02 - Formación de imagen y representación digital

Visualización gráfica para IA

Dra. Dora Alvarado

2026-01-19

¿Cómo se forma una imagen?

Vista desde la ventana, de Joseph Nicéphore Niépce. Ver también www.hrc.utexas.edu

- Cuando trabajamos en visión por computadora / visualización de datos, utilizamos imágenes, que en realidad son arreglos numéricos: matrices, tensores, arrays de intensidades.

- Sin embargo, estos números son el resultado final de un proceso físico, óptico y electrónico altamente indirecto.

Definición 1: Una imagen es una representación visual de un objeto, escena o idea. Es una codificación de cómo la luz fue modificada al interactuar con superficies del mundo y posteriormente medida por un dispositivo.

- Las imágenes pueden ser capturadas, creadas o percibidas. Por ejemplo:
  - Imágenes físicas: Dibujos en papel, pinturas en lienzo, fotografías impresas.
  - Imágenes ópticas: Lo que percibimos con nuestros ojos o a través de lentes (como en una cámara o telescopio).
  - Imágenes mentales: Representaciones visuales que creamos en nuestra mente

Imagen de Computer Vision: Algorithms and Applications

Breve contexto histórico

- En la Grecia antigua, **Euclides (300 a.C.)** sostenía que la visión ocurría porque rayos rectilíneos emanaban del ojo hacia los objetos. En esta concepción, el ojo es un agente activo que "explora" el mundo.
- Uno de sus postulados afirmaba que un objeto deja de ser visible si queda entre rayos visuales adyacentes.

No fue sino hasta el siglo XVI que **Johannes Kepler** estableció la visión como la entendemos hoy:

- la luz entra al sistema óptico.
- se forma una imagen en una superficie receptora.
- la visión es un proceso pasivo (el ojo es un receptor).

Cadena de formación de imagen

Imagen de mybrainfitness.wordpress.com

El proceso de formación de imagen puede entenderse como una **cadena de transformaciones**:

1. La luz incide sobre superficies en el mundo.
2. Las propiedades de esas superficies modifican la luz (reflexión).
3. La luz entra a un sistema óptico (lente).
4. La energía luminosa impacta sensores físicos.
5. Señales analógicas se convierten en valores digitales.
6. El resultado final es una matriz de píxeles.

Cada uno de estos pasos **impone restricciones** y **pérdidas de información**.

Lo que recibimos (los píxeles) es solo una huella indirecta del mundo físico.

PREGUNTA: ¿¿qué información del mundo no está contenida explícitamente en una imagen?

El modelo pinhole

- El modelo pinhole (o cámara de estenopo) es la forma más sencilla de describir cómo una escena en tres dimensiones (3D) se proyecta sobre una superficie plana en dos dimensiones (2D).
- En el modelo pinhole, tenemos una caja completamente cerrada con un pequeño agujero en un lado, y todos los rayos de luz pasan en línea recta por un ese único punto (el centro de proyección).
- Debido al pequeño tamaño del agujero (pinhole), solo un rayo de luz desde cada punto del objeto logra pasar. Al chocar con la pared trasera (el plano de la imagen), se forma una imagen del objeto.
- La distancia focal (f) es la distancia entre el centro de proyección y el plano de imagen.

Efecto de la distancia focal

Importante: Como los rayos se cruzan en el agujero, la imagen proyectada en el fondo de la caja está **invertida** tanto vertical como horizontalmente. Para que los cálculos sean más intuitivos y evitar trabajar con imágenes invertidas, se suele colocar un **plano de imagen virtual** *delante* del agujero.

Considera un objeto de altura H a una distancia Z del centro de la cámara. En el modelo pinhole, la altura proyectada h h en la imagen cumple:

$$h = \frac{fH}{Z}$$

Donde:

- f: Es la **distancia focal** (la distancia entre el pinhole y el plano donde se captura la imagen).
- Z: Es la profundidad o distancia del objeto a la cámara.
- H: Es la altura del objeto.
- h: Es la altura del objeto proyectada en el plano de la imagen.

