

Algoritmos de las Primitivas Gráficas

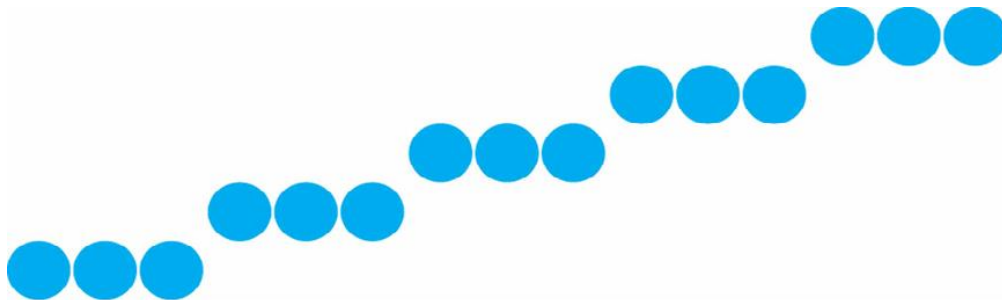
Prof. Wílmer Pereira

<http://www ldc usb ve/~wpereira>

Algoritmos de líneas

Inicialmente para rasterizar una línea se dan los dos extremos y la librería OpenGL se encarga de pintar ... sin embargo hay redondeos como, por ejemplo, la posición (10.48 , 20.51) se convierte a (10 , 21)

- Las posiciones discretas de los píxeles dan origen a líneas que realmente se dibujan escalonadas:



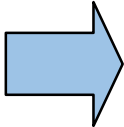
- Mientras más alta la resolución, menor será el efecto de escalonado
- Además hay efectos de suavizado utilizando píxeles con colores tenues junto a los puntos fuertemente coloreados de la línea

Ecuaciones para líneas

- La opción obvia es utilizar la ecuación de la recta clásica:

$$y = mx + b$$

Donde:

m		Es la pendiente de la recta
b		Es el punto de corte en el eje y

- Dado los dos puntos del segmento (x_0, y_0) y (x_1, y_1) :

$$m = \frac{\Delta y}{\Delta x} = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

$$b = y_0 - mx_0$$

- Usando la ecuación de la pendiente, es claro que:

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

$$\Delta x = \frac{\Delta y}{m}$$

Algoritmo DDA

La idea es usar un algoritmo con aproximaciones sucesivas, con un algoritmo iterativo, basado en el cálculo de Δy y Δx

- Partiendo de una pendiente positiva menor a 1, la x se incrementa siempre una unidad ... en cambio ...

$$y_{k+1} = y_k + 1 \text{ o } y_{k+1} = y_k$$

Si $m < 1$ entonces $y_{k+1} = \text{round}(y_k + m)$ y $x_{k+1} = x_k + 1$

- Por otro lado si la pendiente es positiva pero mayor a 1, sería:

$$x_{k+1} = x_k + 1 \text{ o } x_{k+1} = x_k$$

Si $m > 1$ entonces $x_{k+1} = \text{round}(x_k + \frac{1}{m})$ y $y_{k+1} = y_k + 1$

- Por último si la pendiente es negativa entonces:

Si $m < -1$ entonces $y_{k+1} = y_k - m$ y $x_{k+1} = x_k + 1$

Si $m > -1$ entonces $x_{k+1} = x_k - \frac{1}{m}$ y $y_{k+1} = y_k + 1$

Algoritmo DDA

- Todas las operaciones que involucran el cálculo de la pendiente deben redondearse.
- DDA es más rápido que aplicar directamente la ecuación de la recta porque se evita una multiplicación

... pero ...

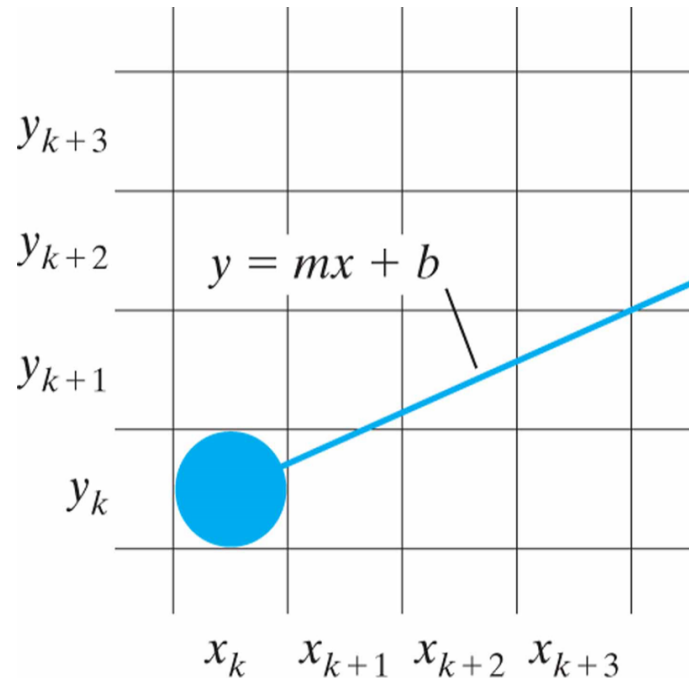
- La acumulación de errores de redondeo puede no dar la mejor posición para el siguiente pixel

... además...

- El cálculo de una operación $1/m$ requiere del procesador punto flotante que no es parte CPU. Más aún debe pasar por el bus que es un recurso crítico cuando los procesos son de alta demanda de computo.

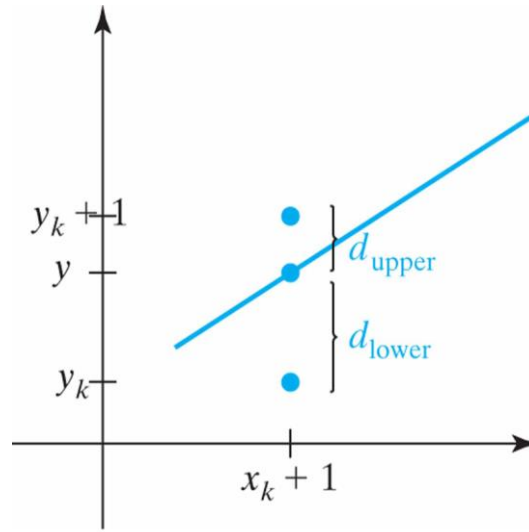
Algoritmo de Bresenham

Este algoritmo es el más eficiente pues sólo involucra cálculos con aritmética entera y además se puede adaptar a círculos y curvas en general



- Al igual que DDA trabaja la rasterización dependiendo de la pendiente de la recta. Sin embargo la diferencia es considerar la distancia en la cual x_{k+1} se encuentra del pixel y_{k+1} o de y_k

Cálculo de distancias



- Para calcular la coordenada y debe evaluarse la ecuación de la recta $y = m(x_k + 1) + b$ donde:

$$d_{lower} = y - y_k = m(x_k + 1) + b - y_k$$
$$d_{upper} = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$$

- Para determinar cual pixel pintar (y_{k+1} o y_k) se calcula la diferencia entre d_{lower} y d_{upper} :

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

Rasterización en Bresenham

- Si la diferencia es negativa, se pinta $(x_k + 1, y_k)$. En caso contrario se pinta $(x_k + 1, y_k + 1)$.
- Dado que $m = \Delta y / \Delta x$ y es deseable eliminar la división (evitar el procesador de punto flotante) entonces:

$$p_k = \Delta x(d_{lower} - d_{upper}) = 2\Delta y.x_k - 2\Delta x.y_k + c$$

Donde c es una constante que se calcula sólo una vez:

$$c = 2\Delta y + \Delta x(2b - 1)$$

- Para hacer los cálculos incrementales y simplificar los cálculos:

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

Haciendo:

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

Como $x_{k+1} = x_k + 1$, entonces:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \quad \text{donde} \quad p_0 = 2\Delta y - \Delta x$$

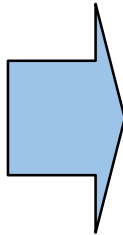
Finalmente para Bresenham ...

