 + 5;
                x := x + 1;
                y := y - 1;
          end;
          puntos_circulo (x, y, valor);
    end;
end;
```

Para que este algoritmo trabaje sólo con enteros se definirá una nueva variable de decisión

$$h = d - ¼$$

con lo que se reemplazará la línea 7 del programa a

$$h := 1 - radio;$$

En cuanto a la comparación $d < o$ se puede reemplazar sin problemas por $h < o$, ya que h se inicia y se incrementa en valores enteros.

II.3.- Atributos de las primitivas

Los atributos de las primitivas dependen de si esta es una línea, un círculo, un polígono, etc. Por ejemplo, una línea puede ser punteada, ancha, a color, etc.; un círculo puede estar relleno con un color o patrón, el texto puede tener una orientación, tamaño de caracteres, etc.

II.3.1.- Primitivas gruesas

Conceptualmente, las primitivas gruesas son generadas dibujando la *silueta* de una primitiva simple (un píxel).

Una estrategias es ubicar el centro de una *brocha* (u otro punto, como por ejemplo, la esquina superior izquierda de una brocha rectangular), la cual tiene una sección transversal especificada, en cada píxel calculado por el algoritmo de generación de la primitiva. Esta técnica deja un conjunto de preguntas por resolver: primero, ¿cuál es la forma *ideal* de la brocha?. Lo típico es usar brochas circulares o rectangulares. Segundo, ¿Cuál debe ser la *orientación* de una brocha no circular?. Tercero, ¿Cuál será el aspecto de los extremos de una línea gruesa?, etc.

Los métodos para dibujar primitivas gruesas son:

II.3.1.1 Replicación de pixeles

Una forma rápida de producir primitivas gruesas es escribir pixeles múltiples que incluyan al pixel calculado por el algoritmo de generación de la primitiva. Este método trabaja bastante bien para líneas; en el caso de líneas con pendiente entre (-1,1), los pixeles son duplicados en columnas y para las otras líneas se duplican en filas. Sin embargo, el efecto es que los puntos extremos de la línea son o verticales u horizontales.

Este algoritmo presenta algunos problemas:

- grandes espaciados en lugares donde los segmentos de línea producen ángulos.

- pérdida de pixeles en donde se produce cambios de replicación

- líneas horizontales y verticales con un grosor distinto a las líneas que están en ángulo

- Si el grosor de una línea es de tamaño par se debe seleccionar la columna o fila a replicar.

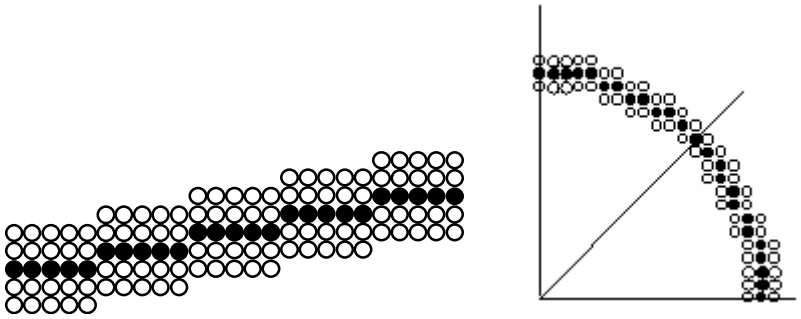


Figura II.9. Primitivas gruesas con replicación de píxeles

II.3.1.2 Lápiz móvil.

Seleccionar un lápiz móvil cuyo centro (o esquina superior izquierda) viaja a lo largo de una trayectoria formada por un pixel de la primitiva trabaja razonablemente para trazado de líneas. El problema de los puntos extremos de la línea persiste, al igual que el problema del grosor variable de la línea, el cual depende del ángulo usado.