Aunque el modelo pinhole es matemáticamente perfecto, en la vida real tiene problemas que las cámaras modernas solucionan con lentes:

- Un agujero muy pequeño deja pasar tan poca luz que la foto saldría negra o necesitaría minutos de exposición.
- Además, si el agujero es *demasiado* pequeño, la luz se dispersa (difracción) y la imagen sale borrosa.
- Por último, si el agujero es muy grande, entran muchos rayos de luz desde un mismo punto, creando manchas en lugar de puntos nítidos.

Imagen de csundergrad.science.uoit.ca, original de Computer Vision: A modern approach, de Forsyth

¿Qué significa representar visualmente información en una computadora?

Imagen de stackexchange.com

- La representación digital de imágenes se basa en dos conceptos fundamentales: **muestreo** y **cuantización**.
- Estos conceptos definen cómo una señal continua (la realidad visual) se transforma en una representación discreta (la imagen digital).
    - **Muestreo**: Proceso de seleccionar puntos específicos de una señal continua para representarla en forma discreta. En imágenes, esto implica dividir el espacio bidimensional en una cuadrícula de píxeles. Cada

píxel representa la información visual en una pequeña región del espacio.

- **Cuantización**: Proceso de asignar valores discretos a las mediciones continuas obtenidas durante el muestreo. En imágenes digitales, esto se traduce en asignar un valor numérico (por ejemplo, un valor RGB) a cada píxel, basado en la intensidad de luz capturada en esa región.

- La imagen digital que almacenamos en una computadora es una versión muestreada de una función, típicamente representada como una matriz (I[i,j]). Cada entrada de esta matriz corresponde a una medición de la señal en una región pequeña del plano, no en un punto matemático exacto.
  - Cuando muestreamos una señal continua, perdemos información. Detalles más pequeños que el intervalo de muestreo simplemente desaparecen o, peor aún, se transforman en patrones falsos.
  - En visualización gráfica, esto ocurre cuando representamos fenómenos densos o altamente variables con una resolución insuficiente.

Definición de imagen

- **Definición 2:** Una imagen es una función f, que asigna un valor a cada punto de un espacio bidimensional.

  Sea $ f(x, y) $ una función de dos variables:

  - $ x $ y $ y $ son las **coordenadas espaciales** de un píxel en la imagen.

  - $ f(x, y) $ es el valor de intensidad o color en ese punto.

- En una **imagen analógica** (como una pintura), la función $f(x, y)$ es continua. Cada punto tiene un valor definido en un espacio continuo.

- En una **imagen digital**, $f(x, y)$ es discreta porque la imagen se divide en una cuadrícula de píxeles. Las coordenadas $ x, y $ y los valores $f(x, y)$ solo existen en puntos discretos.

