1 Línea

Los primitivos de salida son funciones que ofrecen las bibliotecas gráficas para describir estructuras geométricas básicas.
Cada primitivo de salida se especifica con los datos de las coordenadas de entrada y otra información referente a la manera en que se debe desplegar un objeto.
Los puntos y segmentos de línea recta son los componentes geométricos mas simples.

(**Scan Conversion** - Conversión de Rastreo)

Para pintar un punto en una ventana se utilizara la llamada a Xlib siguiente:

XDrawPoint(Display*, **Drawable**, **GC**, **int** *x*, **int** *y*)

Las líneas se dibujaran, de la forma mas básica, calculando los pixeles discreto que corresponden a la especificación de una línea especificada de forma continua.

1.1 Algoritmos para el Trazo de Líneas

La ecuación de intersección de la pendiente cartesiana de una línea recta es

(1) $y = m\,x + b$

donde m representa la pendiente de la línea y b la intersección de y.
Dado que los dos extremos de un segmento de línea se especifica en las posiciones (x_1,y_1) y (x_2,y_2), se puede determinar valores para la pendiente y la intersección de y en b con los siguientes cálculos:

(2) $m = (y_2 - y_1)/(x_2 - x_1)$

(3) $b = y_1 - m\,x_1$

Para cualquier x dentro del intervalo Δx a lo largo de una línea, se puede calcular el intervalo correspondiente a Δy de y a partir de la ecuación (2) como

(4) $\Delta y = m\,\Delta x$

De modo similar se puede obtener el intervalo Δx de x correspondiente a una Δy especifica como:

(5) $\Delta x = \Delta y\ /\ m$

Si $|\,m\,| < 1$, la línea es mas horizontal que vertical.
Según tiende a cero m, la línea tiende a ser horizontal.
Si $|\,m\,| > 1$, la línea es mas vertical que horizontal.
Según tiende a infinito m, la línea tiende a ser vertical.

El siguiente paso es efectuar un muestreo de la línea a puntos discretos correspondiente a los pixeles mas cercanos.

1.1.1 Algoritmo Básico

El algoritmo básico sería utilizar la ecuación anterior para calcular y en termino de x (o viceversa).
Esto sería bastante lento ya que requeriría siempre una multiplicación más una suma:

$y = m\,x + b$

Los siguientes algoritmos mostrarán como se pueden hacer más eficientes estos cálculos utilizando algoritmos más elaborados.

1.1.2 Algoritmo de Línea DDA

El *analizador diferenciador digital* (DDA - *Digital Differential Analyzer*) es un algoritmo de conversion de rastreo que se basa en el calculo ya sea de Δy o Δx por medio de las ecuaciones (4) o (5).
Se efectúa un muestreo de la línea en intervalos unitarios en una coordenada y se determina los valores enteros correspondientes mas próximos a la trayectoria de la línea para la otra coordenada.

Tomemos una línea con pendiente positiva, si la pendiente $|\,m\,| \leq 1$, se hace el muestreo en x en intervalos unitarios ($\Delta x = 1$ y $\Delta y = m$ dado que $m = \Delta y \,/\, \Delta x$) y se calcula cada valor sucesivo de y como:

(6) $y_{k+1} = y_k + m$

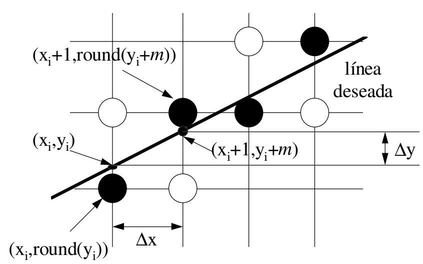
El subíndice toma valores enteros a partir de 1 y aumenta a razón de 1 hasta alcanzar el valor final.
Ya que m puede ser cualquier numero real entre 0 y 1, los valores calculados de y deben redondearse al entero mas cercano.

Para líneas con una pendiente $|\,m\,| > 1$, se revierten las funciones de x y y, o sea, se realiza un muestreo de y en intervalos unitarios ($\Delta y = 1$ y $\Delta x = 1/m$ dado que $m = \Delta y \,/\, \Delta x$) y se calcula cada valor sucesivo de x como:

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

Las ecuaciones (6) y (7) se basan en la suposición de que las líneas deben procesarse del extremo izquierdo al derecho.
Si este procesamiento se revierte, entonces Δx o Δy serian -1, y

$y_{k+1} = y_k - m$ o $x_{k+1} = x_k - 1/m$



El procedimiento completo de dibujo seria el siguiente:

```
void Line(Display* display, Window win, GC gc, int x0, int y0, int x1, int y1)
{
        float x, y, xs, ys;
        int dx, dy, steps;

        dx = x1 - x0;
        dy = y1 - y0;
/* se asigna el punto de donde se comenzara a dibujar la línea */
        x = x0;
        y = y0;
/* verificar si la pendiente es mayor de x o y, para luego asignarla a steps */
        if (abs(dx) > abs(dy))
                steps = abs(dx);
        else
                steps = abs(dy);
/* se divide por la pendiente mayor, para dar xs o ys igual a 1 (o -1) */
        if (steps == 0) {
                XDrawPoint(display,win,gc,round(x),round(y));
```

```
                fprintf(stderr,"this line is a point");
                return;
        }
        xs = dx/steps;
        ys = dy/steps;
/* se cicla uno a la vez hasta llegar al numero de steps máximo */
        for (i = 0; i <= steps; i++)
        {
            XDrawPoint(display,win,gc,round(x),round(y)); /* round(x) -> x+0.5 */
            x = x + xs;
            y = y + ys;
        }
}
```

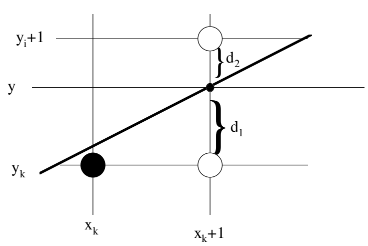
El problema con este algoritmo es que se debe redondear números flotantes a enteros y hacer operaciones sobre números flotantes , lo cual toma tiempo.
Para líneas largas, la acumulación de errores de redondeo en adiciones sucesivas del incremento de punto flotante pueden provocar que las posiciones del pixel calculadas se desvíen de la trayectoria real de la línea.
Se puede mejorar el desempeño del algoritmo al separar los incrementos m y 1/m en partes enteras y fraccionarias, de forma que todos los cálculos se reduzcan a operaciones de enteros.

1.1.3 Algoritmo de Línea Bresenham Básico

Un algoritmo preciso y efectivo para la generación de líneas de rastreo, desarrollado por Bresenham (1965), convierte mediante rastreo las líneas utilizando solo cálculos incrementales con enteros que se pueden adaptar para desplegar también curvas.
El algoritmo busca cual de dos pixeles es el que esta mas cerca según la trayectoria de la línea.

Consideremos el proceso de conversion para líneas con pendiente positiva $0 < m < 1$.
Las posiciones de pixel a lo largo de la trayectoria de una línea se determinan al efectuar un muestreo de x en intervalos unitarios.
Si se inicia desde el extremo izquierdo (x_0,y_0) de una línea determinada, se pasa a cada columna sucesiva y se traza el pixel cuyo valor de y se aproxima mas a la trayectoria de la línea de rastreo.
Si suponemos que se debe desplegar el pixel en (x_k,y_k), a continuación se necesita decidir que pixel se debe desplegar en la columna x_{k+1}.
Las alternativas son los pixeles (x_k+1,y_k), y (x_k+1,y_k+1).

Al realizar el muestreo en la posición x_k+1 designamos la separación de pixeles verticales de la trayectoria de la línea matemática como d_1 y d_2.



La coordenada de y en la línea matemática en la posición de la columna de pixel x_k+1 se calcula como

(10) $$y = m\,(x_k + 1) + b$$

Entonces

$$d_1 = y - y_k = m\,(x_k + 1) + b - y_k$$

y

$$d_2 = (y_k + 1) - y = y_k + 1 - m\,(x_k + 1) - b$$

La diferencia entre estas dos separaciones es

(11) $d_1 - d_2 = 2\ m\ (x_k + 1) - 2\ y_k + 2\ b - 1$

Un parámetro de decisión p_k para el paso k en el algoritmo de línea se puede obtener al reordenar la ecuación anterior, de modo que implique solo cálculos de enteros.
Esto se logra sustituyendo $m = \Delta y / \Delta x$ donde Δx y Δy son las separaciones horizontal y vertical de las posiciones de los extremos de la línea y al definir:

(12)
$$\begin{aligned} p_k = \Delta x\ (d_1 - d_2) &= \Delta x\ (2\ \Delta y / \Delta x\ (x_k + 1) - 2\ y_k + 2\ b - 1) \\ &= 2\ \Delta y\ x_k - 2\ \Delta x\ y_k + 2\ \Delta y + 2\ b\ \Delta x - \Delta x \\ &= 2\ \Delta y\ x_k - 2\ \Delta x\ y_k + c \end{aligned}$$

El signo de p_k es el mismo que el de $d_1 - d_2$ puesto que $\Delta x > 0$ en el ejemplo.
El parámetro c es un constante, donde $c = 2\ \Delta y + 2\ b\ \Delta x - \Delta x$, que es independiente del pixel.
Si el pixel y_k esta mas cerca de la trayectoria de la línea que el pixel $y_k + 1$ (es decir $d_1 < d_2$), entonces el parámetro de decisión p_k es negativo.
En ese caso, trazamos el pixel inferior; de otro mode, trazamos el pixel superior.

Los cambios de coordenadas a lo largo de la línea ocurren en pasos unitarios ya sea en la dirección de x o en la de y.
Por tanto, es posible obtener los valores de parámetros de decisión sucesivos al utilizar cálculos incrementales en enteros.
En el paso $k + 1$, el parámetro de decisión se evalúa con base en la ecuación anterior como

$$p_{k+1} = 2\ \Delta y\ x_{k+1} - 2\ \Delta x\ y_{k+1} + c$$

Al sustraer la ecuación (12) de la anterior obtenemos

$$p_{k+1} - p_k = 2\ \Delta y\ (x_{k+1} - x_k) - 2\ \Delta x(\ y_{k+1} - y_k)$$

Pero $x_{k+1} = x_k + 1$, de manera que

(13) $p_{k+1} = p_k + 2\ \Delta y - 2\ \Delta x(\ y_{k+1} - y_k)$

donde el termino $y_{k+1} - y_k$ es 0 o 1, dependiendo del signo del parámetro p.

Este calculo recurso de los parámetros de decisión se realiza en cada posición entera de x, empezando en el extremo izquierdo de las coordenadas de la línea.
El primer parámetro p_0 se evalúa a partir de la ecuación (12) en la posición del pixel inicial (x_0,y_0), sustituyendo con b = $y_0 - m\ x_0$ y $m = \Delta y / \Delta x$.

$$\begin{aligned} p_0 &= \Delta x\ (2\ \Delta y / \Delta x(x_0 + 1) - 2\ y_0 + 2\ (y_0 - (\Delta y / \Delta x)\ x_0) - 1) \\ &= 2\ \Delta y\ x_0 + 2\ \Delta y - 2\ \Delta x\ y_0 + 2\ \Delta x\ y_0 - 2\ \Delta y\ x_0 - \Delta x \end{aligned}$$

donde se obtiene la siguiente ecuación:

(14) $p_0 = 2\ \Delta y - \Delta x$

En resumen, los pasos son:

1. Se capturan los dos extremos de la línea y se almacena el extremo izquierdo en (x_0,y_0).
2. Se carga (x_0,y_0) en el bufer de estructura, o sea, se traza el primer punto.