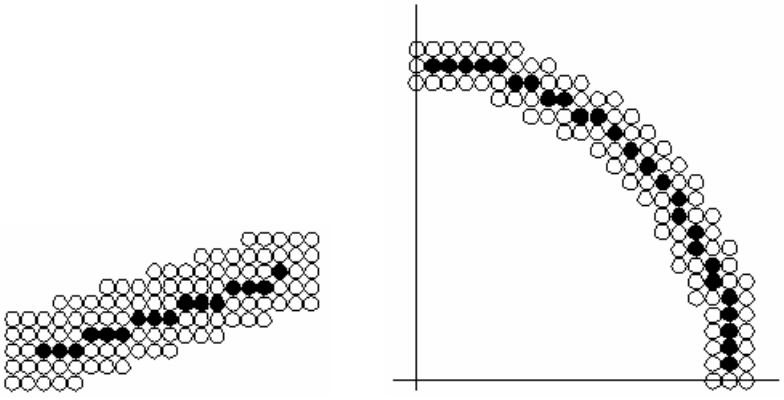


Figura II.10. Primitivas gruesas usando lápiz móvil

La solución en este caso pasa por definir una “huella” que corresponde a la sección transversal del lápiz móvil, el centro (o esquina superior derecha) de esta huella debe ubicarse en el pixel calculado para la primitiva simple, y los pixeles de la huella son copiados para producir la primitiva gruesa.

Este método produce un efecto adverso, algunos pixeles se “dibujan” más de una vez ya que la huella se sobrepone sobre varios pixeles en diferentes oportunidades. Para resolver este problema, el siguiente método define una estructura temporal que guarda los puntos antes de que sean dibujados

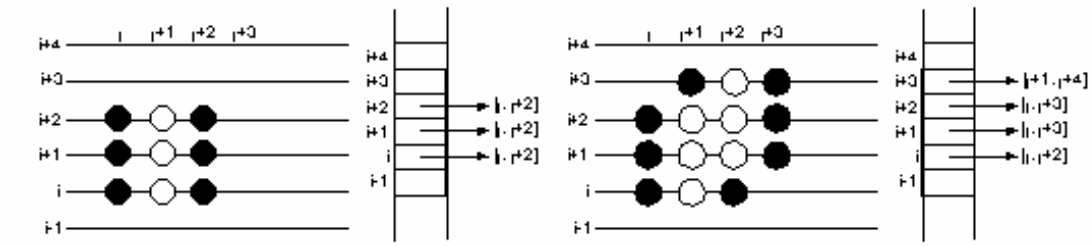


Figura II.10. Optimización de cálculo para primitivas gruesas usando lápiz móvil

II.3.1.3 Llenado de área entre bordes

El tercer método consiste en construir los márgenes internos y externos de la primitiva a una distancia de t/2 (t es el grosor de la primitiva) sobre cada lado de la trayectoria ideal de la primitiva.

Esta técnica permite manejar grosores pares e impares.

Con esta técnica una línea puede ser dibujada como un rectángulo de ancho t y largo equivalente a la longitud de la línea.

En el caso de los círculos, se debe generar dos círculos, uno externo de radio R+t/2 y uno interno de radio R-t/2. Luego se debe rellenar el espacio entre ellos.

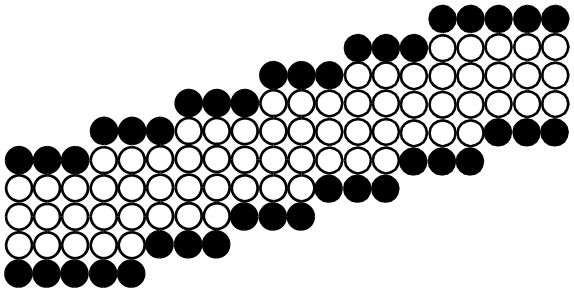


Figura II.11. Primitivas gruesas usando llenado entre áreas

II.3.2.- Rellenado de Primitivas

La tarea de rellenar primitivas se puede dividir en 2 partes: dividir que pixeles se deben usar para el relleno (dependiente de la forma de la primitiva y de los posibles recortes) y qué valores tendrán los pixeles que servirán para el relleno.