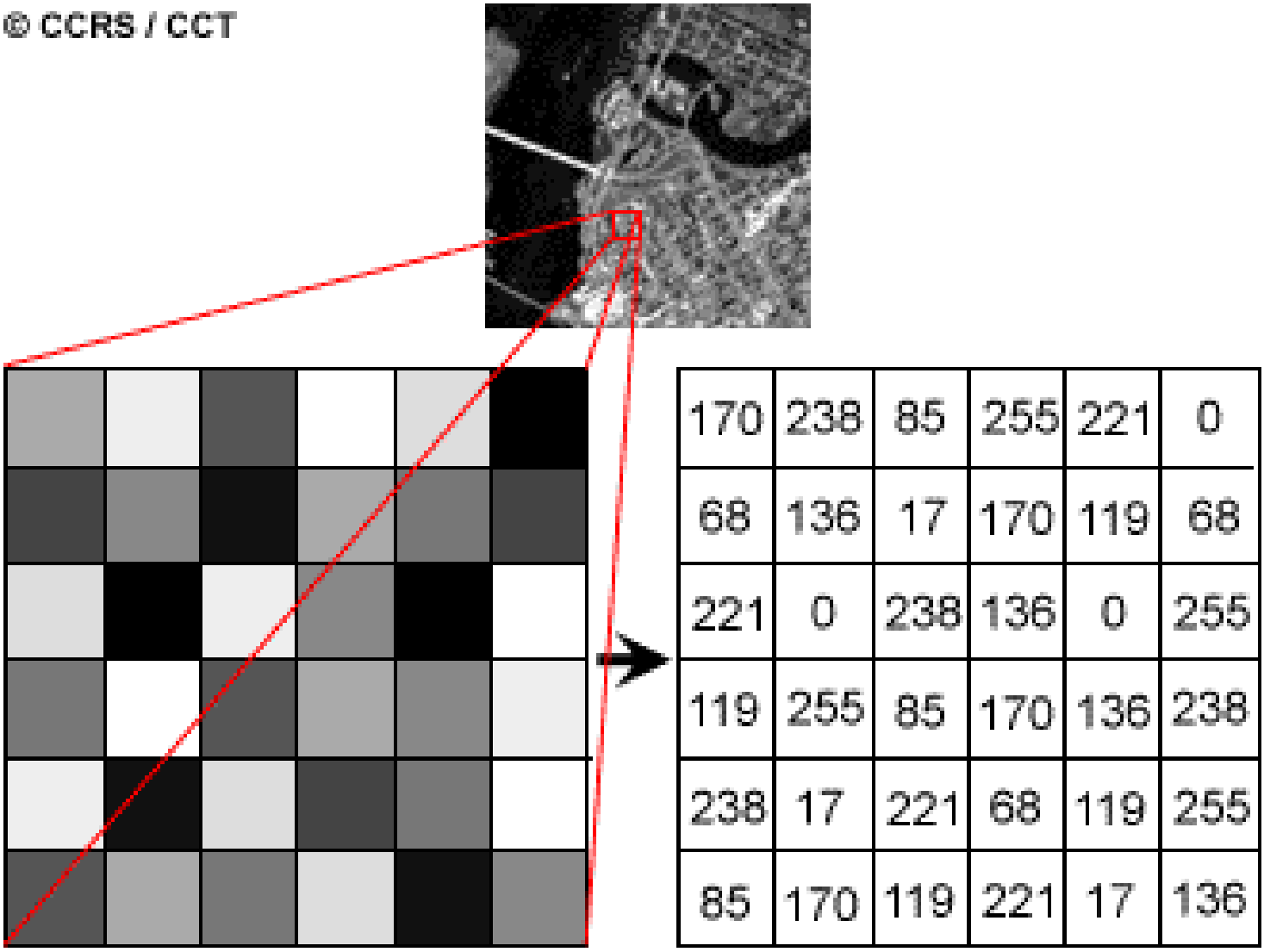


Imagen de natural-resources.canada.ca

Para imágenes en escala de grises, $f(x, y)$ es un escalar que representa la intensidad de luz, por ejemplo:

$$f(x, y) \in [0, 255]$$

donde:

- 0 = negro (ausencia de luz).
- 255 = blanco (máxima intensidad).

PREGUNTA: ¿Por qué los colores toman valores entre 0 y 255?

Por ejemplo, supongamos que tenemos una imagen de 3x3 píxeles en escala de grises:

$$f(x,y) = \begin{bmatrix} 0 & 128 & 255 \\ 64 & 192 & 128 \\ 32 & 96 & 160 \end{bmatrix}$$