- Funciona para líneas de cualquier pendiente y los cálculos son análogos para la simetría de octantes.
- Cuando $d_{lower} = d_{upper}$ es indiferente cual pixel pintar ...
- Cuando

$$\Delta y = 0$$

$$\Delta x = 0$$

$$\Delta y = \Delta x$$



Recta horizontal

Recta vertical

Línea diagonal

Se carga directamente en el *buffer* de visualización, sin ningún procesamiento, para ser directamente rasterizadas

Paralelizar

Si se disponen de varios procesadores podríamos paralelizar por líneas o asociar procesadores a sectores de píxeles

- Suponiendo que hay n_p procesadores y una recta en el primer octante $m < 1$. Se puede calcular el segmento de línea para cada procesador:

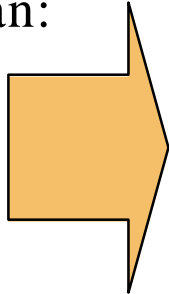
$$\Delta x_p = \frac{\Delta x + n_p - 1}{n_p}$$

Así el primer valor de la k -ésima posición sería:

$$x_k = x_0 + k\Delta x_p$$

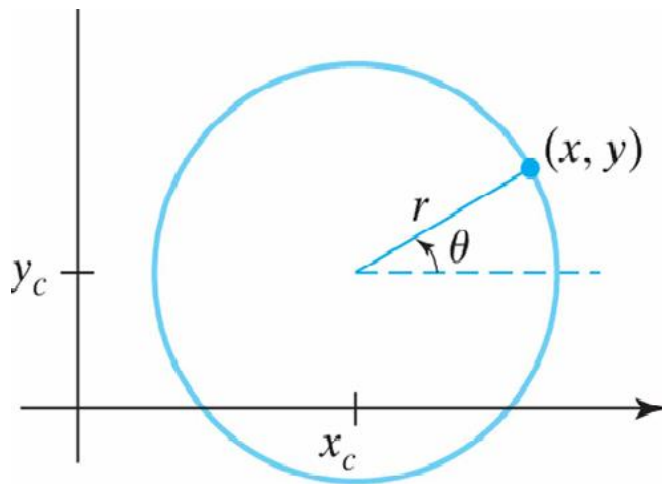
- Por ejemplo para :

- Si $n_p = 4$ y $\Delta x = 15$ entonces $\Delta x_p = 4$
- Así que los extremos serían:


$$\begin{aligned} &x_0 \\ &x_0 + 4 \\ &x_0 + 8 \\ &x_0 + 12 \\ &x_0 + 15 \end{aligned}$$

Algoritmos eficientes de curvas

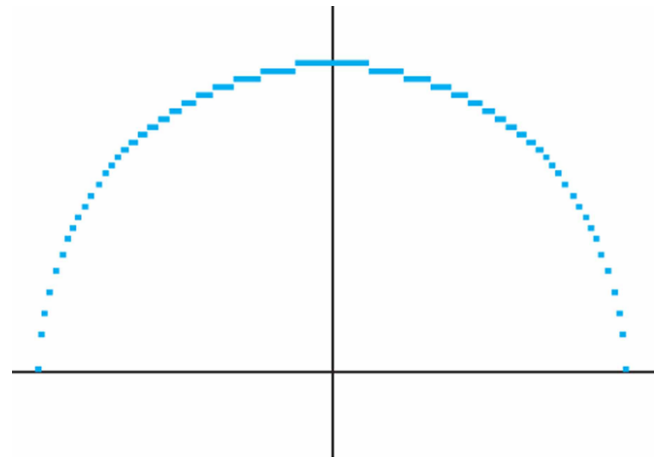
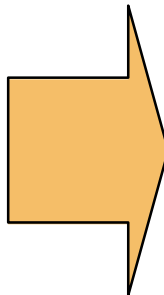
Además de poder aproximar con segmentos de rectas, es deseable construir curvas con figuras de ecuaciones bien conocidas como círculos, elipses, parábolas, hipérbolas, ...



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

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

... pero ...



Círculo con Ecuaciones Trigonómicas

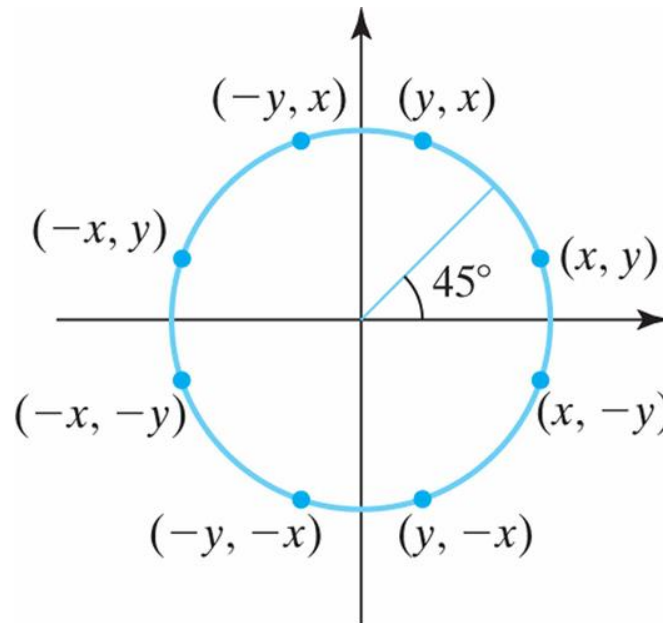
- Una manera de eliminar el espaciado es usar ecuaciones en coordenadas polares en forma normal paramétrica:

$$x = x_c + r \cos\theta$$

$$y = y_c + r \sin\theta$$

... pero ... tiene cálculos trigonométricos ...

- Sin embargo, esto no impide aprovechar la simetría de los octantes para optimizar los cálculos ...



Círculo con Bresenham

La idea es evitar los cálculos en los reales (raíz cuadrada y operaciones trigonométricas), buscando el pixel a pintar más cercano al círculo, paso a paso.

- El objetivo es evaluar si el punto medio entre dos pixeles está dentro, fuera o sobre la ecuación del círculo ...