En primer lugar, se revisará el rellenado de primitivas no recortadas con colores sólidos.

En general, para determinar qué pixeles usar en el relleno se debe determinar la intersección entre las líneas de rastreo (horizontales) y las aristas de la primitiva.

En el caso de un rectángulo, se debe llenar cada espacio desde Xmin a Xmax con el mismo valor de pixel, en términos de algoritmo esto sería:

```
for (y from Ymin to Ymax)
      from(X from Xmin to Xmax)
            writePixel (x,y,valor)
```

Usando esta técnica se debe resolver el problema de los rectángulos que comparten arcos, habría que decidir en estos usos a que rectángulo pertenecen los arcos compartidos.

II.3.2.1 Rellenado de polígonos

El algoritmo de generación de polígonos que se describe a continuación maneja tanto polígonos convexos como cóncavos, incluso aquellos que se autointersectan y que tiene huecos interiores.

El algoritmo opera calculando los espacios existentes entre el arco izquierdo y derecho del polígono. Los tramos externos son calculados por el algoritmo incremental que calcula la intersección arco/línea de rastreo a partir de la intersección con la línea de rastreo previa.

La siguiente figura ilustra un polígono y una línea de rastreo que cruza en y = 8. Esto produce la intersección con los arcos AF, EF, DF y CD. Los arcos AF y CD caen sobre coordenadas enteras, en cambio, la intersección de la línea 8 con los arcos EF y CD no.

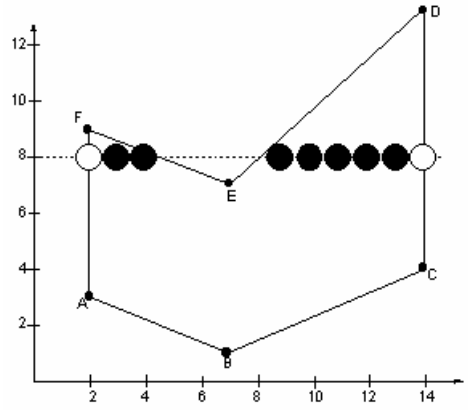
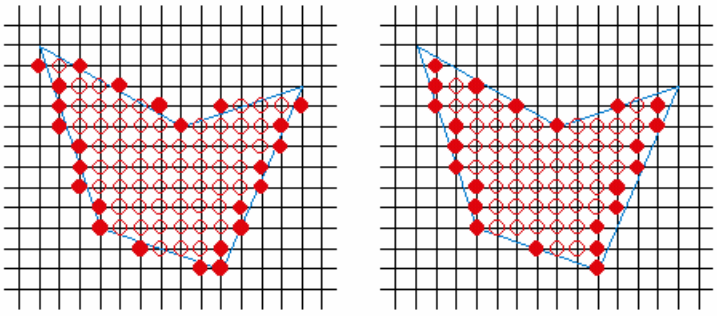


Figura II.12. Rellenado de polígonos

En la figura II.13, se observa un polígono dibujado aprovechando un algoritmo de discretización de líneas y luego rellenándolo. Sin embargo, algunos pixeles extremos quedan fuera de la primitiva por lo que, en la figura b, se ajustan al interior.



(a) *(b)*

Figura II.13. Dos polígonos rellenados con diferentes técnicas

Como en el algoritmo punto medio, se usa un algoritmo incremental para calcular los extremos de los arcos en una línea de rastreo a partir de las correspondientes a la línea anterior de rastreo, sin tener que calcular en forma analítica las intersecciones de la línea de rastreo con cada arco del polígono.