3. Se calculan las constantes Δy, Δx, $2\Delta y$, $2\Delta y$-$2\Delta x$, y se obtiene el valor inicial para el parámetro de decisión como $p_0 = 2\ \Delta y - \Delta x$.
4. En cada x_k a lo largo de la línea, que inicia en k = 0, se efectúa la prueba siguiente: si $p_k < 0$, el siguiente punto que se debe trazar es (x_k+1,y_k) y $p_k +1 = p_k + 2\ \Delta y$. De otro modo, el siguiente punto en trazarse es (x_k+1,y_k+1) y $p_k +1 = p_k + 2\ \Delta y - 2\Delta x$.
5. Se repite el paso 4 otras Δx veces.

Ejemplo

Para ilustrar el algoritmo, utilicemos la línea con extremos (20,10) y (30,18).
Esta línea tiene una pendiente de 0.8, con

$$\Delta x = 10,\ \Delta y = 8$$

El parámetro de decisión inicial tiene el valor

$$p_0 = 2\ \Delta y - \Delta x = 6$$

y los incrementos para calcular parámetros de decisión sucesivos son

$$2\Delta y = 16,\ 2\Delta y - 2\Delta x = -4$$

Trazamos el punto inicial $(x_0,y_0) = (20,10)$ y determinamos las posiciones de pixel sucesivos a lo largo de la trayectoria de la línea a partir del parámetro de decisión como

k	p_k	(x_k+1,y_k+1)
0	6	(21,11)
1	2	(22,12)
2	-2	(23,12)
3	14	(24,13)
4	10	(25,14)
5	6	(26,15)
6	2	(27,16)
7	-2	(28,16)
8	14	(29,17)
9	10	(30,18)

Un trazo de pixeles se genera a lo largo de la trayectoria de esta línea.

En la siguiente rutina, se presenta una implementación del trazo de líneas de Bresenham para pendiente en el rango $0 < | m | < 1$, con trazo de izquierda a derecha en el caso de m positivo y de derecha a izquierda en el caso de m negativo.

```
void LineBres(Display* display, Window win, GC gc, int x0, int y0, int x1, int y1)
{
	int x, y, dx, dy, xend, p, incE, incNE;

	dx = abs(x1 - x0);
	dy = abs(y1 - y0);
	p = 2*dy - dx;
	incE = 2*dy;
	incNE = 2*(dy-dx);
/* determinar que punto usar para empezar, cual para terminar */
	if (x0 > x1) {
		x = x1;
		y = y1;
		xend = x0;
	}
	else {
		x = x0;
		y = y0;
		xend = x1;
	}
/* se cicla hasta llegar al extremo de la línea */
	while (x <= xend)
```

```
        {
            XDrawPoint(display,win,gc,x,y);
            x = x + 1;
            if (p < 0)
                p = p + incE
            else {
                y = y + 1;
                p = p + incNE;
            }
        }
}
```

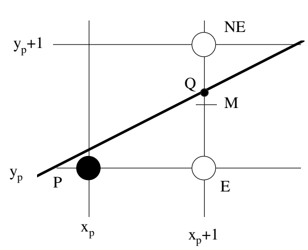
El algoritmo de Bresenham se generaliza para líneas con una pendiente arbitraria al considerar la simetría entre los diversos octantes y cuadrantes del plano de xy.
Para una línea con una pendiente $m > 1$, intercambiamos las funciones de las direcciones de x y y, o sea, pasamos a lo largo de y en pasos unitarios y calculamos los valores sucesivos de x que se aproximan mas a la trayectoria de la línea.
Asimismo, podemos revisar el programa para trazar pixels iniciando desde cualquier extremo.
Si la posición inicial para una línea con una pendiente positiva es el extremo derecho, tanto x como y disminuyen conforme pasamos de derecha a izquierda.
Con el fin de asegurarnos de que los mismos pixeles se tracen sin que importe el extremo en que se comienza, se seleccionara el pixel superior (o inferior) cuando se pase exactamente en el medio ($d_1 = d_2$).
En el caso de pendientes negativas, los procedimientos son similares excepto que ahora, una coordenada decrece conforme la otra aumenta.
Por ultimo, es posible manejar los casos especiales por separado.
Las líneas horizontales ($\Delta y = 0$), las líneas verticales ($\Delta x = 0$) y las diagonales $|\Delta y| = |\Delta x|$ se pueden cargar en forma directa sin procesarlas mediante el algoritmo para el trazo de líneas.

1.1.4 Algoritmo de Punto Medio para la Línea

Una extension al algoritmo de Bresenham es la *técnica del punto medio* (*midpoint technique*). Publicada por primera vez por Pitteway (1967) y adaptada por Van Aken (1984) y otros investigadores.
Para líneas y círculos de enteros, la formulación de punto medio, como la muestra Van Aken (1985), se reduce a la formulación de Bresenham y por lo tanto se generan los mismos pixeles.
Bresenham (1977) mostró que este algoritmo de línea y circulo de enteros proveen la mejor aproximación a líneas y círculos verdaderos al minimizar el error (distancia) a las primitivas reales.
Kappel (1985) discute los efectos de varios ¡criterios de error.

Se asume que la pendiente de la línea es entre 0 y 1.
Otras pendientes se pueden manejar como reflexiones sobre ejes principales.
Se define a (x_0,y_0) como el extremo inferior izquierdo y (x_1,y_1) como el extremo superior derecho.

Consideremos la siguiente figura.



Asumimos que se ha seleccionado el pixel P en (x_p,y_p) y ahora se debe escoger entre el pixel de un incremento a la derecha (pixel del este, E) o el pixel un incremento para la derecha y otro para arriba (pixel del noreste, NE).
Si Q es el punto de intersección de la línea real con la columna $x = x_p + 1$.

En la formulación de Bresenham, la diferencia entre las distancias verticales de *E* y *NE* a *Q* se computan, y el signo de la diferencia, *p*, se usa para seleccionar el pixel cuya distancia de *Q* sea menor como la mejor aproximación a la línea real.
En la formulación del punto medio, se observa de que lado de la línea el punto medio *M* esta.
Es fácil ver, si el punto medio esta sobre la línea real, el pixel *E* esta mas cercano a la línea; si el punto medio esta debajo de la línea real, el pixel *NE* es mas cercano a la línea.
La línea puede pasar entre *E* y *NE* o ambos pixeles están de un lado de la línea, pero en cualquiera de los casos, la prueba del punto medio selecciona el pixel mas cercano.
También, el error, o sea, la distancia vertical entre el pixel escogido y la línea real, siempre es $\leq 1/2$.

La línea se representa con una función implícita (esta representación se extiende muy bien para la formulación de círculos) con coeficientes *A*, *B*, y *C*:

(15) $$f(x,y) = Ax + By + C = 0$$

Si $\Delta y = y_1 - y_0$, y $\Delta x = x_1 - x_0$, la ecuación de la línea tiene la forma de

(16) $$y = (\Delta y / \Delta \mathrm{x})\, x + b$$

Por lo tanto

(17) $$f(x,y) = \Delta y\, x - \Delta \mathrm{x}\, y + b\, \Delta x = 0$$

donde A = Δy, B = $-\Delta x$, y $C = b\,\Delta x$
(Es importante para este algoritmo que A > 0, lo cual se logra ya que se escogió $\Delta y > 0$.)

Es fácil de verificar que:
1. Un punto en la línea está representado por: $f(x,y) = 0$.
2. Un punto debajo de la línea está representado por valores positivos, o sea: $f(x,y) > 0$.
3. Un punto encima de la línea está representado por valores negativos, o sea: $f(x,y) < 0$.

Esto se puede demostrar de la siguiente forma, para $\Delta y > 0$ y $\Delta x > 0$:

Si tomamos cualquier punto (x,y) sobre la línea, tenemos:

$$f(x,y) = \Delta y\, x - \Delta x\, y + b\, \Delta x = 0$$

Si tomamos un punto cualquiera y_s y la misma coordenada anterior de *x*, tenemos:

$$f(x, y_s) = \Delta y\, x - \Delta x\, y_s + b\, \Delta x = k$$

donde *k* es alguna valor resultante de aplicar la nueva coordenada a la ecuación de la línea. ($k = 0$ solo si el punto está sobre la línea.)

Si restamos estas dos ecuaciones, tenemos:

$$f(x, y_s) - f(x,y) = (\Delta y\, x - \Delta x\, y_s + b\, \Delta x) - (\Delta y\, x - \Delta x\, y + b\, \Delta x) = k$$

lo cual es equivalente a:

$$- \Delta x\, y_s + \Delta x\, y = k$$
$$- y_s + y = k/\Delta x$$

Esto indica que pueden existir dos casos:

1. Si $-y_s+y>0$ entonces $k/\Delta x>0$, o sea, si $y>y_s$ entonces $k/\Delta x>0$, y un punto debajo de la línea corresponde a un valor positivo de k.
2. Si $-y_s+y<0$ entonces $k/\Delta x<0$, o sea, si $y<y_s$ entonces $k/\Delta x<0$, y un punto encima de la línea corresponde a un valor negativo de k.

Para aplicar el criterio del punto medio, solo se necesita computar $f(M)=f(x_p+1,y_p+1/2)$ y probar su signo.
Como la decision se basa en el valor de la función en el punto $(x_p+1,y_p+1/2)$, se define una variable de decision $p=(x_p+1,y_p+1/2)$.

Por definición

$$p=f(x_p+1,y_p+1/2)=\Delta y(x_p+1)-\Delta x\,(y_p+1/2)+b\,\Delta x$$

Y existen tres posibilidades para el valor de:

1. Si $p>0$, el punto medio está debajo de la línea, y el pixel *NE* es el mas cercano
2. Si $p<0$, el punto medio está arriba de la línea, y el pixel *E* es el mas cercano
3. Si $p=0$, el punto medio está sobre la línea, y podemos escoger cualquiera de los dos pixeles, por ejemplo *E*.

Luego de esto se analiza que ocurre en la siguiente ubicación de *M* con respecto al punto recién trazado, y así mismo el valor de *p* para la siguiente columna. Estos valores dependen, por su puesto, si escogimos *E* o *EN*:

1. Si *E* se escoge, *M* se incrementa por un paso en la dirección de *x*.
Entonces

$$p_{\text{nuevo}}=f(x_p+2,y_p+1/2)=A(x_p+2)+B(y_p+1/2)+C$$

Dado el *p* anterior:

$$p_{\text{viejo}}=f(x_p+1,y_p+1/2)=A(x_p+1)+B(y_p+1/2)+C$$

Restando p_{viejo} de p_{nuevo} para obtener la diferencia de incremento, se escribe

$$p_{\text{nuevo}}=p_{\text{viejo}}+A$$

Se llama al incremento para sumar después de que se escoge *E*, Δ_E,

$$\Delta_E=A=\Delta y.$$

En otras palabras, se puede derivar el valor de la variable de decision al siguiente paso incrementando del valor del paso actual sin tener que computar $f(M)$ directamente, simplemente sumando Δ_E.