- Dada la ecuación del círculo
cualquier (x, y) sobre el círculo

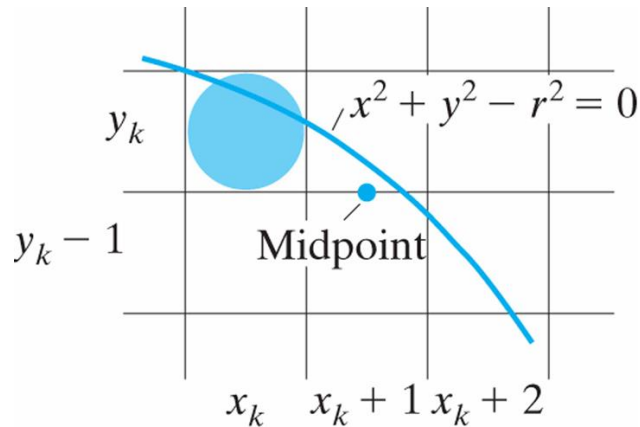


$$f_{cir}(x, y) = x^2 + y^2 - r^2$$
$$f_{cir}(x, y) = 0$$

$$f_{cir}(x, y) = \begin{cases} < 0 & \text{Si } x \text{ está dentro del borde del círculo} \\ = 0 & \text{Si } x \text{ está sobre el borde del círculo} \\ > 0 & \text{Si } x \text{ está fuera del borde del círculo} \end{cases}$$

**En consecuencia, el signo de la función círculo
determinará cual pixel se pintará**

Cálculo del punto medio



- Si el pixel ya pintado es (x_k, y_k) , es necesario saber si se pintará

$$(x_k + 1, y_k)$$
$$(x_k + 1, y_k - 1)$$

- Lo necesario es evaluar la fórmula en el punto medio

$$p_k = f_{cir}(x_k + 1, y_k - 1/2)$$
$$= (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$$

$$p_k = \begin{cases} < 0 & \text{Pintar } y_k \\ = 0 & \text{Indiferente} \\ > 0 & \text{Pintar } y_{k-1} \end{cases}$$

Cálculo del punto medio iterativo

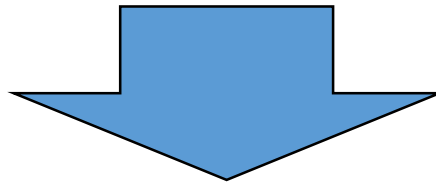
- ... y expresarla iterativamente ...

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

donde y_{k+1} será y_k o y_{k-1} , dependiendo del signo de p_{k+1}

- El término inicial será

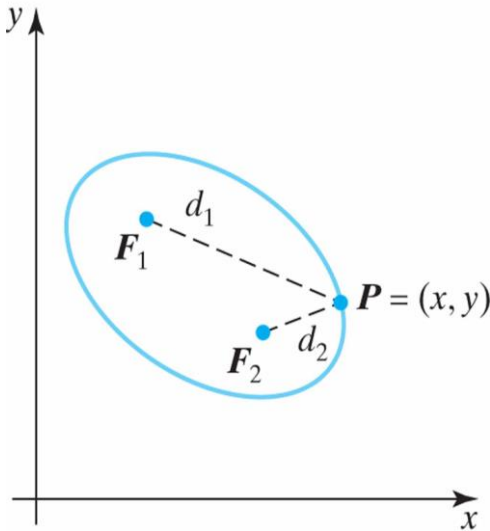
$$\begin{aligned} p_0 &= f_{cir}(1, r - 1/2) \\ &= 1 + (r - 1/2)^2 - r^2 \\ &= 5/4 - r \text{ lo que puede simplificarse en } 1 - r \end{aligned}$$



**Finalmente sólo sumas, resta y productos, lo cual,
hace este algoritmo muy eficiente**

Elipses

Al igual que con el círculo, podemos adaptar el algoritmo de Bresenham para rasterización de elipses



Dado los focos \mathbf{F}_1 y \mathbf{F}_2 , para todo punto de una elipse:

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

Expresado en coordenadas de los focos $\mathbf{F}_1 = (x_1, y_1)$ y $\mathbf{F}_2 = (x_2, y_2)$, la ecuación se reescribe como:

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

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

Esta es la ecuación general de cualquier cónica y en particular la de una elipse que se orienta de cualquier manera con respecto a los ejes coordenados

... sin embargo, tiene muchas operaciones punto flotante ...

Elipse orientada con los ejes

La ecuación se simplifica si se alinea la elipse con los ejes coordenados:

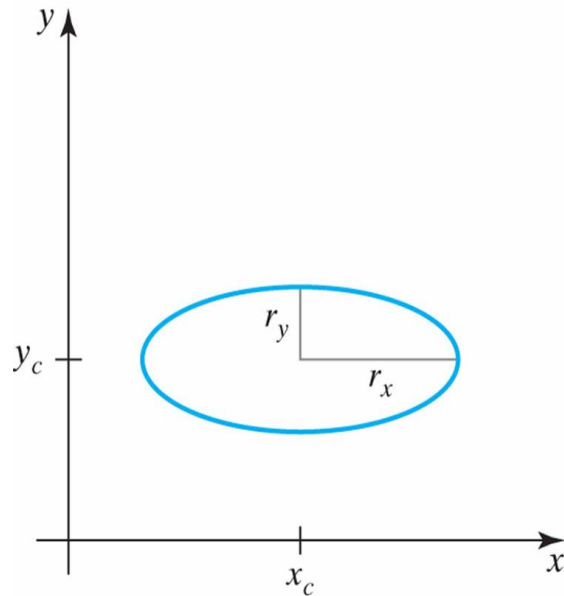
$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

Con coordenadas polares en forma paramétricas sería:

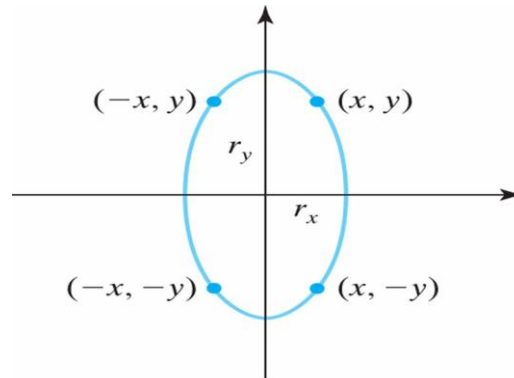
$$x = x_c + r_x \cos\theta$$

$$y = y_c + r_y \sen\theta$$

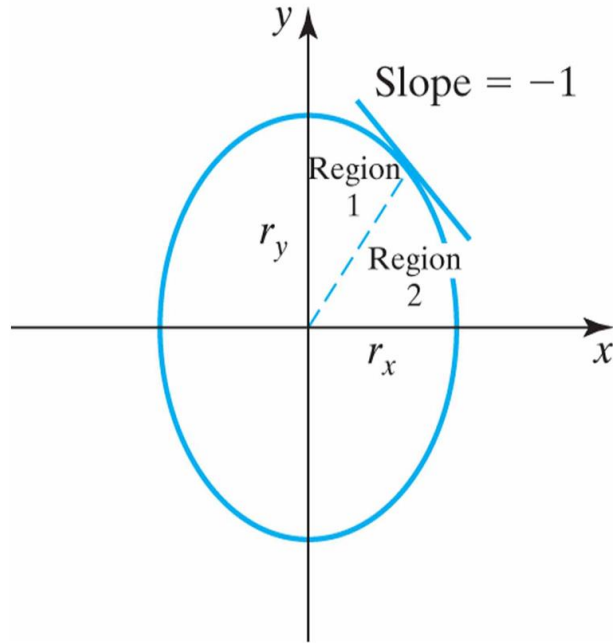
Pero también implica acceder al procesador punto flotante



● A resaltar que hay simetría de cuadrantes más no de octantes en la elipse



Algoritmo de punto medio para la elipse



El algoritmo de rasterización de la elipse es similar al del círculo. Se aplica, al primer cuadrante, en dos partes:

- Si la pendiente de la curva es menor que -1 , x toma pasos unitarios (región 1).
 - Si la pendiente es mayor que -1 , es y la coordenada que toma pasos unitarios (región 2).
- El barrido puede ser en el sentido de las agujas del reloj, comenzando en $(0, r_y)$. Mientras la pendiente de la curva sea mayor que -1 serán pasos unitarios de y . Al ser la pendiente menor a -1 se pasa a pasos unitarios de x .

