UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

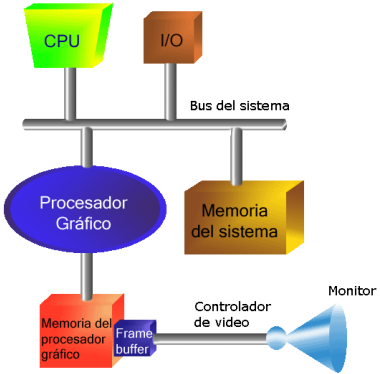
DEPARTAMENTO
DE INFORMÁTICA

Computación Gráfica – Algoritmos de Raster

Semestre 2022-1

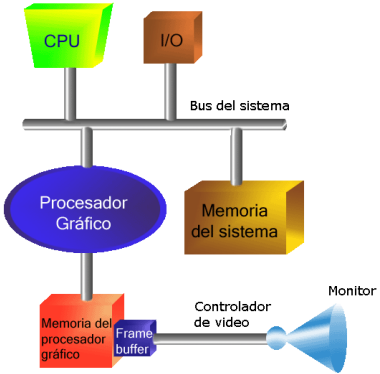
Sistema gráfico de barrido (raster)
ARQUITECTURA

- CPU y memoria principal usadas por todos los programas
- Se necesita un procesador especializado en gráfico:
  - Tiene su propia memoria
  - Mayor eficiencia



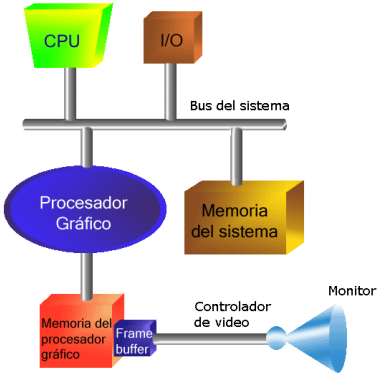
Sistema gráfico de barrido (raster)
FUNCIONMIENTO

- La imagen a desplegar en pantalla se almacena en el frame buffer
- El controlador de video lee el frame buffer y despliega el contenido
- Frame rate: frecuencia que es posible hacer esta operación



Sistema gráfico de barrido (raster)
PREGUNTAS

- ¿Cómo se logra realizar las animaciones?
- ¿Qué pasa si el almacenamiento es más lento que el despliegue de contenido?



Sistema gráfico de barrido (raster)
DOBLE BUFFER

- Un controlador de video muy rápido puede desplegar una imagen inconclusa
- Solución: usar dos buffer
  - Uno para mostrar la imagen
  - Otro para crear la imagen
- Creada la imagen, los roles se invierten
- Propuesto: Investigar cómo se implementa por hardware Z-buffer

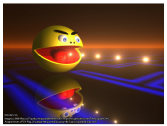
Sistema gráfico de barrido (raster)
MONITORES

Monitores con tubo de rayos catódicos:

- Pantalla de vidrio con una cubierta de diferentes compuestos de fósforo rojo, verde, azul
- El color aparece por la excitación producida por el haz de electrones
- La excitación no es continua
- El haz debe recorrer la pantalla para mantener la imagen
- Frecuencia de refresco

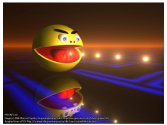
Sistema gráfico de barrido (raster)
FRAME BUFFER

- Pixel $\neq$ Grupo de fósforos.
- Pixel: Número de puntos distinguibles depende de la resolución.



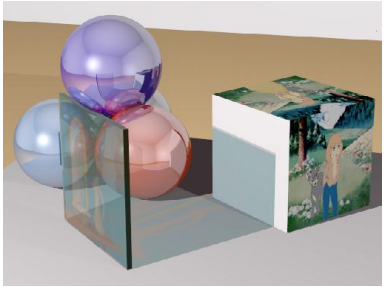
Sistema gráfico de barrido (raster)
PIXEL

- Memoria que almacena el color de cada pixel de la pantalla
  Ejemplo: 8 bits por color $\rightarrow$ 3 bytes por pixel $\rightarrow$ en resolución 640 x 480 $\rightarrow$ 900 Kbytes
- Más colores $\rightarrow$ más bits



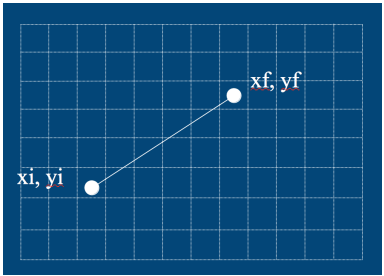
Sistema gráfico de barrido (raster)
COMPONENTE ALFA

- La componente alfa permite definir la transparencia del objeto.
- Es descrita mediante un byte
- Un color requerirá 4 bytes: 3 para el color y 1 para la transparencia