2. Si se escoge *NE*, *M* se incrementa por un paso en ambas direcciones de *x* y *y*.
Entonces

$$p_{\text{nuevo}}=f(x_p+2,y_p+3/2)=A(x_p+2)+B(y_p+3/2)+C$$

Restando p_{viejo} de p_{nuevo} para obtener la diferencia de incremento, se escribe

$$p_{\text{nuevo}}=p_{\text{viejo}}+A+B$$

Se llama al incremento para sumar después de que se escoge *NE*, Δ_{NE},

$$\Delta_{NE} = A+B = \Delta y\text{-}\Delta x$$

Para resumir la técnica incremental de punto medio, en cada paso el algoritmo escoge entre los dos pixeles basado en el signo de la variable de decision calculada en la iteración previa; entonces se actualiza la variable de decision sumando Δ_E o Δ_{NE}, al valor viejo, dependiendo de la selección de pixel.

Como el primer pixel es simplemente el primer extremo (x_0,y_0), se puede calcular directamente el valor inicial de p para escoger entre E y NE.
El primer punto medio esta en $(x_0+1,\ y_0+1/2)$, y

$$\begin{aligned} f(x_0+1,\ y_0+1/2) &= A(x_0+1) + B(y_0+1/2) + C \\ &= A\ x_0 + B\ y_0 + A + B\ /2 + C \\ &= f(x_0,\ y_0) + A + B\ /2 \end{aligned}$$

Pero $(x_0,\ y_0)$ es un punto es la línea y $f(x_0,\ y_0)$ es por lo tanto 0; entonces

$$p_{\text{inicial}} = A + B\ /2 = \Delta\ y - \Delta x\ /2$$

Usando p_{inicial} se escoge el segundo pixel, etc.
Para eliminar la fracción en p_{inicial} se redefine la función original multiplicándola por 2

$$f(x,y) = 2\ (\ Ax + B\ y + C\ )$$

Esto multiplica cada constante y la variable de decisión (y los incrementos Δ_E y Δ_{NE}) por 2 pero esto no afecta el signo de la variable de decisión, que es lo que importa en la prueba.

Las necesidades aritméticas de p_{nuevo} para cualquier paso es simplemente una suma entera.
No se utilizan multiplicaciones, lo que consume mas tiempo.

En resumen, los pasos son:

1. Se capturan los dos extremos de la línea y se almacena el extremo izquierdo en (x_0,y_0).
2. Se carga (x_0,y_0) en el bufer de estructura, o sea, se traza el primer punto.
3. Se calculan las constantes Δy, Δx, $2\Delta y$, $2\Delta y\text{-}2\Delta x$, y se obtiene el valor inicial para el parámetro de decisión como $p_0 = 2\ \Delta y - \Delta x$.
4. En cada x_k a lo largo de la línea, que inicia en $k = 0$, se efectúa la prueba siguiente: si $p_k < 0$, el siguiente punto que se debe trazar es (x_k+1,y_k) y $p_k +1 = p_k + 2\ \Delta y$. De otro modo, el siguiente punto en trazarse es (x_k+1,y_k+1) y $p_k +1 = p_k + 2\ \Delta y - 2\Delta x$.
5. Se repite el paso 4 otras Δx veces.

Nótese que aunque el desarrollo propio del algoritmo es diferente al básico de Bresenham, el resultado final de la extensión de punto medio es exactamente igual al básico de Bresenham.

Ejemplo

Para ilustrar el algoritmo, utilicemos la línea con extremos (20,10) y (30,18).
Esta línea tiene una pendiente de 0.8, con

$$\Delta x = 10,\ \Delta y = 8$$

El parámetro de decisión inicial tiene el valor

$$p_0 = 2\ \Delta y - \Delta x = 6$$

y los incrementos para calcular parámetros de decisión sucesivos son

$$2\Delta y = 16,\ 2\Delta y - 2\Delta x = -4$$

Trazamos el punto inicial $(x_0,y_0) = (20,10)$ y determinamos las posiciones de pixel sucesivos a lo largo de la trayectoria de la línea a partir del parámetro de decisión como

k	p_k	(x_k+1,y_k+1)
0	6	(21,11)
1	2	(22,12)
2	-2	(23,12)
3	14	(24,13)
4	10	(25,14)
5	6	(26,15)
6	2	(27,16)
7	-2	(28,16)
8	14	(29,17)
9	10	(30,18)

El programa se muestra a continuación (solo para $0<|m|<1$):

```
void LineMidPoint(Display* display, Window win, GC gc, int x0, int y0, int x1, int y1)
{
        int x, y, dx, dy, xend, p, incE, incNE;

        dx = x1 - x0;
        dy = y1 - y0;
        p = 2*dy - dx;
        incE = 2*dy;
        incNE = 2*(dy-dx);
/* determinar que punto usar para empezar, cual para terminar */
        if (x0 > x1) {
                x = x1;
                y = y1;
                xend = x0;
        }
        else {
                x = x0;
                y = y0;
                xend = x1;
        }
/* se cicla hasta llegar al extremo de la línea */
        while (x <= xend)
        {
            XDrawPoint(display,win,gc,x,y);
            x = x + 1;
            if (p <= 0)
                p = p + incE;
            else {
                y = y + 1;
                p = p + incNE;
            }
        }
}
```

Es importante que el trazo partiendo de cualquiera de ambos extremos contenga exactamente los mismos pixeles.
El único lugar donde la selección de pixel depende de la dirección de la línea es cuando pasa exactamente por el medio, y la variable de decision es cero; de izquierda a derecha se escoge *E*.
Por simetría al ir de la derecha a la izquierda se esperaría escoger *W* cuando p=0, pero eso escogería un pixel una unidad hacia arriba en y relativo al escogido al pixel escogido de izquierda a derecha.
Por lo tanto se necesita escoger *SW* cuando p=0 para el trazo de derecha a izquierda.
Ajustes similares se deben hacer para líneas en otras pendientes.

La solución alternativa de intercambiar los extremos de la línea para que el trazo siempre proceda en la misma dirección no trabaja cuando se usan estilos de línea.
Si el algoritmo siempre pone a los extremos en orden canónico, el patrón iría de izquierda a derecha para un segmento, y de derecha a izquierda para otro adjunto, como función de la pendiente de la segunda línea; esto crearía una discontinuidad inesperada en el vértice compartido.

1.2 Algoritmos de Generación de Circunferencias

Una circunferencia se define como un conjunto de puntos que se encuentran, en su totalidad, a una distancia r de una posición central (x_c, y_c).
Esta relación de distancia se expresa por medio del teorema de Pitarrosa en coordenadas cartesianas como

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

1.2.1 Algoritmo Básico

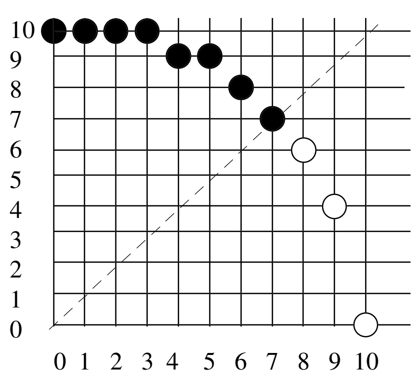
Se podría utilizar una ecuación para calcular la posición de los puntos de una circunferencia pasando a lo largo del eje de las x en pasos unitarios de $x_c - r$ a $x_c + r$ y calcular los valores correspondientes de y en cada posición como

(25) $$y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$$

Un método para dibujar el circulo es aprovechar la simetría de los cuadrantes, dibujando solo uno y trazando los puntos simétricos en los demás cuadrantes.
Para el primer cuadrante se incrementaría x de 0 a r en pasos de 1, resolviendo para cada y positiva.
El trazo para un circulo con radio de 10, centrado en (0,0), utilizando este algoritmo básico, se muestra a continuación:

x	$r^2 - x^2$	$\sqrt{r^2 - x^2}$	(x,y)
0	100	10	(0,10)
1	99	9.95	(1,10)
2	96	9.80	(2,10)
3	91	9.54	(3,10)
4	84	9.17	(4,9)
5	75	8.66	(5,9)
6	64	8	(6,8)
7	51	7.14	(7,7)
8	36	6	(8,6)
9	19	4.36	(9,4)
10	0	0	(10,0)

El resultado de esto se muestra en la siguiente figura:



No obstante, este no es el mejor método para generar una circunferencia.

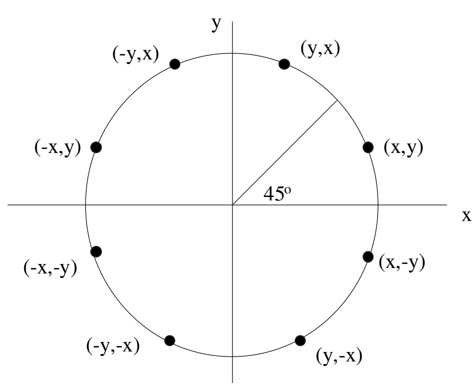
Un problema con este planteamiento es que implica cálculos considerables en cada paso.
Por otro lado, el espacio entre las posiciones de pixel trazadas no es uniforme (aliasing, con mayor concentración de pixeles cuanto mas horizontal sea la línea).
Se podría ajustar el espacio al intercambiar *x* y *y* (pasar por los valores de *y* y calcular los valores de *x*) siempre que el valor absoluto de la pendiente de la circunferencia sea mayor que 1.
Pero esto solo incrementa el calculo y el procesamiento que el algoritmo requiere.

Otra manera de eliminar el espacio irregular consiste en calcular los puntos a lo largo de la frontera circular utilizando las coordenadas polares r y θ.
Al expresar la ecuación de la circunferencia en forma polar parametrica, se obtiene el par de ecuaciones

(26) $$x = x_c + r\cos\theta, \quad y = y_c + r\sin\theta$$

Cuando un despliegue se genera con estas ecuaciones utilizando un tamaño de paso angular fijo, una circunferencia se traza con puntos equidistantes a lo largo de la misma.
El tamaño de paso seleccionado para θ depende de la aplicación, así como del dispositivo de despliegue.
Las separaciones angulares mas grandes a lo largo de la circunferencia se pueden unir con segmentos de línea recta a fin de aproximarse a la trayectoria circular.
En el caso de un frontera mas continua, se puede establecer el tamaño de paso como $1/r$.
Esto hace que se tracen posiciones de pixel que están aproximadamente una unidad aparte.

Es posible también reducir el calculo al considerar la simetría de las circunferencias.
La formula de la circunferencia es similar en cada cuadrante.
Se puede generar la sección circular del segundo cuadrante del plano *xy* al notar que las dos secciones circulares son simétricas con respecto al eje de la *y*.
Y las secciones circulares del tercero y cuarto cuadrante se pueden obtener a partir de las secciones del primero y segundo cuadrante al considerar la simetría en relación con el eje de las *x*.
Se puede llevar esto un paso mas adelante y señalar que también hay simetría entre octantes.
Las secciones circulares en octantes adyacentes dentro de un cuadrante son simétricas con respecto de la línea de 45°, que divide los dos octantes.
En la siguiente figura se ilustran estas condiciones de simetría, donde un punto, en la posición (x,y), en un sector de una octava parte de una circunferencia se diagrama en los siete puntos de la misma en los demás octantes del plano *xy*.
Al aprovechar la simetría de la circunferencia de esta manera, se puede generar todas las posiciones de pixel alrededor de una circunferencia, calculando solo los puntos dentro del sector $x = 0$ y $x = y$.