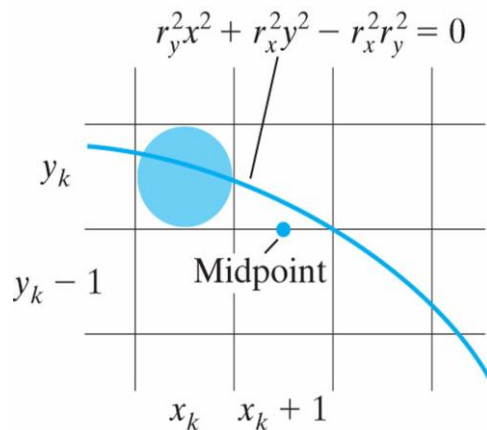
$$f_{\text{elipse}}(x, y) = \begin{cases} < 0 & (x, y) \text{ dentro de la elipse} \\ = 0 & (x, y) \text{ sobre la elipse} \\ > 0 & (x, y) \text{ fuera de la elipse} \end{cases}$$

Cálculos de Bresenham para elipses

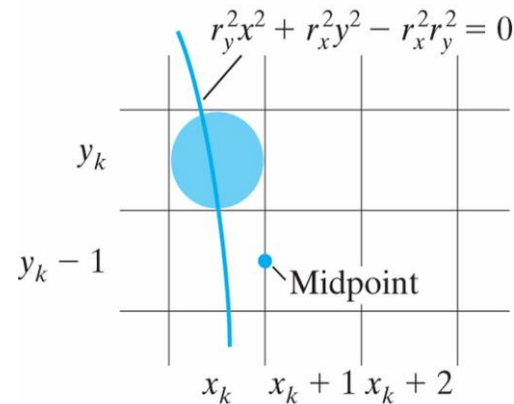
La pendiente de la elipse se calcula como:

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

donde en el borde de las regiones $dx/dy = -1$



Región 1



Región 2

En la región 1, dependiendo del signo de $p1_k \dots$

$$\begin{aligned} p1_k &= f_{ellipse}(x_k + 1, y_k - 1/2) \\ &= (r_y)^2(x_k + 1)^2 + (r_x)^2(y_k - 1/2)^2 - (r_x)^2(r_y)^2 \end{aligned}$$

Cálculos de Bresenham para elipses

● Haciendo el proceso iterativo ...

$$\begin{aligned} p1_{k+1} &= f_{ellipse}(x_{k+1} + 1, y_{k+1} - 1/2) \\ &= (r_y)^2((x_k + 1) + 1)^2 + (r_x)^2(y_{k+1} - 1/2)^2 - (r_x)^2(r_y)^2 \\ &= p1_k + 2(r_y)^2(x_k + 1) + (r_y)^2 + (r_x)^2((y_{k+1} - 1/2)^2 - (y_k - 1/2)^2) \end{aligned}$$

Donde y_{k+1} será o y_k o y_{k-1} dependiendo del signo de $p1_k$

● Para la región 1, el valor inicial de la iteración será en $(x_0, y_0) = (0, r_y)$:

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

● Los cálculos para la región 2, son análogos partiendo de:

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

...

$$\begin{aligned} p2_{k+1} &= p2_k - 2(r_x)^2(y_k - 1) + (r_x)^2 + (r_y)^2((x_{k+1} + 1/2)^2 - (x_k + 1/2)^2) \\ p2_0 &= f_{ellipse}(x_0 + 1/2, y_0 - 1) = (r_y)^2(x_0 + 1/2)^2 + (r_x)^2(y_0 - 1)^2 - (r_x)^2(r_y)^2 \end{aligned}$$

Consideraciones finales para elipses

- Es necesario considerar la simetría de cuadrante para pintar el resto de la elipse.
- Si se desea que la elipse no esté orientada con los ejes coordenados, debe aplicar una transformación de rotación.
- Hay términos que se pueden calcular solamente una vez como:

$$(r_x)^2, 2(r_x)^2 \dots$$

Otras curvas ...

Existen las funciones exponenciales, trigonométricas, polinomios y *spline* que se podrían generar a partir de círculos, elipses o segmentos de líneas

**Sin embargo también están las cónicas
que son funciones cuadráticas**

- Una sección cónica es una ecuación de segundo grado

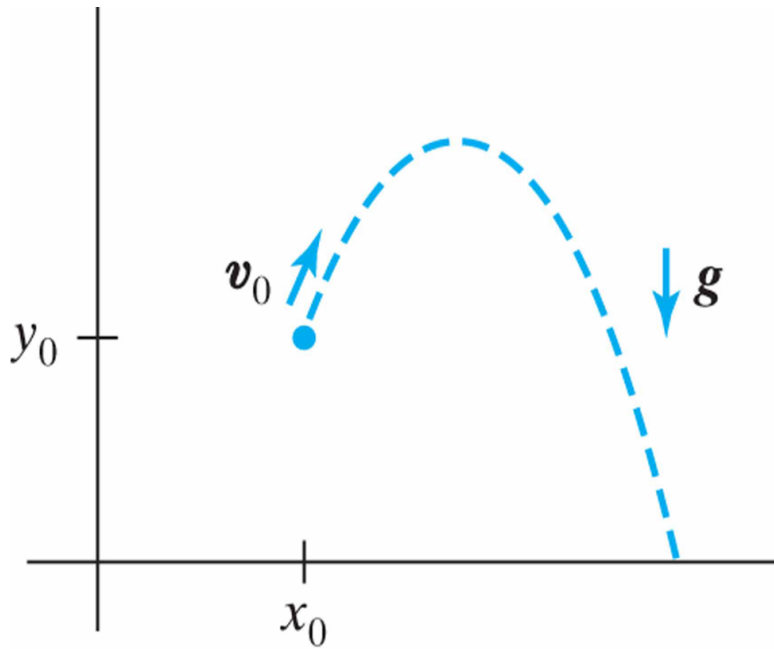
$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

donde A , B , C , D , E y F determinan el tipo de curva según el discriminante $B^2 - 4AC$

$$B^2 - 4AC = \begin{cases} < 0 & \text{Elipse o círculo} \\ = 0 & \text{Parábola} \\ > 0 & \text{Hipérbola} \end{cases}$$

- Elipses, parábolas e hipérbolas son particularmente útiles en animaciones de movimientos sujetos a la gravedad, ondas electromagnéticas o fuerzas nucleares

Movimiento terrestre



Una trayectoria parabólica convencional se modela como:

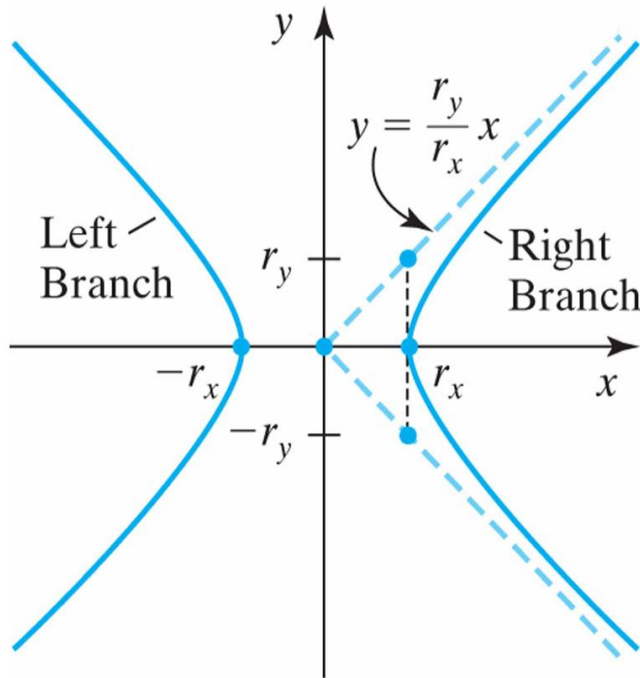
$$y = y_0 + a(x - x_0)^2 + b(x - x_0)$$

donde a y b representan la velocidad inicial V_0 y g es la gravedad.

Entonces la ecuación paramétrica del movimiento parabólico es:

$$\begin{aligned} x &= x_0 + v_{x0}t \\ y &= y_0 + v_{y0}t - \frac{1}{2}gt^2 \end{aligned}$$

Trayectorias de escape ...



La hipérbola es útil en aplicaciones científicas para colisiones de partículas cargadas o trayectorias de escape de meteoritos y cometas

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

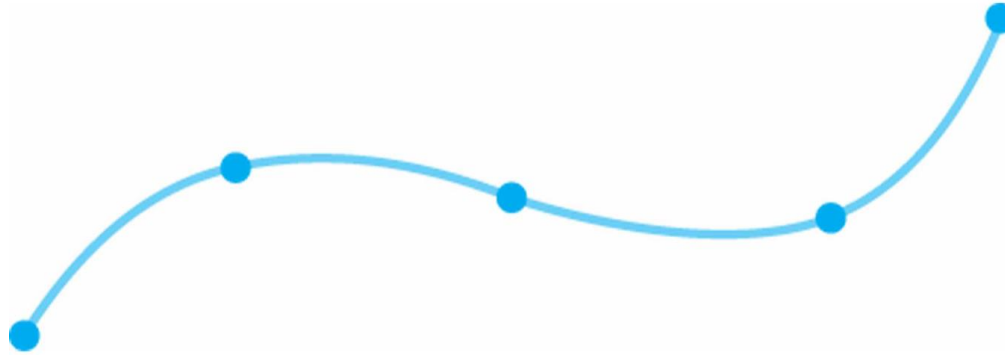
donde

$x \leq -r_x$ es la rama izquierda

$x \geq r_x$ es la rama derecha

Como la parábola y la hipérbola tienen ejes de simetría, se puede aplicar la técnica de punto medio (Bresenham) para rasterizarlas

Interpolación con polinomios y spline



- Interpolating a curve with polynomial functions is practical in many cases. The lower the degree of the polynomial “smoother” the curve ...

- A polynomial of degree n is defined as:

$$\begin{aligned} y &= \sum_{k=0}^n a_k x_k \\ &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_n x^n \end{aligned}$$

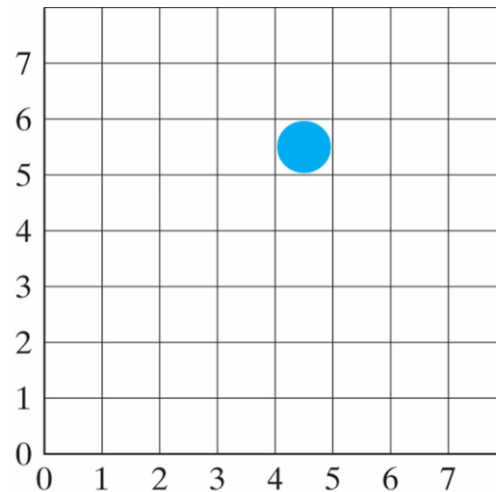
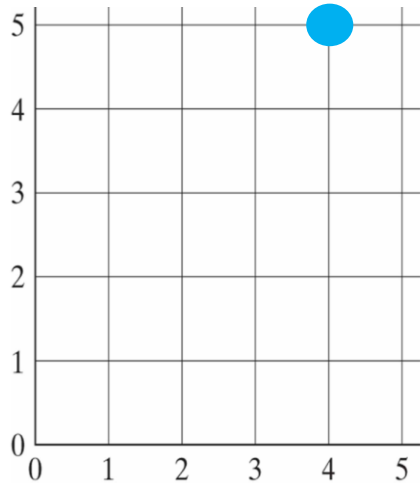
- *Splines* are constructed with sections of quadratic or cubic polynomials between each pair of points, ensuring continuity. Additionally, it must be imposed that the slope at the extremes of each polynomial is equal so that the curve is smooth.

... Finalmente la paralelización es posible aprovechando las simetrías ...

Mantenimiento de la geometría

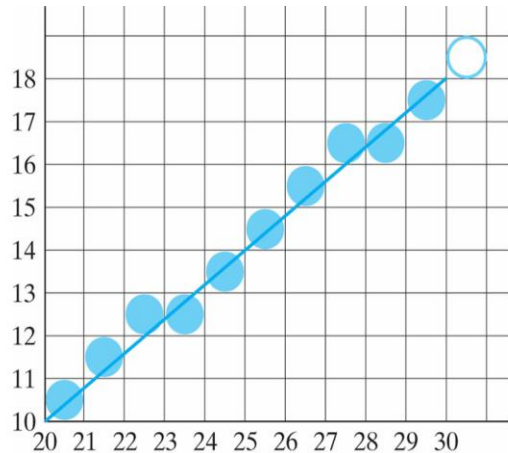
Dado que los píxeles son puntos discretos, es importante que las proporciones y la forma del objeto se preserven al ser rasterizado

- En primer lugar se debe definir como se visualizan los píxeles en el sistema de coordenadas

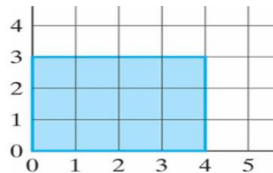


- Esto tendrá un impacto en como se visualizan las líneas en la pantalla

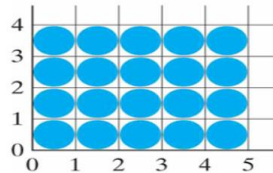
Ejemplo de líneas y rellenos pixelados



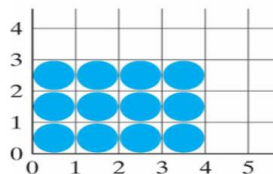
La siguiente línea se muestra en coordenadas (20 , 10) y (30 , 18). Si se pinta el sombreado la línea tendría 11 unidades horizontales y 9 unidades verticales, cuando debería tener 10 y 8 respectivamente.