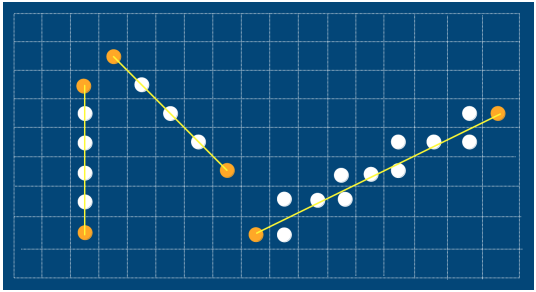
Segmento de línea

- Arreglo 2D de pixeles
- Pixeles centrados en coordenadas enteras
- $setPixel(x, y)$: función para iluminar un pixel



Segmento de línea

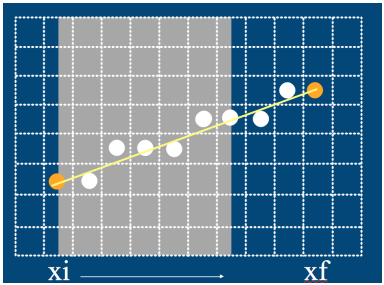
Dado los puntos extremos de un segmento: ¿Cómo identificar los pixeles que lo representan?



Segmento de línea

Algoritmo “Ingenuo”

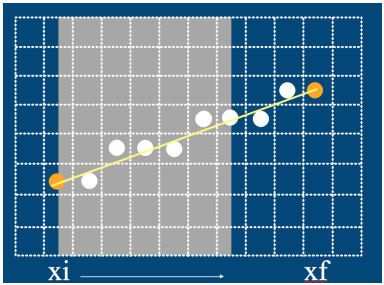
- Extremos: $(x_i, y_i) \rightarrow (x_f, y_f)$.
- Sea f la función que describe $y = f(x)$.
- $\forall x \in [x_i, x_f], setPixel(x, round(y))$.



Segmento de línea

Algoritmo “Ingenuo”

- Extremos: $(x_i, y_i) \rightarrow (x_f, y_f)$.
- Sea f la función que describe $y = f(x)$.
- $\forall x \in [x_i, x_f], setPixel(x, round(y))$.

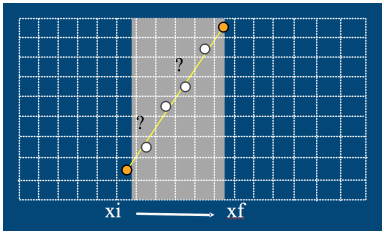


¿Problemas con este algoritmo?

Segmento de línea

Algoritmo “Ingenuo”

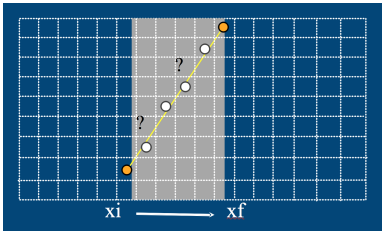
- Extremos: $(x_i, y_i) \rightarrow (x_f, y_f)$.
- Sea f la función que describe $y = f(x)$.
- $\forall x \in [x_i, x_f], setPixel(x, round(y))$.



Segmento de línea

Algoritmo “Ingenuo”

- Extremos: $(x_i, y_i) \rightarrow (x_f, y_f)$.
- Sea f la función que describe $y = f(x)$.
- $\forall x \in [x_i, x_f], setPixel(x, round(y))$.



Si la pendiente $m > 1$ la línea se puede cortar.

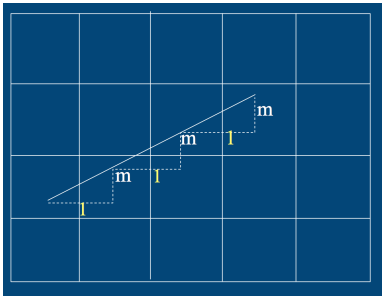
$f(x) = \frac{3}{2}x = y$

x	1	2	3	4	5	6
y	1.5	3	4.5	6	7.5	9
$round(y)$	2	3	5	6	8	9

Segmento de línea

Algoritmo Incremental

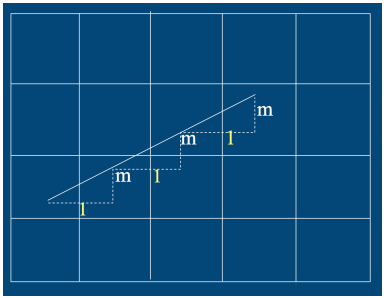
1: $x \leftarrow x_i$
2: $y \leftarrow y_i$
3: $setPixel(x, round(y))$
4: **while** $x < x_f$ **do**
5: $\quad x \leftarrow x + 1$
6: $\quad y \leftarrow y + m$
7: $\quad setPixel(x, round(y))$
8: **end while**