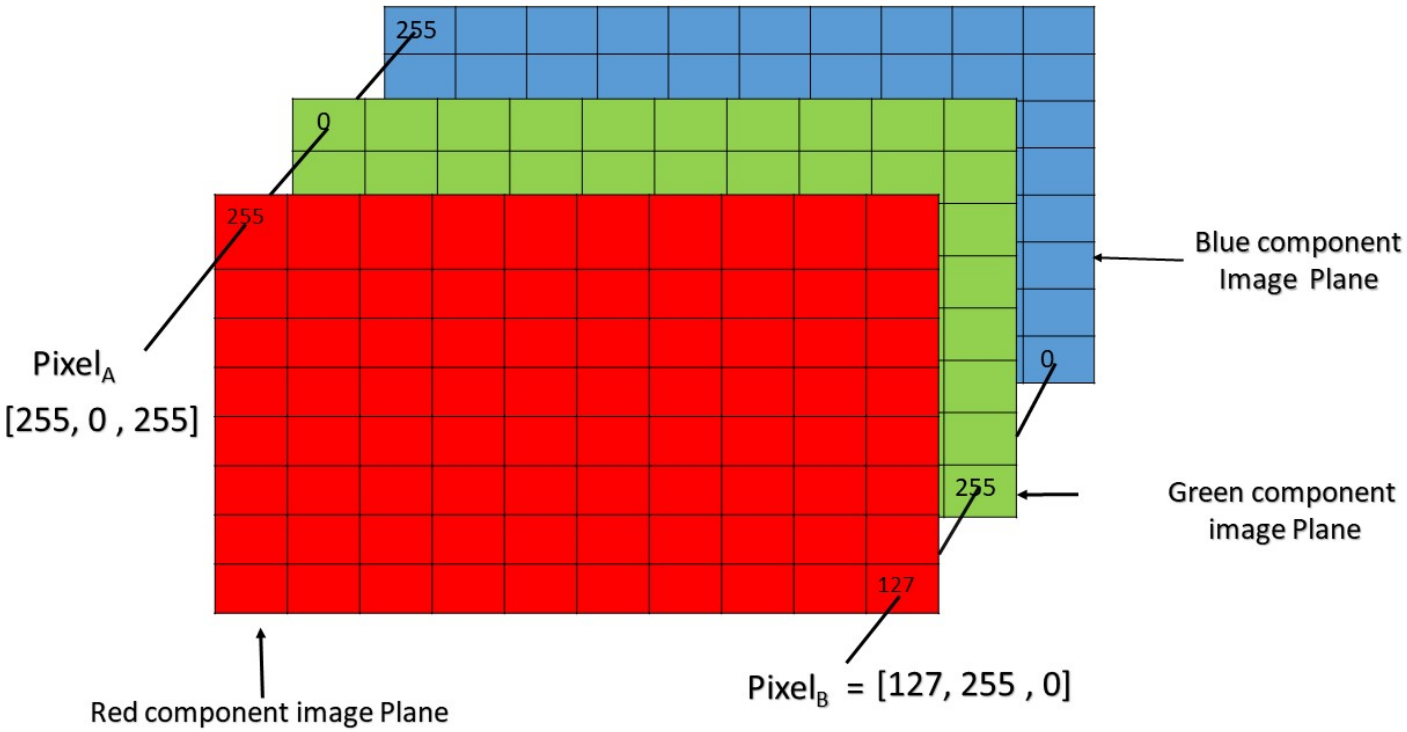
- ¿Qué color tiene el píxel en $(0,0)$?
- ¿Qué color tiene el píxel en $(0,2)$?

Para imágenes a color, $f(x,y)$ es un vector que contiene tres valores, uno para cada componente del modelo de color (generalmente RGB):

$$f(x,y) = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Donde R, G, B son las intensidades de los colores rojo, verde y azul, típicamente en el rango $[0, 255]$.

Esta representación vectorial justifica la notación $f : \mathbb{R}^2 \to \mathbb{R}^3$.



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Imagen de reddit.com

Ahora, si la imagen es a color, cada píxel será un vector:

$$f(x,y) = \begin{bmatrix} [255,0,0] & [64,64,64] & [255,0,255] \\ [255,255,0] & [0,255,255] & [0,0,255] \\ [128,128,128] & [0,255,0] & [32,32,32] \end{bmatrix}$$

Aquí:

- ¿En qué valores de x y y encontramos rojo puro?
- ¿En qué valores de x y y encontramos verde puro?
- ¿En qué valores de x y y encontramos azul puro?

El píxel

- El píxel es una **muestra** de una señal subyacente.
- Representa el resultado de integrar la información visual sobre una región del espacio y almacenarla como un valor discreto.
- Esta interpretación explica fenómenos cotidianos como el desenfoque o el aliasing. Cuando una imagen se amplía excesivamente, no aparecen nuevos detalles porque esos detalles nunca fueron muestreados. Cuando una imagen se reduce demasiado, patrones que sí existían pueden desaparecer o mezclarse.
- En visualización de datos, algo análogo ocurre cuando agregamos datos en bins demasiado grandes o cuando usamos resoluciones inadecuadas para mapas de calor.