Determinar las posiciones de pixel a lo largo de una circunferencia mediante las ecuaciones anteriores también requiere una cantidad considerable de tiempo de calculo.
La ecuación cartesiana (24) implica multiplicaciones y cálculos de raíces cuadradas, en tanto que las ecuaciones paramétricas (26) contiene multiplicaciones y cálculos trigonométricos.

Los algoritmos de circunferencia mas efectivos se basan en el calculo en incremento de los parámetros de decisión, como en el algoritmo de línea de Bresenham, que solo implica operaciones de enteros.

1.2.2 Algoritmo de Punto Medio para la Circunferencia

Bresenham (1977) desarrollo un generador incremental que genera todos los puntos en un circulo centrado en el origen, utilizando un algoritmo de punto medio para la circunferencia.
El código resultante es similar al patentado por Bresenham (1983).

El algoritmo de línea de Bresenham para despliegues de rastreo se adapta a la generación de circunferencias al establecer los parámetros de decisión para identificar el pixel mas cercano a la circunferencia en cada paso del muestreo.
Como la ecuación de circunferencia (24) no es lineal, evaluaciones de la raíz cuadrada serian necesarias para calcular las distancias del pixel de la trayectoria circular.
El algoritmo de línea de Bresenham evita los cálculos de raíces cuadradas al comparar los cuadrados de las distancias de separación entre pixeles.

Al igual que en el algoritmo de línea de rastreo, se efectúa un muestreo en intervalos unitarios y se determina la posición del pixel mas cercano a la trayectoria especifica de la circunferencia en cada paso.
Para un radio r determinado y una posición central (x_c, y_c), se puede establecer primero el algoritmo para calcular las posiciones de pixel alrededor de una trayectoria circular centrada en el origen de coordenadas (0,0).
Así, cada posición calculada (x,y) se mueve a su posición propia en la pantalla al sumar x_c a x y y_c a y.
A lo largo de la sección circular de $x = 0$, $x = y = r/\sqrt{2}$ en el primer cuadrante, la pendiente de la curva varia entre 0 y -1.
Por lo tanto, se puede tomar pasos unitarios en la dirección positiva de x en este octante y utilizar un parámetro de decisión para determinar cual de las dos posiciones posibles de y esta mas próxima a la trayectoria de la circunferencia en cada paso.
Las posiciones de los otros siete octantes se obtiene por simetría.

Para aplicar el método del punto medio, se define una función de circunferencia como:

(27) $$f_{\text{circ}}(x,y) = x^2 + y^2 - r^2$$

Cualquier punto (x,y) en la frontera de la circunferencia con radio r satisface la ecuación $f_{\text{circ}}(x,y) = 0$.
Si el punto esta en el interior de la circunferencia, la función de la circunferencia es negativa.
Esto se demuestra definiendo la función para ese punto dentro de la circunferencia dado por::

$$f_{\text{dentro}}(x, y_d) = x^2 + y_d^{\,2} - r^2 = p$$

y

$$f_{\text{dentro}}(x, y_d) - f_{\text{circ}}(x,y) = (x^2 + y_d^{\,2} - r^2) - (x^2 + y^2 - r^2) = p - 0$$

$$= y_d^{\,2} - y^2 = p$$

dado que, en el primer cuadrante, un punto dentro de la circunferencia significa que

$$y_d^{\,2} < y^2$$

entonces $p < 0$.

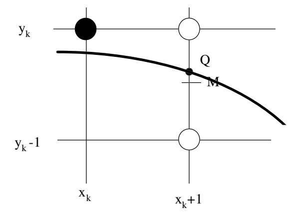
De lo contrario si el punto esta fuera de la circunferencia, $p > 0$.

Para resumir, la posición relativa se puede determinar al verificar el signo de la función de circunferencia:

(28) $f_{\text{circ}}(x,y) \begin{cases} < 0, & \text{si } (x,y) \text{ esta dentro de la frontera de la circunferencia} \\ = 0, & \text{si } (x,y) \text{ en la frontera de la circunferencia} \\ > 0, & \text{si } (x,y) \text{ esta fuera de la frontera de la circunferencia} \end{cases}$

Las pruebas de función de circunferencia de las condiciones anteriores se realizan para las posiciones medias entre los pixeles cercanos a la trayectoria de la circunferencia es el parámetro de decisión en el algoritmo de punto medio y se puede determinar cálculos incrementales para esta función como se hizo en el algoritmo de línea.

La siguiente figura muestra el punto medio entre los dos pixeles candidatos en la posición de muestreo $x_k + 1$.



Suponiendo que se acaba de trazar el pixel en (x_k, y_k), en seguida se necesita determinar si el pixel en la posición $(x_k + 1, y_k)$, o aquel en la posición $(x_k + 1, y_k - 1)$ esta mas cerca de la circunferencia.
El parámetro de decisión es la función de circunferencia (27) evaluada en el punto medio entre esos dos pixeles.

(29) $p_k = f_{\text{circ}}(x_k + 1, y_k - 1/2) = (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$

Si $p_k < 0$, este punto medio esta dentro de la circunferencia y el pixel en la línea de rastreo y_k esta mas próximo a la frontera de la circunferencia.
De otro modo, la posición media se localiza fuera de la frontera de la circunferencia o en esta y se selecciona el pixel en la línea de rastreo $y_k - 1$.

Los parámetros de decisión sucesivos se obtienen al utilizar cálculos incrementales.
Se obtiene una expresión recursiva para el siguiente parámetro de decisión cuando se evalúa la función de circunferencia en la posición de muestreo $x_{k+1} + 1 = x_k + 2$.

$$\begin{aligned} p_{k+1} &= f_{\text{circ}}(x_{k+1} + 1, y_{k+1} - 1/2) = [(x_{k+1} + 1)^2 + (y_{k+1} - 1/2)^2 - r^2 \\ &= [(x_k + 1)+1]^2 + (y_{k+1} - 1/2)^2 - r^2 \end{aligned}$$

$$\begin{aligned} p_{k+1} - p_k &= ((x_k + 2)^2 + (y_{k+1} - 1/2)^2 - r^2) - ((x_k + 1)^2 + (y_k - 1/2)^2 - r^2) \\ &= (x_k + 2)^2 + (y_{k+1} - 1/2)^2 - (x_k + 1)^2 - (y_k - 1/2)^2 \\ &= x_k^2 + 4x_k + 4 + y_{k+1}^2 - y_{k+1} + ¼ - x_k^2 - 2x_k - 1 - y_k^2 + y_k - ¼ \\ &= 2x_k + 3 + y_{k+1}^2 - y_{k+1} - y_k^2 + y_k \\ &= 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \end{aligned}$$

(30) $p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$

donde y_{k+1} es ya sea y_k o $y_k - 1$, dependiendo del signo de p_k.(y_{k+1} corresponde a x_{k+1})

Los incrementos para obtener p_{k+1} se calculan según el signo de p_k
Si p_k es negativa, $y_{k+1} = y_k$

$$\begin{aligned} p_{k+1} &= p_k + 2(x_k + 1) + (y_k^2 - y_k^2) - (y_k - y_k) + 1 \\ &= p_k + 2(x_k + 1) + 1 \\ &= p_k + 2x_{k+1} + 1 \end{aligned}$$

Si p_k es positiva, $y_{k+1} = y_k - 1$

$$
\begin{aligned}
p_{k+1} &= p_k + 2(x_k + 1) + ((y_k - 1)^2 - y_k{}^2) - ((y_k - 1) - y_k) + 1 \\
&= p_k + 2(x_k + 1) + (y_k^2 - 2\,y_k + 1 - y_k{}^2) - (y_k - 1 - y_k) + 1 \\
&= p_k + 2(x_k + 1) - 2\,y_k + 1 + 1 + 1 \\
&= p_k + 2(x_k + 1) - 2(\,y_k - 1) + 1 \\
&= p_k + 2x_{k+1} + 1 - 2y_{k+1}
\end{aligned}
$$

La evaluación de los términos $2x_{k+1}$ y $2y_{k+1}$ también pueden efectuarse de modo incremental como

$$
\begin{aligned}
2x_{k+1} &= 2x_k + 2 \\
2y_{k+1} &= 2y_k - 2
\end{aligned}
$$

La posición de inicio es $(x_0, y_0) = (0,r)$, donde $2x_0=0$, $2\,y_0 = 2r$.
Cada valor sucesivo se obtiene al sumar 2 al valor previo de $2x$ y sustrayendo 2 del valor previo de $2y$.

El parámetro de decisión inicial se obtiene al evaluar la función de circunferencia en la posición de inicio $(x_0, y_0) = (0,r)$

$$p_0 = f_{\text{circ}}(1, r - 1/2) = 1 + (r - 1/2)^2 - r^2$$

o

$$p_0 = 1 + r^2 - \text{r} + ¼ - r^2 = 5/4 - r \tag{31}$$

Si el radio r se especifica como un entero, se puede redondear simplemente p_0 a

$$p_0 = 1 - r \text{ (para } r \text{ como un entero).}$$

Puesto que todos los incrementos son enteros.

Al igual que en el algoritmo para el trazo de líneas de Bresenham, el métodos del punto medio calcula las posiciones de pixel a lo largo de una circunferencia utilizando adiciones y sustracciones de enteros, si se supone que los parámetros de la circunferencia se especifican en coordenadas enteras de pantalla.
Se puede resumir los pasos del algoritmo de la circunferencia de punto medio como sigue:

1. Se capturan el radio r y el centro de la circunferencia (x_c,y_c) y se obtiene el primer punto de una circunferencia centrada en el origen como $(x_0,y_0) = (0,r)$.
2. Se calcula el valor inicial del parámetro de decisión como
   $p_0 = 5/4 - r$.
3. En cada posición x_k, iniciando en k=0, se efectúa la prueba siguiente:
   Si $p_k < 0$, el siguiente punto a lo largo de la circunferencia centrada en (0,0) es
   (x_k+1,y_k) y $p_{k+1} = p_k + 2x_{k+1}+1 = p_k + 2x_k+3$.
   De otro modo, el siguiente punto a lo largo de la circunferencia es
   (x_k+1,y_k-1) y $p_{k+1} = p_k + 2x_{k+1}+1 - 2y_{k+1} = p_k + 2x_k - 2y_k+5$,
   donde $2x_{k+1} = 2x_k + 2$ y $2y_{k+1} = 2y_k - 2$.
4. Se determinan puntos de simetría en los siete octantes.
5. Se mueve cada posición de pixel calculada (x,y) a la trayectoria circular centrada en (x_c,y_c) y se traza los valores de las coordenadas
   $x = x + x_c$, $y = y + y_c$.
6. Se repiten los pasos 3 a 5 hasta que $x \geq y$.

<u>Ejemplo</u>

Dado el radio de una circunferencia r=10, se demuestra el algoritmo de la circunferencia de punto medio al determinar las posiciones a lo largo del octante de la circunferencia en el primer cuadrante desde x=0 hasta x=y.