Por ejemplo, en la línea de rastreo 8 de la figura II.12 hay 2 arcos de pixeles dentro del polígono. Los arcos se pueden rellenar mediante un proceso de tres pasos:

1. Hallar las intersecciones de la línea de rastreo con todos los arcos del polígono.
2. Ordenar las intersecciones aumentado la coordenada x
3. Colocar todos los pixeles entre pares de intersecciones que se encuentren dentro de la región: la paridad es inicialmente par y cada intersección que se detecte invierte el bit de paridad; se dibuja cuando la paridad es impar.

Los dos primeros pasos de este proceso, la detección de intersecciones y ordenamiento se verán más adelante. La lista ordenada de coordenadas x es (2,4.5,8.5,13) para la figura II.12.

El tercer paso requiere:

3.1. Que dada una intersección con un valor de x arbitrario y fraccionario ¿Cómo se determina cuál de los dos pixeles a cada lado de la intersección es el interior?

3.2. ¿Cómo se trata el caso especial de las intersecciones en coordenadas enteras en los pixeles?

3.3. ¿Cómo se trata en caso especial del paso 3.2 para vértices compartidos?

3.4. ¿Cómo se trata en caso especial del paso 3.2 si los vértices definen una arista horizontal?

Para manejar el caso 3.1 determinamos sin aproximamos hacia la derecha a una intersección fraccionaria y estamos dentro del polígono, redondeamos hacia abajo la coordenada x de la intersección para definir el pixel interior; si estamos afuera del polígono, redondeamos, hacia arriba para quedar dentro.

Ejemplo: La intersección de la línea de rastreo 8 con x = 4.5 se resuelve redondeando x hacia abajo, e.d., x = 4, como pixel interior.

En el caso de x = 8.5 se resuelve redondeando x = 9 (hacia arriba), ya que, estabamos fuera del polígono.

El caso 3.2 se maneja con el mismo criterio que el caso de los rectángulos.

Para el caso 3.3 contamos el vértice y_{min} de una arista en el cálculo de la paridad pero no se incluye el vértice y_{max}, así sólo se dibujará el vértice y_{max} si es el vértice de un polígono adyacente.

Algoritmo de línea de rastreo

- No todos los puntos de una arista interesan para una línea de rastreo
- Se observa que, varias aristas intersectadas por una línea de rastreo i, también son intersectadas por la línea de rastreo i+1.
- Por tanto, el avanzar de una línea de rastreo a otra podemos calcular la nueva intersección x de la arista usar de la intersección de x anterior de acuerdo a
- $x_{i+1} = (x_i+1)/m$
- Donde m es la pendiente de la arista
- Este método se puede ajustar para manejar sólo aritmética entera, lo cual nos da el algoritmo siguiente:

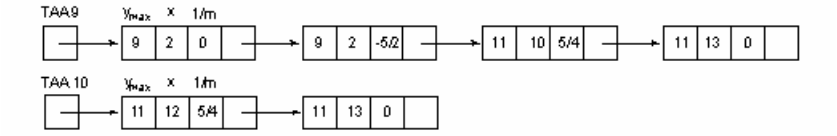
```
procedure rastreo-arista-izquierda(xmin,ymin,xmax,ymax,valor:integer);
var x, y, num, den, incr: integer;
begin
     x:= xmin;
     num := xmax – xmin;
     den := ymax – ymin;
     incr := den;
     for y := ymin to ymax - 1 do begin
           escribir_pixel(x, y, valor);
           incr: = incr + num:
           if(incr > den) then begin
                 x := x + 1;
                 incr := incr - den;
                 end;
           end;
end;
```

Este algoritmo sirve para arista izquierda y para pendientes mayores que 1, los otros casos se manejan con argumentos similares.

Algoritmo de línea de rastreo

TAA (Tabla de Arcos Activos) Dado que cada línea de rastreo lleva el control de puntos de intersección con las aristas.

Ejemplo : Para la figura II.12 la TAA de las líneas de rastreo 9 y 10 serían.