(a)



(b)

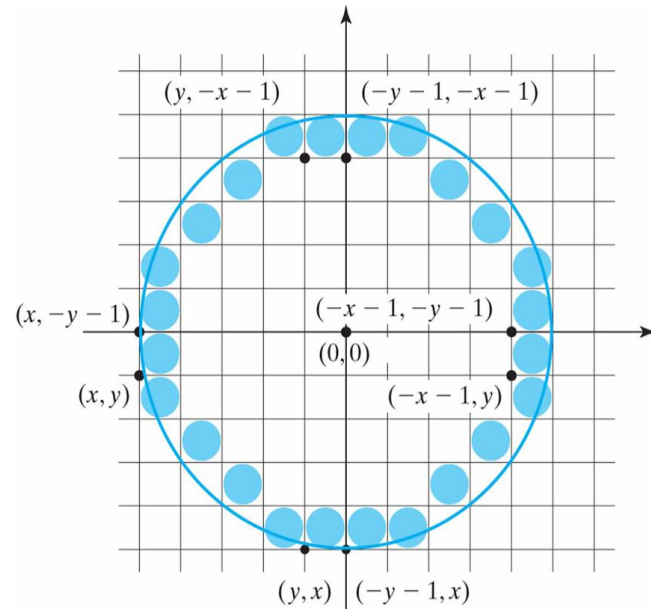
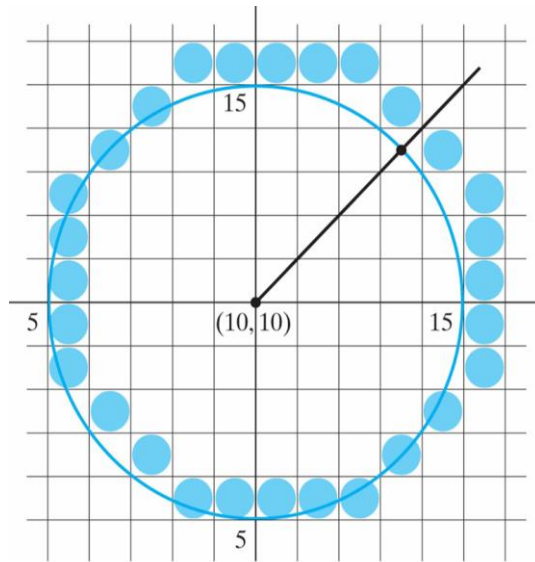


(c)

En el caso de rellenos, estos deberían incluir sólo píxeles que estén en el interior de la superficie ...

Borde de curvas

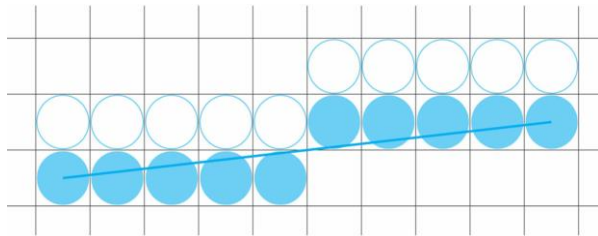
- El mismo problema se presenta al pintar bordes de curvas. Suponga ambos círculos con radio 5.



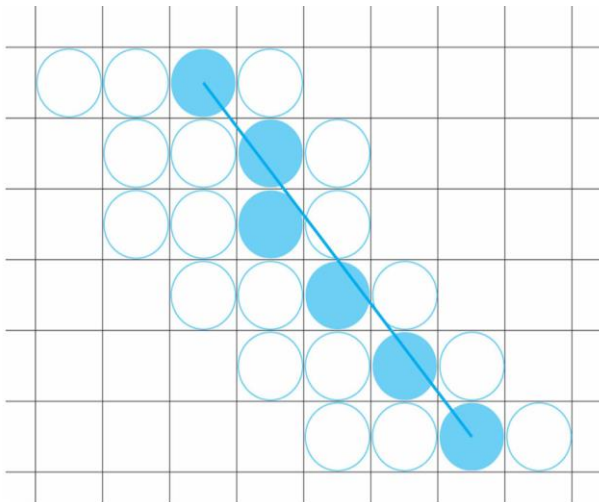
- El segundo tiene el diámetro en 10 unidades, el cual es el correcto. En cambio, el primero por no respetar la regla de sólo pintar puntos interiores, tiene un diámetro de 11 unidades ...

Ancho de líneas

Esta característica depende de la pendiente del segmento de recta ya que para $m < 1$ la expansión debe ser vertical. En cambio para $m > 1$ la expansión debe ser horizontal



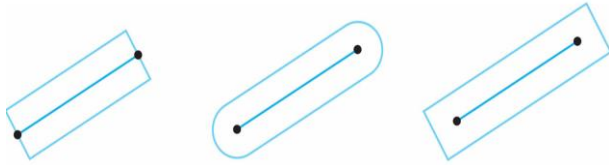
$m < 1$



$m > 1$

Cuando la pendiente es $m=1$ la línea se verá más delgada en un factor $\frac{1}{\sqrt{2}}$

Terminación de líneas



El último caso permite holgura al concatenar otra superficie o línea

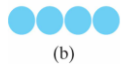
- Cuando hay acople de líneas se debe ser cuidadoso mientras más gruesa sea la línea



La primera terminación no es muy conveniente si el ángulo que forman las líneas es muy pequeño

Patrón de líneas

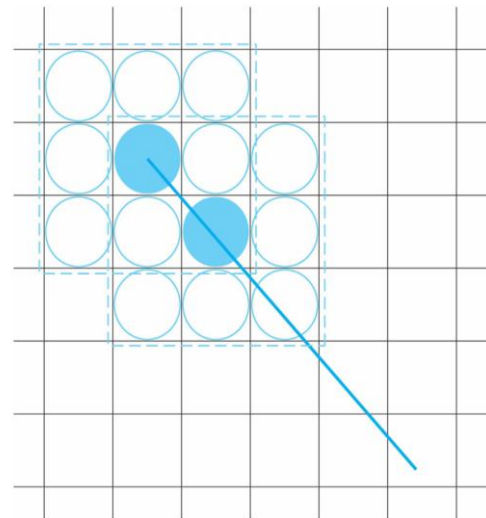
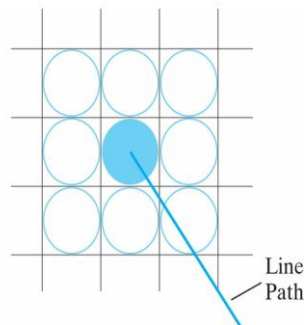
El estilo debe ser cuidadosamente diseñado pues es sensible a la pendiente del segmento de recta o la curva



Aún sin patrón la longitud de una línea horizontal con respecto a la diagonal a 45° se aprecia más grande en un factor $\sqrt{2}$

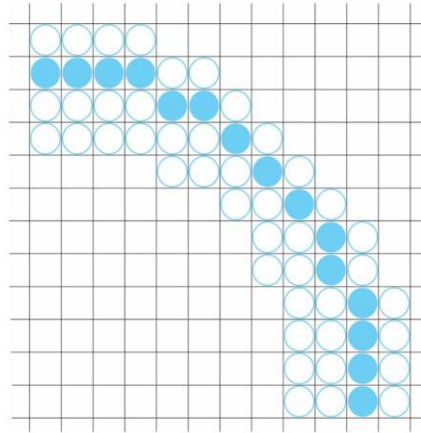
- Cuando se utiliza una brocha o trazado particular, debe haber un acople puede casi siempre implica superposición

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

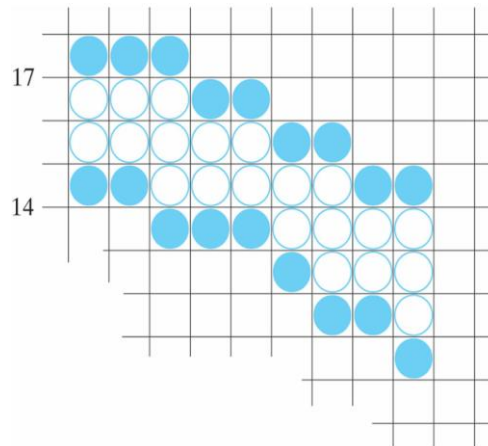


Grosor en el perímetro de una curva

- Para un arco de círculo de ancho 4, el cálculo va depender de la simetría de octante, es decir, de la pendiente ...