Segmento de línea

Algoritmo Incremental

1: $x \leftarrow x_i$
2: $y \leftarrow y_i$
3: $setPixel(x, round(y))$
4: **while** $x < x_f$ **do**
5: $\quad x \leftarrow x + 1$
6: $\quad y \leftarrow y + m$
7: $\quad setPixel(x, round(y))$
8: **end while**



Si la pendiente $m > 1$ la línea se puede cortar.
$f(x) = \frac{3}{2}x = y, x_i = 1, y_i = 1$

x	1	2	3	4	5	6
y	1	2.5	4	5.5	7	8.5
$round(y)$	1	3	4	6	7	9

Algoritmo de Bresenham

Bresenham

Definición: es un algoritmo eficiente para dibujar una línea en una pantalla raster (compuesta de pixeles).

Bresenham, J.E. "*Algorithm for Computer Control of a Digital Plotter*". IBM Systems Journal, 4(1), pp. 25-30, 1965.

Características:

- Usa aritmética entera
- Supone la pendiente m en el rango $[0, 1]$.
- Parte del punto inferior-izquierdo al punto superior-derecho.

Ecuación de la recta:
$y = f(x) = mx + c$

Punto de la recta (valor exacto):
$(x, f(x)) = (x, mx + c)$

Punto de la recta (valor en pantalla):
$(x, round(f(x))) = (x, round(mx + c))$

Bresenham
FORMALIZACION

Dado que:

$$x_2 = x_1 + \Delta x$$
$$y_1 = mx_1 + c$$
$$y_2 = mx_2 + c$$

Luego:

$$\begin{aligned} y_2 &= mx_2 + c \\ &= m(x_1 + \Delta x) + c \\ &= mx_1 + m\Delta x + c \\ &= (mx_1 + c) + m\Delta x \\ &= y_1 + m\Delta x \end{aligned}$$

Dado que:

$$x_2 = x_1 + \Delta x$$
$$y_1 = mx_1 + c$$
$$y_2 = mx_2 + c$$

Luego:

$$\begin{aligned} y_2 &= mx_2 + c \\ &= m(x_1 + \Delta x) + c \\ &= mx_1 + m\Delta x + c \\ &= (mx_1 + c) + m\Delta x \\ &= y_1 + m\Delta x \end{aligned}$$

Pero sabiendo que $\Delta x = 1$ en nuestro caso, obtenemos finalmente que:

$$y_2 = y_1 + m$$
$$x_2 = x_1 + 1$$

- Segmento de recta desde (x_i, y_i) a (x_f, y_f).
- (x, y): centro de un pixel
- ε: error (distancia) a la ordenada (y) del pixel.
- $\varepsilon \in [-0{,}5, 0{,}5]$
- $y + \varepsilon$: valor exacto de la ordenada de la recta
- m: pendiente de la recta

La gran pregunta para la siguiente iteración $(x+1)$ es: ¿Qué pixel pintamos? $(x+1, y)$ o $(x+1, y+1)$

Bresenham
FORMALIZACION

- Segmento de recta desde (x_i, y_i) a (x_f, y_f).
- (x, y): centro de un pixel
- ε: error (distancia) a la ordenada (y) del pixel.
- $\varepsilon \in [-0{,}5, 0{,}5]$
- $y + \varepsilon$: valor exacto de la ordenada de la recta
- m: pendiente de la recta

La gran pregunta para la siguiente iteración $(x+1)$ es: ¿Qué pixel pintamos? $(x+1, y)$ o $(x+1, y+1)$

Respuesta: dependerá del error y notar que éste es inicialmente 0. En la segunda iteración será m. Cuando el error llegue a ser superior a 0.5 entonces debemos "dar el salto" y al actulizarlo éste se debe volver negativo.

Segmento de línea de Bresenham

Pseudo algoritmo 1

1: $\varepsilon \leftarrow 0$
2: $y \leftarrow y_i$
3: **for** $x = x_i; x < x_f; x++$ **do**
4: $\quad setPixel(x, y)$
5: $\quad$ **if** $\varepsilon + m < 0{,}5$ **then**
6: $\quad\quad \varepsilon \leftarrow \varepsilon + m$
7: $\quad$ **else**
8: $\quad\quad y \leftarrow y + 1$
9: $\quad\quad \varepsilon \leftarrow \varepsilon + m - 1$
10: $\quad$ **end if**
11: **end for**

Segmento de línea de Bresenham

Pseudo algoritmo 1

¿Mejoras?