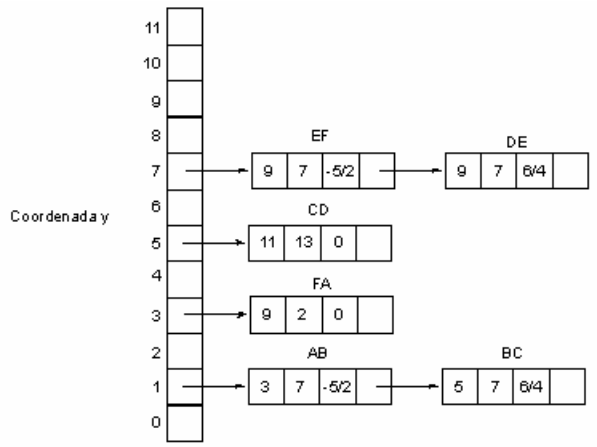


Las aristas en la TAA se ordenan de acuerdo con el valor de sus intersecciones con x, para que se puedan rellenar los tramos definidos por pares de valores de intersección (reordenados apropiadamente).

Al avanzar a la siguiente línea de rastreo $y+1$ se actualiza la TAA, primero eliminando las aristas en donde $y+1 > y_{max}$, luego agregando las aristas en que $y+1 = y_{min}$ y, finalmente calculando las nuevas instrucciones con x usando el algoritmo de rastreo de la arista correspondiente.

TA (tabla de arista o arcos) Contiene todas las aristas ordenadas de acuerdo a su y menor.

Ejemplo: La TA para la figura II.12 es:



Una vez formada la tabla de aristas se completan los siguientes pasos:

1. Asignar y igual a la menor coordenada y que tenga una entrada en la tabla TA.
2. Asignar la TAA para que esté inicialmente vacía (crear TAA).
3. Repetir hasta que TAA y TA estén vacías:
   - 3.1. Mover del espacio y de la TA a la TAA las aristas con $y_{min} = y$ y luego ordenar la TAA con base en x.
   - 3.2. Rellenar los valores de los pixeles deseados en la línea de rastreo y usando pares de coordenada x de la TAA
   - 3.3. Eliminar de la TAA las entradas donde $y = y_{max}$
   - 3.4. Incrementar y en 1 (próx. línea de rastreo).
   - 3.5 Actualizar x para la nueva y en cada arista no vertical que permanezca en la TAA.

Alternativas:

- Los triángulos y trapezoides pueden tratarse en forma especial.
- Los polígonos también se pueden tratar como mallas de triángulos (problema de triangularización).

II.4.- Recortes

II.4.1.- Recorte de puntos extremos

Si las fronteras de la coordenada x del rectángulo de recorte están en x_{min} y x_{max} y las fronteras de la coordenada y están en y_{min} y y_{max} debe satisfacer cuatro desigualdades.

$$X_{min} \leq x \leq x_{max}$$

$$Y_{min} \leq y \leq y_{max}$$

Si no se cumple alguna de las 4 desigualdades, el punto está fuera del rectángulo de recorte

II.4.2.- RECORTE DE LINEA

El caso es el siguiente

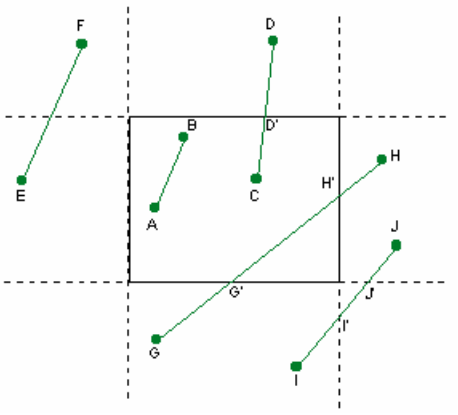


Figura II.14. Líneas con diferentes recortes

Caso AB se acepta trivialmente ya que los puntos extremos caen dentro del rectángulo.

Caso CD un punto está dentro y el otro fuera.