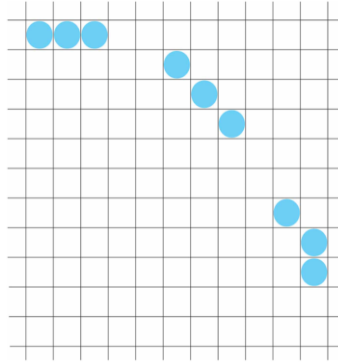


- Otro método es definir dos curvas paralelas cuya separación sea el ancho que se desea

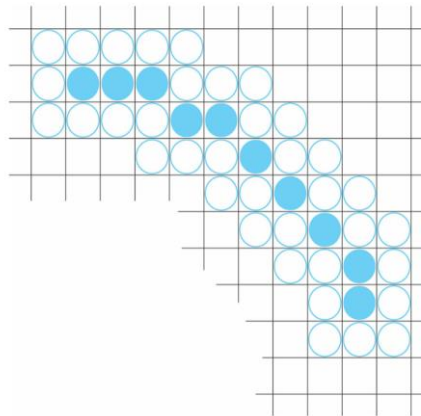


Curva con patrón de línea o brocha

- Para líneas con trazos y espacio, es importante respetar la separación entre los trazos, lo cual, depende de la pendiente

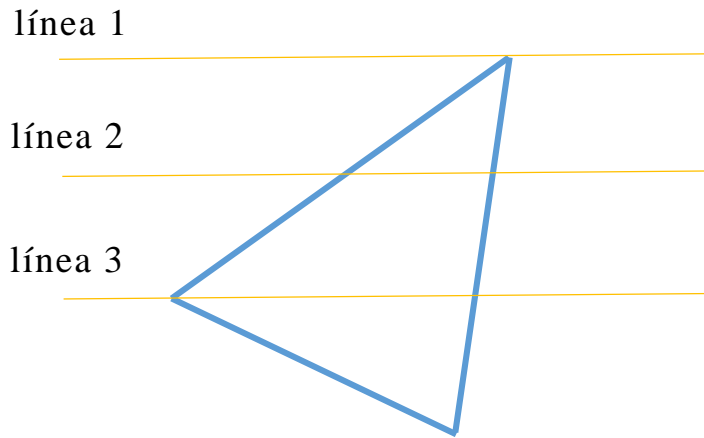


- En este caso es un patrón de brocha rectangular



Relleno de polígonos

En general lo conveniente es tener polígonos convexos y tener cuidado en los vértices si se usa el algoritmo par/impar

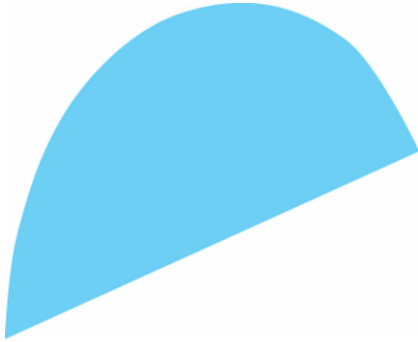


La línea 2 sigue la estrategia pintar cada par ... La línea 1 cuenta doble ... ¿Y la línea 3?

- La solución está en verificar si la línea de barrido deja los dos segmento del mismo lado o de lados diferentes ... en ese caso la línea 3 deja cada arco de lados diferentes de la línea de barrido

Relleno de superficies

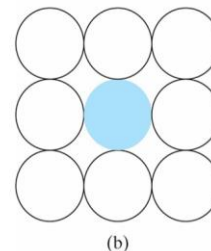
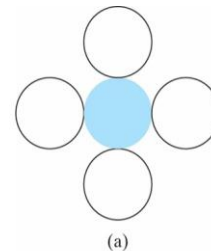
Cuando se rellenan superficies delimitadas por curvas, lo importante es saber si la curva es una cónica o con bordes irregulares



En este caso podemos aplicar el mismo método de los polígonos convexos. Sin embargo hay que considerar es una combinación del procedimiento de líneas con el procedimiento de curvas

- En cambio cuando los bordes son irregulares, lo ideal es pintar a partir de un punto interior y rellenar hasta el perímetro ...

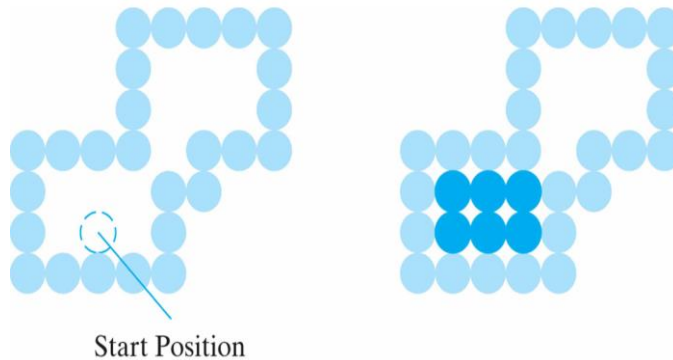
Es posible rellenar con un color por defecto o un patrón y las dos técnicas básicas son:
4-conectados u 8-conectados



Algoritmo recursivo de relleno

```
void boundaryFill4(int x, int y, int fillColor, int borderColor) {  
    int interiorColor;  
    getPixel(x, y, interiorColor);  
    if ((interiorColor!=borderColor)&&(interiorColor!=fillColor))  
    {  
        setPixel(x, y); // Set color of pixel to fillColor.  
        boundaryFill4(x + 1, y , fillColor, borderColor);  
        boundaryFill4(x - 1, y , fillColor, borderColor);  
        boundaryFill4(x , y + 1, fillColor, borderColor);  
        boundaryFill4(x , y - 1, fillColor, borderColor)  
    }  
}
```

- Además de que la recursión puede hacer el algoritmo ineficiente, 4-conectados tiene un problema particular...