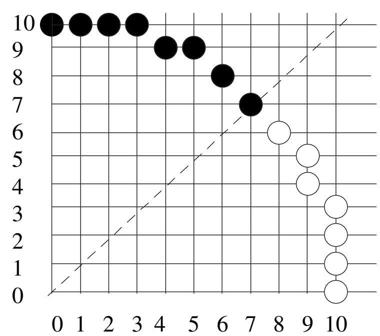
El valor inicial del parámetro de decisión es $p_0 = 1 - r = -9$

En el caso de la circunferencia centrada en el origen de las coordenadas, el punto inicial es
$(x_0,y_0) = (0,10)$
y los términos de incremento iniciales para calcular los parámetros de decisión son
$2\ x_0 = 0,\qquad 2\ y_0 = 20$

Los valores sucesivos del parámetro de decisión y las posiciones a lo largo de la trayectoria de la circunferencia se calculan mediante el método del punto medio como

k	p_k	(x_{k+1},y_{k+1})	$2\ x_{k+1}$	$2y_{k+1}$
0	-9	(1,10)	2	20
1	-6	(2,10)	4	20
2	-1	(3,10)	6	20
3	6	(4,9)	8	18
4	-3	(5,9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14

En la siguiente figura se muestran las posiciones generadas en el primer cuadrante:



El procedimiento siguiente despliega una circunferencia de rastreo utilizando el algoritmo de punto medio. La entrada para el procedimiento son las coordenadas para el centro y radio de la circunferencia.

```
void PlotPoint(Display* display, Window win, GC gc, int xc, int yc, int x, int y)
{
/* draw symmetry points */
        XDrawPoint(display,win,gc,xc + x,yc + y);
        XDrawPoint(display,win,gc,xc - x,yc + y);
        XDrawPoint(display,win,gc,xc + x,yc - y);
        XDrawPoint(display,win,gc,xc - x,yc - y);
        XDrawPoint(display,win,gc,xc + y,yc + x);
        XDrawPoint(display,win,gc,xc - y,yc + x);
        XDrawPoint(display,win,gc,xc + y,yc - x);
        XDrawPoint(display,win,gc,xc - y,yc - x);
}
void CircleMidPoint(int xc, int yc, int r)
{
        int x, y, p;

        x = 0;
        y = r;
        p = 1 - r;
        PlotPoint(xc,yc,x,y);
/* se cicla hasta trazar todo un octante */
        while (x < y)
        {
            x = x + 1;
            if (p < 0)
                p = p + 2*x + 1;
            else {
```

```
            y = y - 1;
            p = p + 2*(x - y) + 1;
        }
        PlotPoint(xc,yc,x,y);
    }
}
```

1.2.3 Algoritmo con Diferencias Parciales de Segundo Orden

Se puede inclusive mejorar el rendimiento de este algoritmo utilizando mas extensivamente cálculos incrementales. Esto se hace utilizando diferencias parciales de primer y segundo orden.
La estrategia es evaluar la función directamente en dos puntos adyacentes, calcular su diferencia (la cual, para polinomios, es siempre un polinomio de menor grado), y aplicar esa diferencia en cada iteración.
Como para cada punto se consideran dos alternativas, para un p_k se pudieran computar dos valores distintos de p_{k+1} dependiendo de la comparación de p_k con 0.
La idea es computar de antemano los dos valores posibles de p_{k+1} y calcular a continuación las diferentes posibilidades de p_{k+2} considerando las diferentes alternativas.

Si se escoge y_k (E) en la iteración actual, el punto de evaluación se mueve de (x_k,y_k) a (x_k+1,y_k), o sea $p_k < 0$.

La diferencia de primer orden sería

$$\Delta_{E\text{viejo}} \text{ en } (x_k,y_k) = p_{k+1} - p_k = 2x_{k+1}+1 = 2x_k+3$$

y si se volviera a trazar el punto E:

$$\Delta_{E\text{nuevo}} \text{ en } (x_k+1,y_k) = p_{k+2} - p_{k+1} = 2(x_k+1)+3 = 2x_k+5$$

Y la diferencia de segundo orden es

$$\Delta_{E\text{nuevo}} - \Delta_{E\text{viejo}} = (2x_k+5) - (2x_k+3) = 2$$

De forma similar la diferencia de primer orden para el calculo de SE en ese mismo punto es

$$\Delta_{SE\text{viejo}} \text{ en } (x_k,y_k) = p_{k+1} - p_k = 2x_{k+1}-2y_{k+1}+1 = 2x_k-2y_k+5$$

Por lo tanto, si se trazara a continuación el punto SE seria:

$$\Delta_{SE\text{nuevo}} \text{ en } (x_k+1,y_k) = p_{k+2} - p_{k+1} = 2(x_k+1) - 2y_k + 5 = 2x_k - 2y_k +7$$

Y la diferencia de segundo orden es

$$\Delta_{SE\text{nuevo}} - \Delta_{SE\text{viejo}} = (2x_k-2y_k+7) - (2x_k-2y_k+5) = 2$$

Si se escoge y_k-1 (SE) en la iteración actual, el punto de evaluación se mueve de (x_k,y_k) a (x_k+1,y_k-1), o sea $p_k > 0$.

La diferencia de primer orden es

$$\Delta_{E\text{viejo}} \text{ en } (x_k,y_k) = p_{k+1} - p_k = 2x_{k+1}+1 = 2x_k+3$$

Por lo tanto, si se trazara a continuación el punto E seria:

$$\Delta_{E\text{nuevo}} \text{ en } (x_k+1,y_k-1) = p_{k+2} - p_{k+1} = 2(x_k+1)+3 = 2x_k+5$$

Y la diferencia de segundo orden es

$\Delta_{E\text{nuevo}} - \Delta_{E\text{viejo}} = (2x_k + 5) - (2x_k + 3) = 2$

De forma similar la diferencia de primer orden para SE es

$\Delta_{SE\text{viejo}}$ en $(x_k, y_k) = p_{k+1} - p_k = 2x_{k+1} - 2y_{k+1} + 1 = 2x_k - 2y_k + 5$

y si se volviera a trazar el punto SE:

$\Delta_{SE\text{nuevo}}$ en $(x_k + 1, y_k - 1) = p_{k+2} - p_{k+1} = 2(x_k + 1) - 2(y_k - 1) + 5 = 2x_k - 2y_k + 9$

Y la diferencia de segundo orden es

$\Delta_{SE\text{nuevo}} - \Delta_{SE\text{viejo}} = (2x_k - 2y_k + 9) - (2x_k - 2y_k + 5) = 4$

El algoritmo revisado consiste de los siguientes pasos:

1. Escoger el pixel basado en el signo de la variable p_k computado durante la iteración previa.
2. Actualizar la variable de decisión p_k con Δ_E o Δ_{SE}, usando el valor del Δ correspondiente computado durante la iteración previa.
3. Actualizar los Δs para tener en cuenta el movimiento al pixel nuevo, usando las diferencias de constantes previamente computadas.
4. Hacer el movimiento.

Δ_E y Δ_{SE} se inicializan usando el pixel inicial (0,r), con

$p_0 = 1 - r$;
$\Delta_E = 2x_k + 3$;
$\Delta_{SE} = 2x_k - 2y_k + 5$.

Ejemplo:

Para el circulo anterior con $r = 10$ y con centro (0,0), los valores iniciales para el punto $(0,r)$ se calculan de la siguiente forma:

$p_0 = 1 - r = -9$
$\Delta_E = 2x_k + 3 = 3$
$\Delta_{SE} = 2x_k - 2y_k + 5 = -20 + 5 = -15$

Los valores completos son y punto a dibujarse son:

k	p_k	*deltaE*	*deltaSE*	(x_{k+1}, y_{k+1})
0	-9	*3	-15	(1,10)
1	-6	*5	-13	(2,10)
2	-1	*7	-11	(3,10)
3	6	9	*-9	(4,9)
4	-3	*11	-5	(5,9)
5	8	13	*-3	(6,8)
6	5	15	1	(7,7)

* muestra que delta se agrega al siguiente calculo de p_k.

Se obtiene no solamente el mismo trazo sino también los mismo puntos de decisión p_k que con el método anterior.

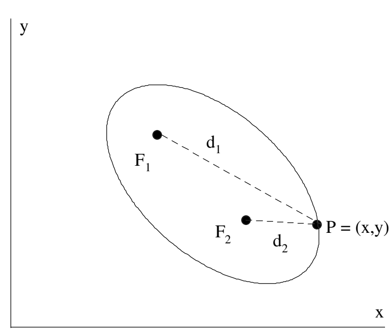
La función revisada seria:

```
void PlotPoint(Display* display, Window win, GC gc, int xc, int yc, int x, int y)
{
/* draw symmetry points */
        XDrawPoint(display,win,gc,xc + x,yc + y);
        XDrawPoint(display,win,gc,xc - x,yc + y);
        XDrawPoint(display,win,gc,xc + x,yc - y);
        XDrawPoint(display,win,gc,xc - x,yc - y);
        XDrawPoint(display,win,gc,xc + y,yc + x);
        XDrawPoint(display,win,gc,xc - y,yc + x);
        XDrawPoint(display,win,gc,xc + y,yc - x);
        XDrawPoint(display,win,gc,xc - y,yc - x);
}
void CircleMidPoint(Display* display, Window win, GC gc, int xc, int yc, int r)
{
        int x, y, p, deltaE, deltaSE;

        x = 0;
        y = r;
        p = 1 - r;
        deltaE = 3;
        deltaSE = 5 - r*2;
        PlotPoint(display,win,gc,xc,yc,x,y);
/* se cicla hasta trazar todo un octante */
        while (x < y)
        {
            x = x + 1;
            if (p < 0) {       /* caso1: pk+1 = pk + deltaEnuevo */
                p = p + deltaE;
                deltaE = deltaE + 2;
                deltaSE = deltaSE + 2;
            }
            else {             /* caso2: pk+1 = pk + deltaSEnuevo */

                y = y - 1;
                p = p + deltaSE;
                deltaE = deltaE + 2;
                deltaSE = deltaSE + 4;
            }
            PlotPoint(display,win,gc,xc,yc,x,y);
        }
}
```

1.3 Algoritmos de Generación de Elipses

Expresado de forma ambigua, una elipse es una circunferencia alargada.
Por lo tanto, las curvas elípticas se pueden generar al modificar los procedimientos para el trazo de circunferencias con el fin de considerar las diversas dimensiones de una elipse a lo largo de los ejes mayor y menor.

Una elipse se define como el conjunto de puntos en que la suma de las distancias desde dos posiciones fijas (*focos*) sea la misma para todos los puntos.



Si las distancias de los dos focos desde cualquier punto P=(x,y) en la elipse se representan como d_1 y d_2, entonces la ecuación general de una elipse puede expresarse como

$$d_1 + d_2 = \text{constante}$$

Al expresar d_1 y d_2 en términos de las coordenadas focales $F_1 = (x_1,y_1)$ y $F_2 = (x_2,y_2)$ se tiene

$$\sqrt{(x-x_1)^2+(y-y_1)^2} + \sqrt{(x-x_2)^2+(y-y_2)^2} = \text{constante}$$

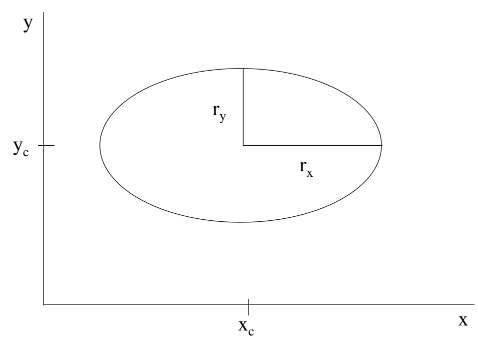
Si elevamos al cuadrado esta ecuación, aislamos el radical restante y luego la elevamos al cuadrado una vez mas, podemos volver a expresar la ecuación general de la elipse en la forma

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

donde los coeficientes A, B, C, D, E, y F se evalúan en términos de las coordenadas focales y las dimensiones de los ejes mayor y menor de la elipse.
El eje mayor es el segmento de línea recta que se extiende desde una lado de la elipse al otra a través de los focos. El eje menor abarca la dimensión mas corta de la elipse, dividiendo en dos partes el eje mayor en la posición central entre los dos focos.