Color indexado vs 24 bits

Las primeras pantallas utilizaban **color indexado**: un conjunto de colores, generalmente 16 o 256, podía mostrarse.

La mayoría de las pantallas actuales utilizan **24 bits**: cada color (R, G, B) puede especificarse mediante tres números de 8 bits que representan los niveles de rojo, verde y azul del color. Cualquier color que pueda mostrarse en la pantalla se compone de alguna combinación de estos tres colores "primarios".

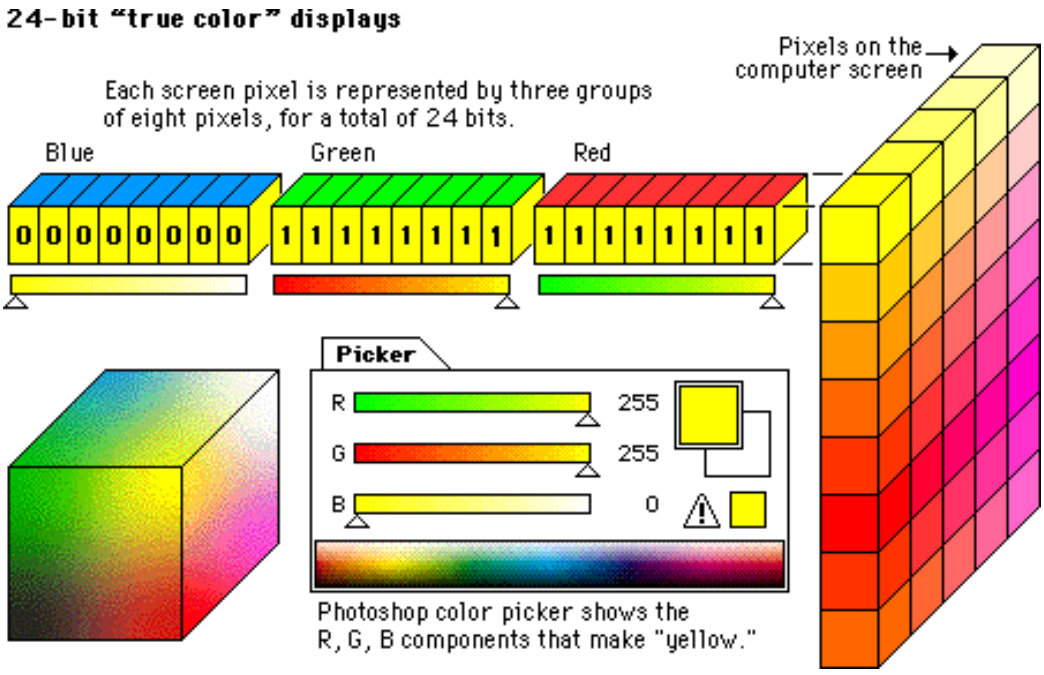


Figura 1: Color de 24 bits

Imagen de cs.ucc.ie

Buffer de cuadro (frame buffer)

En cualquier caso, los valores de color de todos los píxeles en la pantalla se almacenan en un gran bloque de memoria conocido como un **buffer de cuadro** (frame buffer).

Cambiar la imagen en la pantalla implica cambiar los valores de color almacenados en el buffer de cuadro. La pantalla se redibuja muchas veces por segundo, de modo que casi inmediatamente después de que los valores de color cambien en el buffer de cuadro, los colores de los píxeles en la pantalla cambiarán para coincidir, y la imagen mostrada se modificará.