Algoritmos de estilo de llenado

En conclusión el llenado de superficies es con líneas de barrido o puntos interiores que crecen hasta tocar los bordes

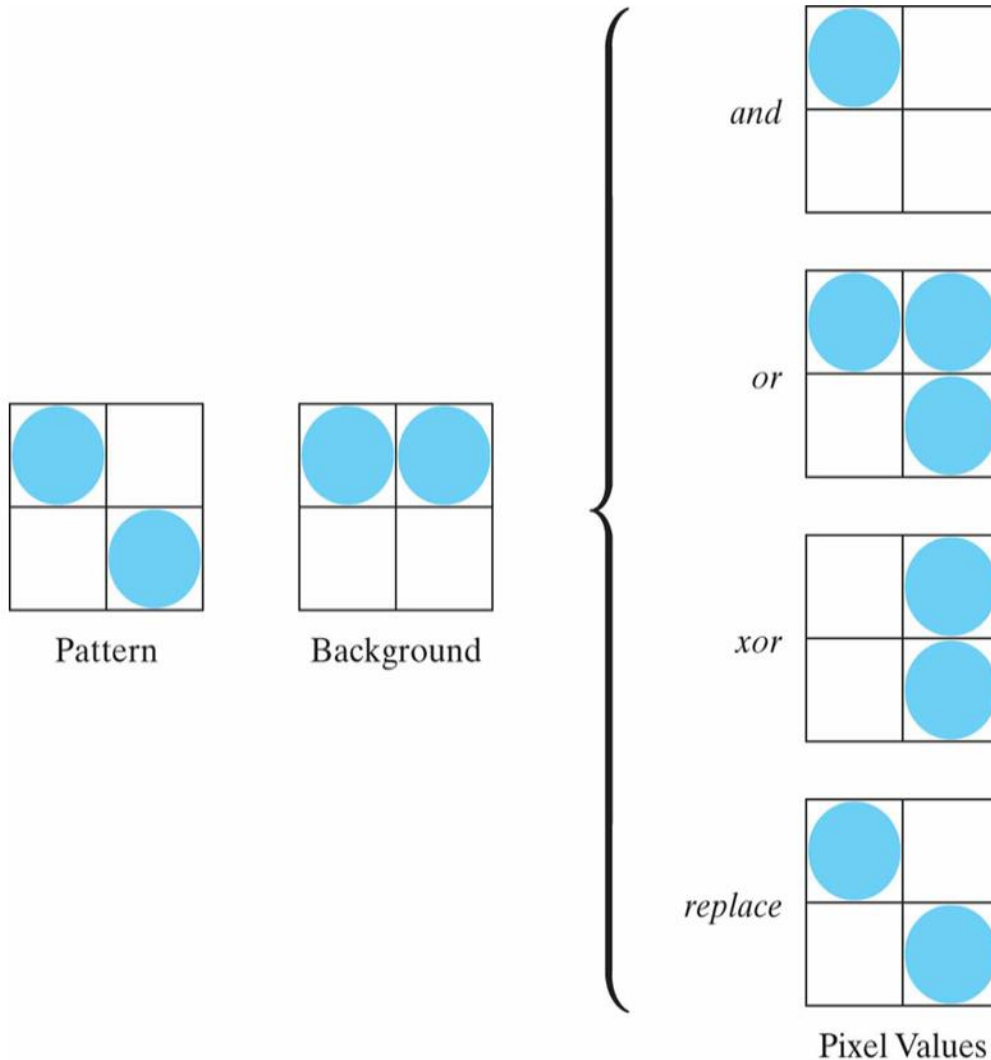
- Si se trata de un patrón y además hay transparencia en los objetos frontales hay varias consideraciones a tomar en cuenta
- Si el relleno es con patrones, debe partir un punto interno, preferiblemente sobre el borde. Además debe ser relativo a ese punto de la superficie, en caso de que los bordes se muevan.
- Cuando hay transparencia, se puede hacer manualmente, prescindiendo de `GL_BLEND`, ... las condiciones serían:

$$\mathbf{P} = t\mathbf{F} + (1-t)\mathbf{B}$$

donde \mathbf{F} , \mathbf{B} y \mathbf{P} son los colores de los colores frontales, posterior y resultante
 t es el factor de transparencia del color frontal

Esto aplica para cada color

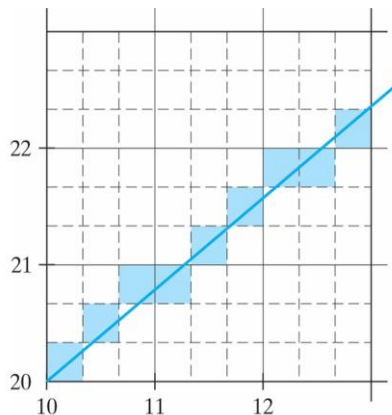
Relleno de superficies con patrones



Suavizado

Los efectos ocurren cuando hay submuestreo, es decir, no se captura suficiente información por lo que se distorsionan las imágenes

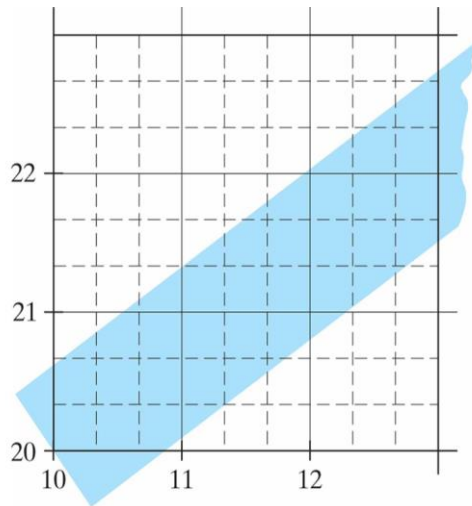
- Al transformar una señal de analógica a digital, el teorema de Nyquist exige muestrear, al menos, al doble de la frecuencia de la señal portadora
- En el caso de la rasterización una opción es aumentar la resolución de la pantalla ... pero ... no siempre es posible ... por lo tanto, una de las soluciones más populares es modificar la intensidad de los píxeles haciendo **supermuestreo**



Como sólo es posible tocar tres subpíxeles (porque se seleccionan según los calculos con Bresenham), se pintará con tres niveles de intensidad ...

Supermuestreo

- Si se desean más intensidades se aumenta el tamaño de la matriz de supermuestreo (4x4 tendríamos 4 intensidades posibles)



En caso de una línea con cierto ancho, las intensidades serían proporcionales a la cantidad de subpíxeles que se tocan ... En este caso el máximo serían 9 posibles valores de intensidad

- Al pintar se puede considerar el valor del fondo. Por ejemplo si la línea ancha toca n subpíxeles, quedarían m subpíxeles de color de fondo. Para este supermuestreo se debe cumplir que:

$$n+m = 9$$

$$Pixel_{Color} = \frac{n.ColorFrontal + m.ColorFondo}{9}$$

Máscara de subpíxeles

1	2	1
2	4	2
1	2	1

El color del pixel lo determina los subpíxeles que toque la línea que se esté intentando rasterizar ...

- El cálculo de la intensidad sería un promedio de la contribución de cada subpixel. Por ejemplo el subpixel central contribuye con $1/4$ de la intensidad. Los laterales $1/8$ y los esquineros $1/16$...
- También se puede tomar en cuenta la intensidad de los subpíxeles de los píxeles vecinos para promediar y hacer más suave la variación de intensidad entre los píxeles ...

Otras tres técnicas de suavizado

① Se calcula el área que cubre la línea con los subpíxeles. Si por ejemplo se cubre 90% del área del pixel entonces se pinta con una intensidad del 90% de la máxima intensidad. Esto es suponiendo cierto ancho en la línea.

② Este método es similar a la máscara de subpíxeles pero es con una función continua de cubrimiento del pixel

③ Dado que la diagonal a 45° es más grande en un factor $\sqrt{2}$, se usa mayor intensidad para compensar apariencia de menor intensidad por la separación entre los píxeles. Las líneas horizontales y verticales se pintarían a menor intensidad y el resto entre esos dos valores ...