1.3.1 Algoritmo Básico

Las ecuaciones de la elipse se simplifican, en gran medida, si se orientan los ejes mayor y menor para alinearse con los ejes de las coordenadas, como se muestra en la siguiente figura.



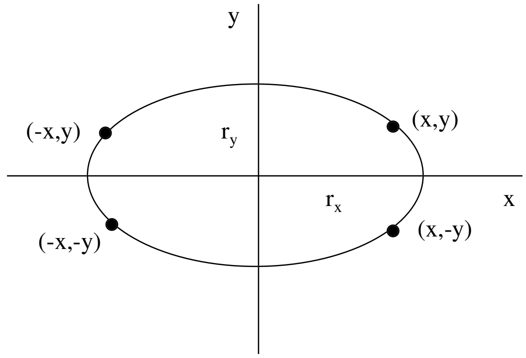
El parámetro r_x por convención designa el eje mayor y el parámetro r_y designa el eje menor.
La ecuación de la elipse puede expresarse en termino de las coordenadas del centro de la elipse y los parámetros r_x y r_y, como

(35) $$\left(\frac{x-x_c}{r_x}\right)^2 + \left(\frac{y-y_c}{r_y}\right)^2 = 1$$

Al utilizar las coordenadas polares r y θ, también es posible describir la elipse en posición estándar con las ecuaciones parametricas:

(36) $$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$

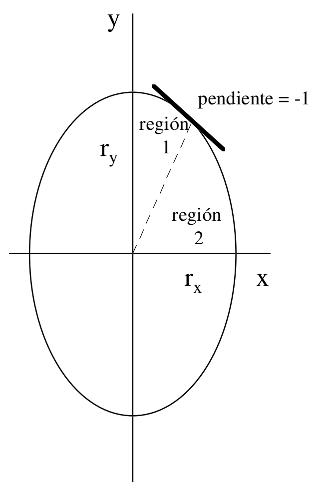
Se puede aplicar consideraciones sobre la simetría para reducir aun mas los cálculos.
Una elipse en posición estándar es simétrica entre cuadrantes, pero a diferencia de la circunferencia, no es simétrica entre los dos octantes de un cuadrante.
De este modo, debemos calcular las posiciones de pixel a lo largo del arco elíptico a través de una cuadrante, entonces obtenemos por simetría las tres posiciones de los otros tres cuadrantes.



1.3.2 Algoritmo de Punto Medio para la Elipse

El planteamiento que se utiliza aquí es similar a aquel empleado en el despliegue de una circunferencia.
Dado los parámetros r_x, r_y, (x_c, y_c), se determina los puntos (x, y) para una elipse en posición estándar centrada en el origen y luego se altera los puntos, de modo que la elipse este centrada en (x_c, y_c).
Si se desea también desplegar la elipse en posición no estándar, entonces se podría rotar sobre las coordenadas de su centro para reorientar los ejes mayor y menor (algo que se vera mas adelante en las secciones que contemplan rotaciones).

El método de punto medio para elipse se aplica a lo largo del primer cuadrante en dos partes, de acuerdo a una elipse con la pendiente $r_x < r_y$, como se muestra a continuación.



Se procesa este cuadrante tomando pasos unitarios en la dirección de x donde la pendiente de la curva tiene una magnitud menor que 1 y tomando pasos unitarios en la dirección de y donde la pendiente tiene una magnitud mayor que 1.

Las regiones 1 y 2 pueden procesarse de varias maneras.
Se puede iniciar en la posición $(0, r_y)$ y pasar en el sentido del reloj a lo largo de la trayectoria elíptica en el primer cuadrante, al alternar de pasos unitarios en x a pasos unitarios en y cuando la pendiente adquiere un valor menor que -1.
De modo alternativo, se puede iniciar en $(r_x, 0)$ y seleccionar puntos en el sentido contrario al de las manecillas del reloj, alternando de pasos unitarios en y a pasos unitarios en x cuando la pendiente adquiere un valor mayor que -1.
(Con procesadores paralelos se podría calcular en forma simultánea las posiciones de pixel en las dos regiones.)

Como en el ejemplo de una implementación secuencial del algoritmo de punto medio, se toma la posición $(0, r_y)$ y se pasa a lo largo de la trayectoria de la elipse en el sentido de las manecillas del reloj a través del primer cuadrante.

Se define la función de una elipse con base en la ecuación (35) con $(x_c, y_c) = (0,0)$ como

(37) $$f_{\text{elipse}}(x,y) = r_y{}^2x^2 + r_x{}^2y^2 - r_x{}^2\, r_y{}^2 = 0$$

Si un punto esta en el interior de la elipse, la función de la elipse es negativa.
Esto se demuestra definiendo la función para ese punto dentro de la elipse dado por::

$$f_{\text{dentro}}(x, y_d) = r_y{}^2x^2 + r_x{}^2\, y_d{}^2 - r_x{}^2\, r_y{}^2 = p$$

y

$$\begin{aligned} &f_{\text{dentro}}(x, y_d) - f_{\text{elipse}}(x,y) \\ &\quad = (r_y{}^2x^2 + r_x{}^2\, y_d{}^2 - r_x{}^2\, r_y{}^2) - (r_y{}^2x^2 + r_x{}^2y^2 - r_x{}^2\, r_y{}^2) = p - 0 \\ &\quad = r_x{}^2\, y_d{}^2 - r_x{}^2\, y^2 = p \end{aligned}$$

dado que un punto dentro de la elipse significa que

$$r_x{}^2\, y_d{}^2 < r_x{}^2\, y^2$$

entonces $p < 0$.

De lo contrario si el punto esta fuera de la circunferencia, $p > 0$.

Se resume las propiedades de la siguiente forma:

(38) $$f_{\text{elipse}}(x,y) \begin{cases} < 0, & \text{si } (x,y) \text{ esta dentro de la frontera de la elipse} \\ = 0, & \text{si } (x,y) \text{ en la frontera de la elipse} \\ > 0, & \text{si } (x,y) \text{ esta fuera de la frontera de la elipse} \end{cases}$$

Así la función de elipse $f_{\text{elipse}}(x,y)$ sirve como un parámetro de decisión en el algoritmo de punto medio.
En cada posición del muestreo, se selecciona el pixel siguiente a lo largo de la trayectoria de la elipse de acuerdo con el signo de la función de elipse evaluada en el punto medio entre los dos pixeles candidatos.

Al iniciar en $(0, r_y)$ se toma pasos unitarios en la dirección de x hasta que se alcanza la frontera entre la región 1 y la región 2.
Entonces, se une los pasos unitarios en la dirección de y sobre el restante de la curva en el primer cuadrante.
En cada paso se necesita probar el valor de la pendiente de la curva.
La pendiente de la curva (la tangente) se calcula a partir de la ecuación (37) como

(39) $$dy/dx = - 2\, r_y{}^2x \,/\, 2\, r_x{}^2y$$

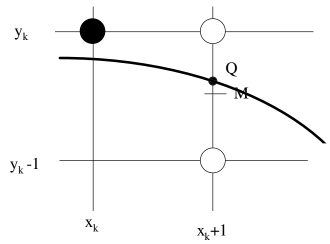
En la frontera entre la región 1 y la región 2 $dy/dx = - 1$
(dy/dx <-1 en la región 1 y dy/dx >-1 en la región 2), y en la frontera

(40) $$2\, r_y{}^2x = 2\, r_x{}^2y$$

Por lo tanto, se mueve hacia fuera de la región 1 siempre que

(41) $$2\, r_y{}^2x \geq 2\, r_x{}^2y$$

En la siguiente figura se muestra el punto medio entre los dos puntos candidatos en la posición de muestreo $x_k + 1$ en la primera región.



Si se supone que (x_k, y_k) se selecciono en el paso anterior, se determina la siguiente posición a lo largo de la trayectoria de la elipse evaluando el parámetro de decisión (es decir, la función de elipse (37)) en este punto medio:

(41) $$p1_k = f_{\text{elipse}}(x_k + 1, y_k - 1/2) = r_y^2(x_k + 1)^2 + r_x^2 (y_k - 1/2)^2 - r_x^2 r_y^2$$

Si $p1_k < 0$, el punto medio esta dentro de la elipse y el pixel en la línea de rastreo y_k esta mas cerca de la frontera de la elipse.
De otro modo, la posición media esta afuera de la frontera de la elipse o sobre esta y se selecciona el pixel en la línea de rastreo $y_k - 1$.

En la siguiente posición del muestro $x_{k+1} + 1 = x_k + 2$, el parámetro de decisión para la región 1 se evalúa como

$$p1_{k+1} = f_{\text{elipse}}(x_{k+1} + 1, y_{k+1} - 1/2) =$$
$$= r_y^2(x_k + 1 + 1)^2 + r_x^2 (y_{k+1} - 1/2)^2 - r_x^2 r_y^2$$

o

$$p1_{k+1} - p1_k =$$
$$= (r_y^2(x_k + 1 + 1)^2 + r_x^2 (y_{k+1} - 1/2)^2 - r_x^2 r_y^2) - (r_y^2(x_k + 1)^2 + r_x^2 (y_k - 1/2)^2 - r_x^2 r_y^2)$$
$$= 2 r_y^2(x_k + 1) + r_y^2 + r_x^2 [(y_{k+1} - 1/2)^2 - (y_k - 1/2)^2]$$

o

(42) $$p1_{k+1} = p1_k + 2 r_y^2(x_k + 1) + r_y^2 + r_x^2 [(y_{k+1} - 1/2)^2 - (y_k - 1/2)^2]$$

donde y_{k+1} es ya sea y_k o $y_k - 1$, dependiendo del signo de $p1_k$.

Los parámetros de decisión se incrementan de la siguiente forma:

Si $p1_k < 0$, entonces $y_{k+1} = y_k$, y

$$p1_{k+1} = p1_k + 2 r_y^2(x_k + 1) + r_y^2 = p1_k + 2 r_y^2 x_{k+1} + r_y^2$$

Si $p1_k \geq 0$, entonces $y_{k+1} = y_k - 1$, y

$$p1_{k+1} = p1_k + 2 r_y^2(x_k + 1) + r_y^2 + r_x^2 [(y_k - 3/2)^2 - (y_k - 1/2)^2]$$
$$= p1_k + 2 r_y^2(x_k + 1) + r_y^2 - 2 r_x^2 y_k + 2$$
$$= p1_k + 2 r_y^2 x_{k+1} + r_y^2 - 2 r_x^2 y_{k+1}$$

Al igual que en el algoritmo de la circunferencia, los incrementos de los parámetros de decisión pueden calcularse utilizando solo adición y sustracción, dado que los valores para los términos $2 r_y^2 x$ y $2 r_x^2 y$ también pueden obtenerse en forma incremental.
En la posición inicial $(0, r_y)$ los dos términos se evalúan como

(43) $$2 r_y^2 x = 0$$

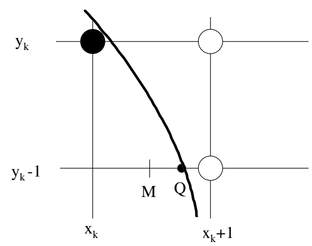
(44) $$2\, r_x^2 y = 2\, r_x^2\, r_y$$

Conforme se incrementan x y y, los valores actualizados se obtiene al sumar 2 r_y^2 a la ecuación (43) y sustraer 2 r_x^2 de la ecuación (44).
Los valores actualizados se comparan con cada paso y se mueve de la región 1 a la región 2 cuando se satisface la condición (40).