Casos EF, GH e IJ los puntos caen fuera.

II.4.2.1 Algoritmo de Recorte de Cohen - Sutherland

Este método permite detectar en forma sencilla si una línea se encuentra a la izquierda de x_{min}, arriba de y_{max}, bajo y_{min} o a la derecha de x_{max} donde (x_{min}, y_{min}) y (x_{max}, y_{max}) son las coordenadas que definen el rectángulo.

Si el segmento no pude aceptarse o rechazarse tan fácilmente se puede dividir para hacerlo.

El método divide el plano de presentación en 9 regiones.

A cada región se le asigna un código de 4 bits, determinado por la posición de la región con respecto a los medios planos exteriores de las aristas del rectángulo de recorte. Los 4 bits en el código, corresponden a las condiciones:

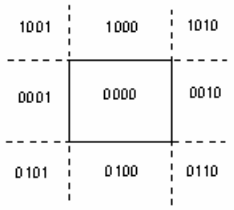


Figura II.15. Asignación de bits a las regiones

bit 1 fuera del medio plano sobre la arista superior $y > y_{max}$

bit 2 fuera del medio plano bajo la arista inferior $y < y_{min}$

bit 3 fuera del medio plano a la derecha de la arista derecha $x > x_{max}$

bit 4 fuera del medio plano a la izquierda de la arista izquierda $x < x_{min}$

Entonces, si los dos extremos de una línea tienen código cero, la línea está totalmente dentro del rectángulo.

Si los puntos extremos están fuera del rectángulo de recorte, entonces, tendrán códigos con 1's intercalados y al aplicar la operación lógica **and** debe ser cero, sino se rechaza trivialmente.

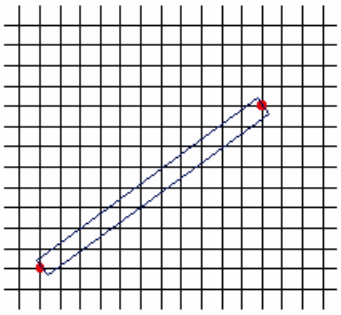
II.5.- Antialiasing

Las primitivas vistas hasta ahora tienen el problema de poseer aristas irregulares, este problema se denomina escalonamiento o serrado. La aplicación de técnicas para reducir este fenómeno se conoce como “antialiasing”.

Una técnica para aminorar el impacto del escalonamiento es aumentar la reducción, sin embargo, esto hace más costoso el hardware, ya que aumenta la memoria requerida por las imágenes y aumenta el tiempo de discretización.

II.5.1.- Muestreo de área no ponderada

Cualquiera que sea la primitiva dibujada, incluyendo las líneas, pueden ser consideradas ocupando una superficie en la pantalla. Por ejemplo, en la siguiente figura se ha supuesto que una línea puede “verse” como un rectángulo con una superficie sobre la pantalla.



Así, cada pixel “tocado” por este rectángulo debería manifestar su presencia en la línea a través de una intensidad que es proporcional a la sección del pixel “tocada” por la línea. Obviamente, en este caso suponemos que cada pixel es representado por más de 1 bit.

La técnica de establecer la intensidad en forma proporcional al área cubierta la llamamos muestreo de área no ponderada.

Esta técnica posee tres propiedades: la primera es la que la intensidad de un pixel intersectado por una línea decrece conforme

aumenta la distancia entre el centro del pixel y la orilla; la segunda propiedad es que una primitiva no puede influir en la intensidad de un pixel que no está intersectado y finalmente; la tercera propiedad es que las áreas iguales contribuyen con la misma intensidad, sin importar la distancia entre el centro del pixel y el área.

Con esta técnica se conservan las propiedades 1 y 2 de la técnica anterior, sin embargo, la tercera cambia, es decir, las áreas iguales contribuyen en forma desigual: un área pequeña cerca del centro del pixel tiene mayor influencia que una lejos del centro.