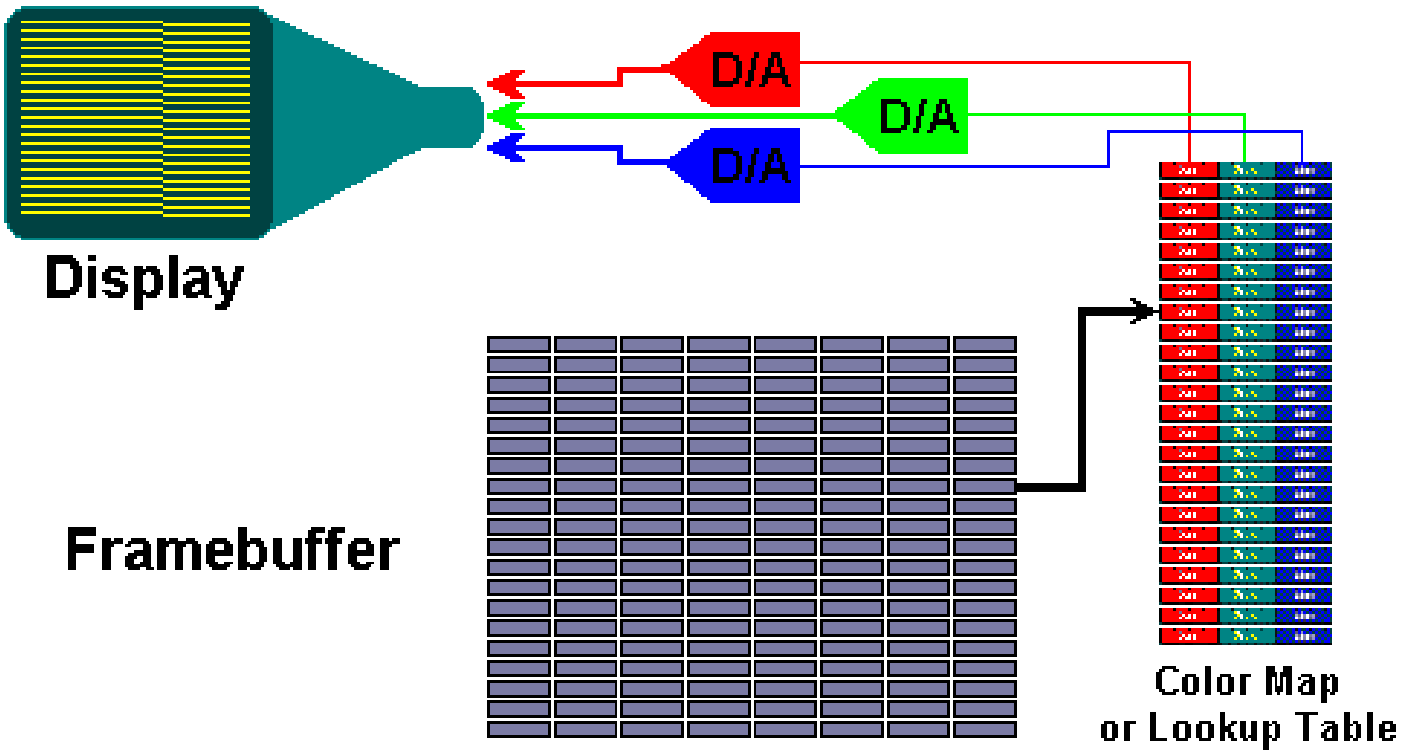


Figura 2: Buffer de cuadro

Imagen de groups.csail.mit.edu

Representación funcional de imágenes

La representación de imágenes como funciones matemáticas es una base fundamental para el análisis y procesamiento digital. Este enfoque permite:

- **Transformaciones geométricas:** Rotaciones, escalados, etc.
- **Procesamiento de píxeles:** Aplicación de filtros y operaciones matemáticas.
- **Análisis avanzado:** Reconocimiento de patrones, detección de bordes, etc.

Gráficas de ráster y gráficas vectoriales

Gráficas de ráster

El término "ráster" se refiere técnicamente al mecanismo usado en los antiguos monitores de tubo de rayos catódicos: un haz de electrones se movía a lo largo de las filas de píxeles, haciéndolos brillar.

Este haz era desviado por potentes imanes, lo que controlaba el brillo de los píxeles modulando la intensidad del haz de electrones.