En la región 1, el valor inicial del parámetro de decisión se obtiene al evaluar la función de elipse en la posición del inicio (0, r_y):

$$\begin{aligned} p1_0 &= f_{\text{elipse}}(1, r_y - 1/2) = \\ &= r_y^2 + r_x^2 (r_y - 1/2)^2 - r_x^2 r_y^2 \\ &= r_y^2 - r_x^2 r_y + 1/4\, r_x^2 \end{aligned} \quad (45)$$

En la región 2, se realiza un muestreo en pasos unitarios en la dirección negativa de y y, el punto medio se toma entre pixeles horizontales en cada paso.



En el caso de esta región, el parámetro de decisión se evalúa como,

$$\begin{aligned} p2_k &= f_{\text{elipse}}(x_k + 1/2, y_k - 1) = \\ &= r_y^2 (x_k + 1/2)^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2 \end{aligned} \quad (46)$$

Si $p2_0 > 0$, la posición media esta fuera de la frontera de la elipse y se selecciona el pixel en la posición x_k.
Si $p2_0 \leq 0$, el punto medio esta dentro de la frontera de la elipse o sobre la misma y se selecciona la posición de pixel $x_k + 1$.

Esto otra vez se demuestra definiendo la función para un punto dentro de la elipse dado por::

$$f_{\text{dentro}}(x_d, y) = r_y^2 x_d^2 + r_x^2 y^2 - r_x^2 r_y^2 = p$$

y

$$\begin{aligned} & f_{\text{dentro}}(x_d, y) - f_{\text{elipse}}(x,y) \\ &= (r_y^2 x_d^2 + r_x^2 y^2 - r_x^2 r_y^2) - (r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2) = p - 0 \\ &= r_y^2 x_d^2 - r_y^2 x^2 = p \end{aligned}$$

dado que un punto dentro de la elipse significa que

$$r_y^2 x_d < r_y^2 x^2$$

entonces $p < 0$.

De lo contrario si el punto esta fuera de la circunferencia, $p > 0$.

Para determinar la relación entre los parámetros de decisión sucesivos en la región 2, se evalúa la función de elipse en el siguiente paso de muestreo, $y_{k+1} - 1 = y_k - 2$:

(47) $p2_{k+1} = f_{\text{elipse}}(x_{k+1} + 1/2,\ y_{k+1} - 1) =$
$= r_y^{\,2}(x_{k+1} + 1/2)^2 + r_x^{\,2}\,((y_k - 1) - 1)^2 - r_x^{\,2}\, r_y^{\,2}$

o

$$p2_{k+1} - p2_k =$$
$$= (\, r_y^{\,2}(x_{k+1} + 1/2)^2 + r_x^{\,2}\,((y_k - 1) - 1)^2 - r_x^{\,2}\, r_y^{\,2}) - (\, r_y^{\,2}(x_k + 1/2)^2 + r_x^{\,2}\,(y_k - 1)^2 - r_x^{\,2}\, r_y^{\,2})$$
$$= r_y^{\,2}\,[\,(x_{k+1} + 1/2)^2 - (x_k + 1/2)^2\,] - 2\, r_x^{\,2}(y_k - 1) + r_x^{\,2}$$

o

$$p2_{k+1} = p2_k + r_y^{\,2}\,[\,(x_{k+1} + 1/2)^2 - (x_k + 1/2)^2\,] - 2\, r_x^{\,2}(y_k - 1) + r_x^{\,2}$$

donde x_{k+1} es ya sea x_k o $x_k + 1$, dependiendo del signo de $p2_k$.

Los parámetros de decisión se incrementan de la siguiente forma:

Si $p2_k > 0$, entonces $x_{k+1} = x_k$, y

$$p2_{k+1} = p2_k - 2\, r_x^{\,2}(y_k - 1) + r_x^{\,2} = p2_k - 2\, r_x^{\,2} y_{k+1} + r_x^{\,2}$$

Si $p2_k \leq 0$, entonces $x_{k+1} = x_k + 1$, y

$$p2_{k+1} = p2_k + r_y^{\,2}\,[\,(x_k + 3/2)^2 - (x_k + 1/2)^2\,] - 2\, r_x^{\,2}(y_k - 1) + r_x^{\,2}$$
$$= p2_k + 2\, r_y^{\,2}(\,x_k + 1) - 2\, r_x^{\,2}(y_k - 1) + r_x^{\,2}$$
$$= p2_k + 2\, r_y^{\,2} x_{k+1} - 2\, r_x^{\,2}\, y_{k+1} + r_x^{\,2}$$

Cuando se captura la región 2, la posición inicial de $(x_0,\ y_0)$ se toma como la ultima posición seleccionada en la región 1 y el parámetro de decisión inicial en la región 2 es

(49) $p2_0 = f_{\text{elipse}}(x_0 + 1/2,\ y_0 - 1) = r_y^{\,2}(x_0 + 1/2)^2 + r_x^{\,2}\,(y_0 - 1)^2 - r_x^{\,2}\, r_y^{\,2}$

Con el propósito de simplificar el calculo de $p2_0$, se podría seleccionar posiciones de pixel en el sentido del reloj iniciando en $(r_x,0)$.
Así, los pasos unitarios se tomarían en la dirección positiva de y hasta la ultima posición seleccionada en la región 1.

Es posible adaptar el algoritmo de punto medio para generar una elipse en una posición no estándar al utilizar la función de elipse de la ecuación (34) y calcular las posiciones de pixel en la trayectoria elíptica entera.
De forma alternativa, se pudiera reorientar los ejes de la elipse en una posición estándar, utilizando los métodos de transformación que se verán mas adelante.

Si se supone que r_x, r_y, y el centro de la elipse se dan en coordenadas de pantalla enteras, solo se necesita cálculos incrementales en enteros con el fin de determinar valores para los parámetros de decisión en el algoritmo de punto medio para las elipses.
Los incrementos $r_x^{\,2}$, $r_y^{\,2}$, $2r_x^{\,2}$ y $2r_y^{\,2}$ se evalúan una vez en el inicio del procedimiento.

En los siguientes pasos se presenta una lista del resumen del algoritmo de la elipse de punto medio:

1. Se capturan el radio r_x, r_y y el centro de la circunferencia (x_c,y_c) y se obtiene el primer punto de una elipse centrada en el origen como
   $(x_0,y_0) = (0,r_y)$, donde $2\, r_y^2 x = 0$ y $2\, r_x^2 y = 2\, r_x^2\, r_y$
2. Se calcula el valor inicial del parámetro de decisión como
   $p1_0 = r_y^{\,2} - r_x^{\,2}\, r_y + 1/4\, r_x^{\,2}$
3. En cada posición x_k, en la región 1, iniciando en k=0, se efectúa la prueba siguiente:
   Si $p1_k < 0$, el siguiente punto a lo largo de la elipse centrada en (0,0) es

(x_k+1, y_k) y $p1_{k+1} = p1_k + 2\, r_y^{\,2} x_{k+1} + r_y^{\,2}$.

De otro modo, el siguiente punto a lo largo de la circunferencia es

$(x_k+1, y_k -1)$ y $p1_{k+1} = p1_k + 2\, r_y^{\,2} x_{k+1} + r_y^{\,2} - 2\, r_x^{\,2}\, y_{k+1}$

donde $x_{k+1} = x_k + 1$ y $y_{k+1} = y_k - 1$.

4. Se evalúa el valor inicial del parámetro de decisión en la región 2 utilizando el ultimo punto (x_0, y_0) calculado en la región 1 como:

   . $p2_0 = r_y^{\,2}(x_0 + 1/2)^2 + r_x^{\,2}(y_0 - 1)^2 - r_x^{\,2}\, r_y^{\,2}$
5. En cada posición y_k, en la región 2, iniciando en k=0, se efectúa la prueba siguiente:

   Si $p2_k > 0$, el siguiente punto a lo largo de la elipse centrada en (0,0) es

   (x_k, y_k-1) y $p2_{k+1} = p2_k - 2\, r_x^{\,2} y_{k+1} + r_x^{\,2}$.

   De otro modo, el siguiente punto a lo largo de la circunferencia es

   $(x_k+1, y_k -1)$ y $p2_{k+1} = p2_k + 2\, r_y^{\,2} x_{k+1} - 2\, r_x^{\,2}\, y_{k+1} + r_x^{\,2}$

   donde $x_{k+1} = x_k + 1$ y $y_{k+1} = y_k - 1$.
6. Se determinan puntos de simetría en los otros tres cuadrantes.
7. Se mueve cada posición de pixel calculada (x,y) a la trayectoria elíptica centrada en (x_c,y_c) y se traza los valores de las coordenadas

   $x = x + x_c$, $y = y + y_c$.
8. Se repiten los pasos para la región 1 hasta que $2\, r_y^{\,2} x \geq 2\, r_x^{\,2}\, y$.

Ejemplo

Dado el radio de una circunferencia r_x=8 y r_y=6, se demuestra el algoritmo de la elipse de punto medio al determinar las posiciones a lo largo del primer cuadrante.

Los valores iniciales del parámetro de decisión son

$2\, r_y^2 x = 0$ (con incremento $2\, r_y^2 = 72$)
$2\, r_x^2 y = 2\, r_x^2\, r_y$ (con incremento $-2\, r_x^2 = -128$)

En el caso de la región 1, el punto inicial para la elipse centrada en el origen es $(x_0,y_0) = (0,6)$, y el valor del parámetro de decisión inicial es

$$p1_0 = r_y^{\,2} - r_x^{\,2}\, r_y + 1/4\, r_x^{\,2} = 36 - 64*6 + 64/4 = 36 - 384 + 16 = -332$$

Los valores sucesivos del parámetro de decisión y las posiciones a lo largo de la trayectoria de la elipse para la región 1 (hasta $2\, r_y^{\,2} x \geq 2\, r_x^{\,2}\, y$), se calculan mediante el método del punto medio como

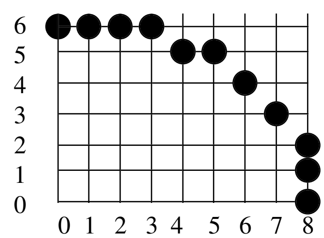
k	$p1_k$	(x_{k+1},y_{k+1})	$2\, r_y^2 x_{k+1}$	$2\, r_x^2 y_{k+1}$
0	-332	(1,6)	72	768
1	-224	(2,6)	144	768
2	-44	(3,6)	216	768
3	208	(4,5)	288	640
4	-108	(5,5)	360	640
5	288	(6,4)	432	512
6	244	(7,3)	504	384