1: $\varepsilon \leftarrow 0$
2: $y \leftarrow y_i$
3: **for** $x = x_i; x < x_f; x++$ **do**
4: $\quad setPixel(x, y)$
5: $\quad$ **if** $\varepsilon + m < 0{,}5$ **then**
6: $\quad\quad \varepsilon \leftarrow \varepsilon + m$
7: $\quad$ **else**
8: $\quad\quad y \leftarrow y + 1$
9: $\quad\quad \varepsilon \leftarrow \varepsilon + m - 1$
10: $\quad$ **end if**
11: **end for**

Segmento de línea de Bresenham

Pseudo algoritmo 1

1: $\varepsilon \leftarrow 0$
2: $y \leftarrow y_i$
3: **for** $x = x_i; x < x_f; x++$ **do**
4: $\quad setPixel(x, y)$
5: $\quad$ **if** $\varepsilon + m < 0{,}5$ **then**
6: $\quad\quad \varepsilon \leftarrow \varepsilon + m$
7: $\quad$ **else**
8: $\quad\quad y \leftarrow y + 1$
9: $\quad\quad \varepsilon \leftarrow \varepsilon + m - 1$
10: $\quad$ **end if**
11: **end for**

¿Mejoras?
Notar que m es float.

¿Se podrá hacer solo con álgebra entera?

Segmento de línea de Bresenham

Dado que $m = \frac{\Delta y}{\Delta x} = \frac{y_f - y_i}{x_f - x_i}$

$$\begin{aligned} \varepsilon + m &< 0{,}5 \\ \varepsilon + \frac{\Delta y}{\Delta x} &< 0{,}5 \\ 2\varepsilon\Delta x + 2\Delta y &< \Delta x \\ 2(\varepsilon' + \Delta y) &< \Delta x \end{aligned}$$

Para el útlimo paso se utilizó $\varepsilon' = \varepsilon\Delta x$.

Para actualizar el error tenínamos que si $\varepsilon + m < 0{,}5$:

$$\begin{aligned} \varepsilon &\leftarrow \varepsilon + m \\ \varepsilon\Delta x &\leftarrow \varepsilon\Delta x + \Delta y \\ \varepsilon' &\leftarrow \varepsilon' + \Delta y \end{aligned}$$

Y en el caso contrario:

$$\begin{aligned} \varepsilon &\leftarrow \varepsilon + m - 1 \\ \varepsilon\Delta x &\leftarrow \varepsilon\Delta x + \Delta y - \Delta x \\ \varepsilon' &\leftarrow \varepsilon' + \Delta y - \Delta x \end{aligned}$$

Segmento de línea de Bresenham

Pseudo algoritmo 2

1: $\varepsilon' \leftarrow 0$
2: $y \leftarrow y_i$
3: **for** $x = x_i; x < x_f; x++$ **do**
4: $\quad setPixel(x, y)$
5: $\quad$ **if** $2(\varepsilon' + \Delta y) < \Delta x$ **then**
6: $\quad\quad \varepsilon' \leftarrow \varepsilon' + \Delta y$
7: $\quad$ **else**
8: $\quad\quad y \leftarrow y + 1$
9: $\quad\quad \varepsilon' \leftarrow \varepsilon' + \Delta y - \Delta x$
10: $\quad$ **end if**
11: **end for**

Segmento de línea de Bresenham

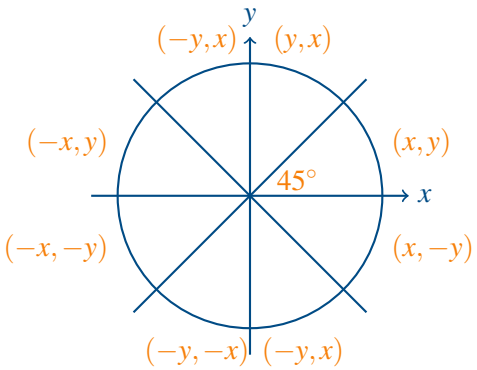
Pseudo algoritmo 2

1: $\varepsilon' \leftarrow 0$
2: $y \leftarrow y_i$
3: **for** $x = x_i; x < x_f; x++$ **do**
4: $setPixel(x, y)$
5: **if** $2(\varepsilon' + \Delta y) < \Delta x$ **then**
6: $\varepsilon' \leftarrow \varepsilon' + \Delta y$
7: **else**
8: $y \leftarrow y + 1$
9: $\varepsilon' \leftarrow \varepsilon' + \Delta y - \Delta x$
10: **end if**
11: **end for**

¡¡¡Solo valores enteros!!!

Segmento de línea

Para Bresenham en pendientes distintas al rango [0, 1] se usa simetría:



Segmento de línea

Valor de la pendiente en cada tramo