Los valores sucesivos del parámetro de decisión y las posiciones a lo largo de la trayectoria de la elipse para la región 2 se calculan teniendo como punto inicial $(x_0,y_0) = (7,3)$, y parámetro de decisión inicial:

$$\begin{aligned} p2_0 &= r_y^{\,2}(x_0 + 1/2)^2 + r_x^{\,2}(y_0 - 1)^2 - r_x^{\,2}\, r_y^{\,2} \\ &= 36*(7+1/2)^2 + 64(3-1)^2 - 64*36 \\ &= 36*225/4 + 64*4 - 64*36 = 2025 + 256 - 2304 = -23 \end{aligned}$$

k	$p2_k$	(x_{k+1},y_{k+1})	$2\, r_y^2 x_{k+1}$	$2\, r_x^2 y_{k+1}$
0	-23	(8,2)	576	256
1	361	(8,1)	576	128
2	297	(8,0)	-	-

En la siguiente figura se muestran las posiciones generadas en el primer cuadrante:



La entrada para el siguiente procedimiento son las coordenadas para el centro y radios mayor y menor de la elipse. Se generan las posiciones a lo largo de la curva del primer cuadrante y luego se generan posiciones simétricas.

```
void PlotPoint(Display* display, Window win, GC gc, int xc, int yc, int x, int y)
{
/* draw symmetry points */
        XDrawPoint(display,win,gc,xc + x,yc + y);
        XDrawPoint(display,win,gc,xc - x,yc + y);
        XDrawPoint(display,win,gc,xc + x,yc - y);
        XDrawPoint(display,win,gc,xc - x,yc - y);
}
void EllipseMidPoint(Display* display, Window win, GC gc, int xc, int yc, int rx, int ry)
{
        int x, y, p, px, py;
        int rx1, ry2, tworx2, twory2;

        ry2 = ry*ry;
        rx2 = rx*rx;
        twory2 = 2 * ry2;
        tworx2 = 2 * rx2;
/* región 1 */
        x = 0;
        y = ry;
        PlotPoint(display,win,gc,xc,yc,x,y);
        p = round(ry2 - rx2*ry + (0.25*rx2));
        px = 0;
        py = tworx2*y;
        while (px < py) { /* se cicla hasta trazar la región 1 */
            x = x + 1;
            px = px + twory2;
            if (p < 0)
                p = p + ry2 + px;
            else {
                y = y - 1;
                py = py - tworx2;
                p = p + ry2 + px - py;
            }
            PlotPoint(display,win,gc,xc,yc,x,y);
        }
/* región 2 */
        p = round(ry2*(x+0.5)*(x+0.5) + rx2*(y-1)*(y-1) - rx2*ry2));
        px = 0;
        py = tworx2*y;
        while (y > 0)   { /* se cicla hasta trazar la región 2 */
            y = y - 1;
            py = py - tworx2;
            if (p > 0)
                p = p + rx2 - py;
            else {
                x = x + 1;
               px = px + twory2;
                p = p + rx2 - py + px;
            }
            PlotPoint(display,win,gc,xc,yc,x,y);
        }
}
```

1.4 Algoritmos de Generación de Parábolas

1.4.1 Algoritmo Básico de la Parábola

El algoritmo básico para una parábola se da por la siguiente ecuación, considerando que el origen es el punto mínimo de la parábola:

$$x = y^2$$

1.4.2 Algoritmo de Punto Medio para la Parábola

Para aplicar el método del punto medio, se define una función de parábola como:

$$f_{\text{para}}(x,y) = x - y^2$$

Cualquier punto (x,y) en la frontera de la parábola satisface la ecuación $f_{\text{para}}(x,y) = 0$.
Si el punto esta en el interior de la parábola, la función de la parábola es positiva.
Esto se demuestra definiendo la función para ese punto dentro de la parábola dado por:

$$f_{\text{dentro}}(x, y_d) = x - y_d^2 = p$$

y

$$f_{\text{dentro}}(x, y_d) - f_{\text{para}}(x,y) = (x - y_d^2) - (x - y^2) = p - 0$$
$$= y^2 - y_d^2 = p$$

Para un punto y_d , $y > 0$, y_d esta dentro del parábola, o sea
$$y^2 > y_d^2$$
entonces $p > 0$.
De lo contrario si el punto esta fuera de la circunferencia, $p < 0$.

Para resumir, la posición relativa se puede determinar al verificar el signo de la función de circunferencia:

$$f_{\text{para}}(x,y) \begin{cases} < 0, & \text{si } (x,y) \text{ esta fuera de la frontera de la parabola} \\ = 0, & \text{si } (x,y) \text{ en la frontera de la parabola} \\ > 0, & \text{si } (x,y) \text{ esta dentro de la frontera de la parabola} \end{cases}$$

Las pruebas de función de parábola de las condiciones anteriores se realizan para las posiciones medias entre los pixeles cercanos a la trayectoria de la parábola.
El parámetro de decisión en el algoritmo de punto medio se puede determinar en cálculos incrementales como se hizo en el algoritmo de la circunferencia.

Suponiendo que se acaba de trazar el pixel en (x_k, y_k), en seguida se necesita determinar si el pixel en la posición $(x_k + 1, y_k)$, o aquel en la posición $(x_k + 1, y_k + 1)$ esta mas cerca de la circunferencia.
El parámetro de decisión es la función de parábola evaluada en el punto medio entre esos dos pixeles.

$$p_k = f_{\text{para}}(x_k + 1, y_k + 1/2) = (x_k + 1) - (y_k + 1/2)^2$$

Si $p_k < 0$, el punto medio esta fuera de la circunferencia y el pixel en la línea de rastreo y_k esta mas próximo a la frontera de la circunferencia.
De otro modo, la posición media se localiza dentro de la frontera de la parábola o en esta y se selecciona el pixel en la línea de rastreo $y_k + 1$.

Los parámetros de decisión sucesivos se obtienen al utilizar cálculos incrementales.

Se obtiene una expresión recursiva para el siguiente parámetro de decisión cuando se evalúa la función de parábola en la posición de muestreo $x_{k+1} + 1 = x_k + 2$.

$$\begin{aligned} p_{k+1} &= f_{\text{para}}(x_{k+1} + 1, y_{k+1} + 1/2) = (x_{k+1} + 1) - (y_{k+1} + 1/2)^2 \\ &= [(x_k + 1)+1] - (y_{k+1} + 1/2)^2 \end{aligned}$$

$$\begin{aligned} p_{k+1} - p_k &= ((x_k + 2) - (y_{k+1} + 1/2)^2) - ((x_k + 1) - (y_k + 1/2)^2) \\ &= (x_k + 2) - (y_{k+1} + 1/2)^2 - (x_k + 1) + (y_k + 1/2)^2 \\ &= x_k + 2 - y_{k+1}^2 - y_{k+1} - ¼ - x_k - 1 + y_k^2 + y_k + ¼ \\ &= 1 - y_{k+1}^2 - y_{k+1} + y_k^2 + y_k \\ &= 1 - (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) \end{aligned}$$

$$p_{k+1} = p_k + 1 - (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$$

donde y_{k+1} es ya sea y_k o $y_k + 1$, dependiendo del signo de p_k.

Los incrementos para obtener p_{k+1} se calculan según el signo de p_k
Si p_k es negativa, $y_{k+1} = y_k$

$$\begin{aligned} p_{k+1} &= p_k - (y_k^2 - y_k^2) - (y_k - y_k) + 1 \\ &= p_k + 1 \end{aligned}$$

Si p_k es positiva, $y_{k+1} = y_k + 1$

$$\begin{aligned} p_{k+1} &= p_k + 1 - ((y_k + 1)^2 - y_k^2) - ((y_k + 1) - y_k) \\ &= p_k + 1 - (y_k^2 + 2\,y_k + 1 - y_k^2) - (y_k + 1 - y_k) \\ &= p_k + 1 - 2\,y_k - 1 - 1 \\ &= p_k - 2(\,y_k + 1) + 1 \\ &= p_k - 2y_{k+1} + 1 \end{aligned}$$

El parámetro de decisión inicial se obtiene al evaluar la función de circunferencia en la posición de inicio
$(x_0, y_0) = (0,0)$, $p_0 = f_{\text{para}}(1, 1/2) = 1 - (1/2)^2 = 1 - ¼ = 3/4$
Puesto que todos los incrementos son enteros, se puede redondear simplemente $p_0 = 1$

Se puede resumir los pasos del algoritmo de la circunferencia de punto medio como sigue:

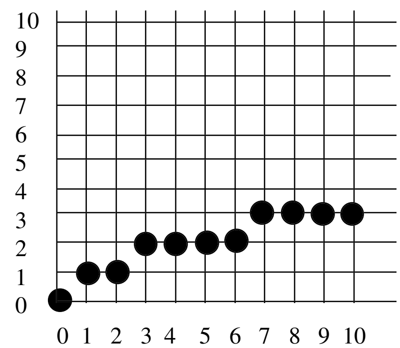
1. Se obtiene el primer punto de una parábola centrada en el origen como $(x_0,y_0) = (0,0)$.
2. Se calcula el valor inicial del parámetro de decisión como $p_0 = 3/4 = 1$.
3. En cada posición x_k, iniciando en k=0, se efectúa la prueba siguiente:
   Si $p_k \leq 0$, el siguiente punto a lo largo de la parábola centrada en (0,0) es
   (x_k+1,y_k) y $p_{k+1} = p_k +1$.
   donde $2x_{k+1} = 2x_k + 2$ y $2y_{k+1} = 2y_k$.
   Si $p_k > 0$, el siguiente punto a lo largo de la parábola es
   (x_k+1,y_k+1) y $p_{k+1} = p_k - 2y_{k+1} + 1$,
   donde $2x_{k+1} = 2x_k + 2$ y $2y_{k+1} = 2y_k + 2$.
4. Se determinan puntos de simetría en los dos cuadrantes (x,y) y $(x,-y)$.
5. Se repiten los pasos 3 y 4 mientras $x \leq 100$.

Los valores sucesivos del parámetro de decisión y las posiciones a lo largo de la trayectoria de la parábola, para los primeros 10 puntos, son los siguientes:

(x,y) analítico	k	p_k	(x_{k+1},y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
(1,1)	0	1	(1,1)	2	2
(2,1.414)	1	0	(2,1)	4	2
(3,1.732)	2	1	(3,2)	6	4
(4,2)	3	-2	(4,2)	8	4

(5,2.236)	4	-1	(5,2)	10	4
(6,2.449)	5	0	(6,2)	12	4
(7,2.646)	6	1	(7,3)	14	6
(8,2.828)	7	-4	(8,3)	16	6
(9,3)	8	-3	(9,3)	18	6
(10,3.162)	9	-2	(10,3)	20	6

En la siguiente figura se muestran las posiciones generadas en el primer cuadrante:



El siguiente procedimiento despliega una parábola de rastreo utilizando el algoritmo de punto medio.

```
void PlotPoint(Display* display, Window win, GC gc, int x, int y)
{
/* draw symmetry points */
        XDrawPoint(display,win,gc,x,y);
        XDrawPoint(display,win,gc,x,- y);
}
void ParabolaMidPoint(Display* display, Window win, GC gc)
{
        int x, y, p;

        x = 0;
        y = 0;
        p = 1;
        PlotPoint(display,win,gc,x,y);
/* se cicla hasta trazar todo un octante */
        while (x <= 100)
        {
            x = x + 1;
            if (p > 0) {
                y = y + 1;
                p = p - 2*y + 1;
            }
            else
                p = p + 1;
            PlotPoint(display,win,gc,x,y);
        }
}
```