Los valores de color almacenados en el buffer de cuadro determinaban la intensidad del haz. (En una pantalla a color, cada píxel tenía un punto rojo, uno verde y uno azul, que eran iluminados por separado por el haz.)

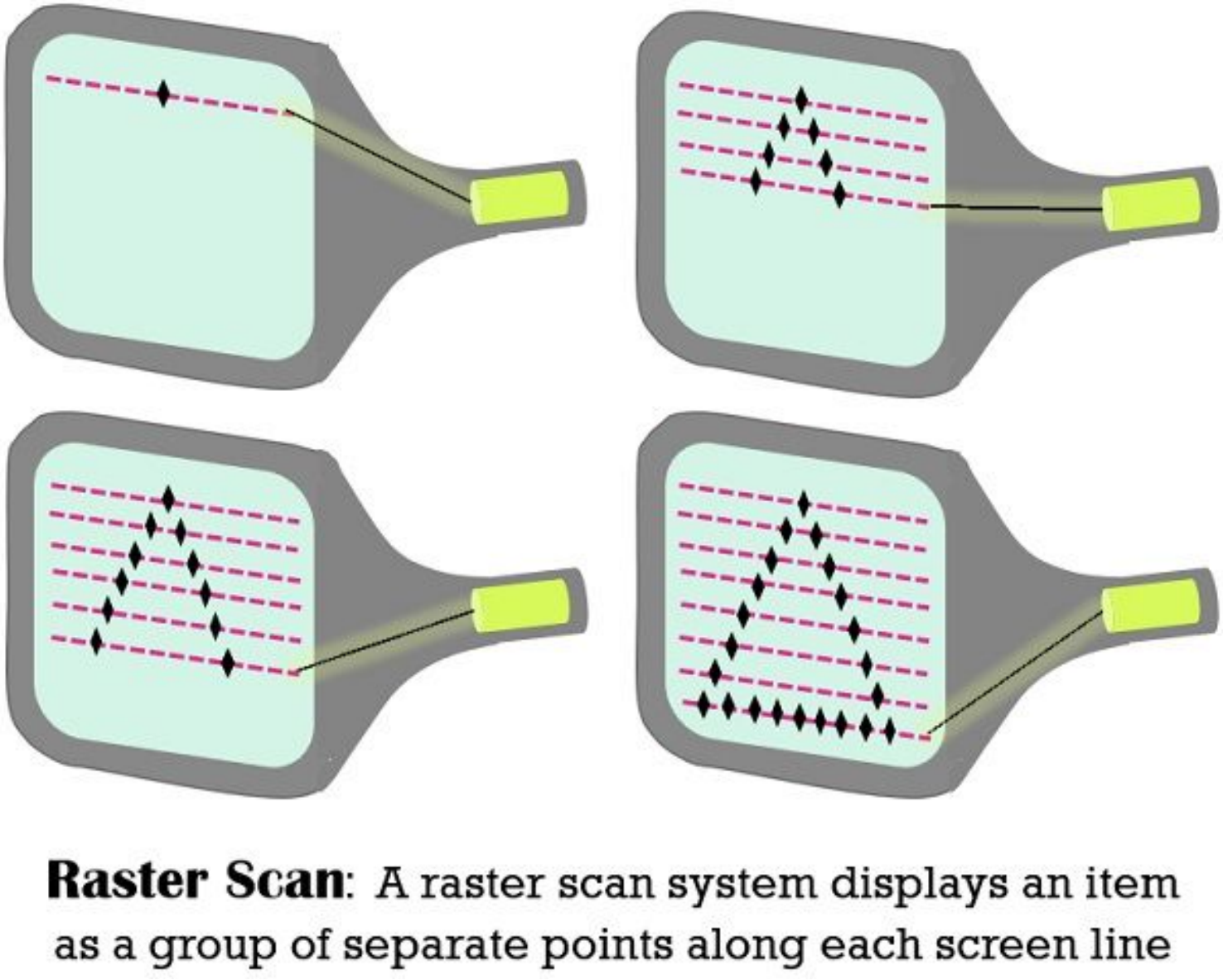


Figura 3: Raster scan

Imagen de techdifferences.com

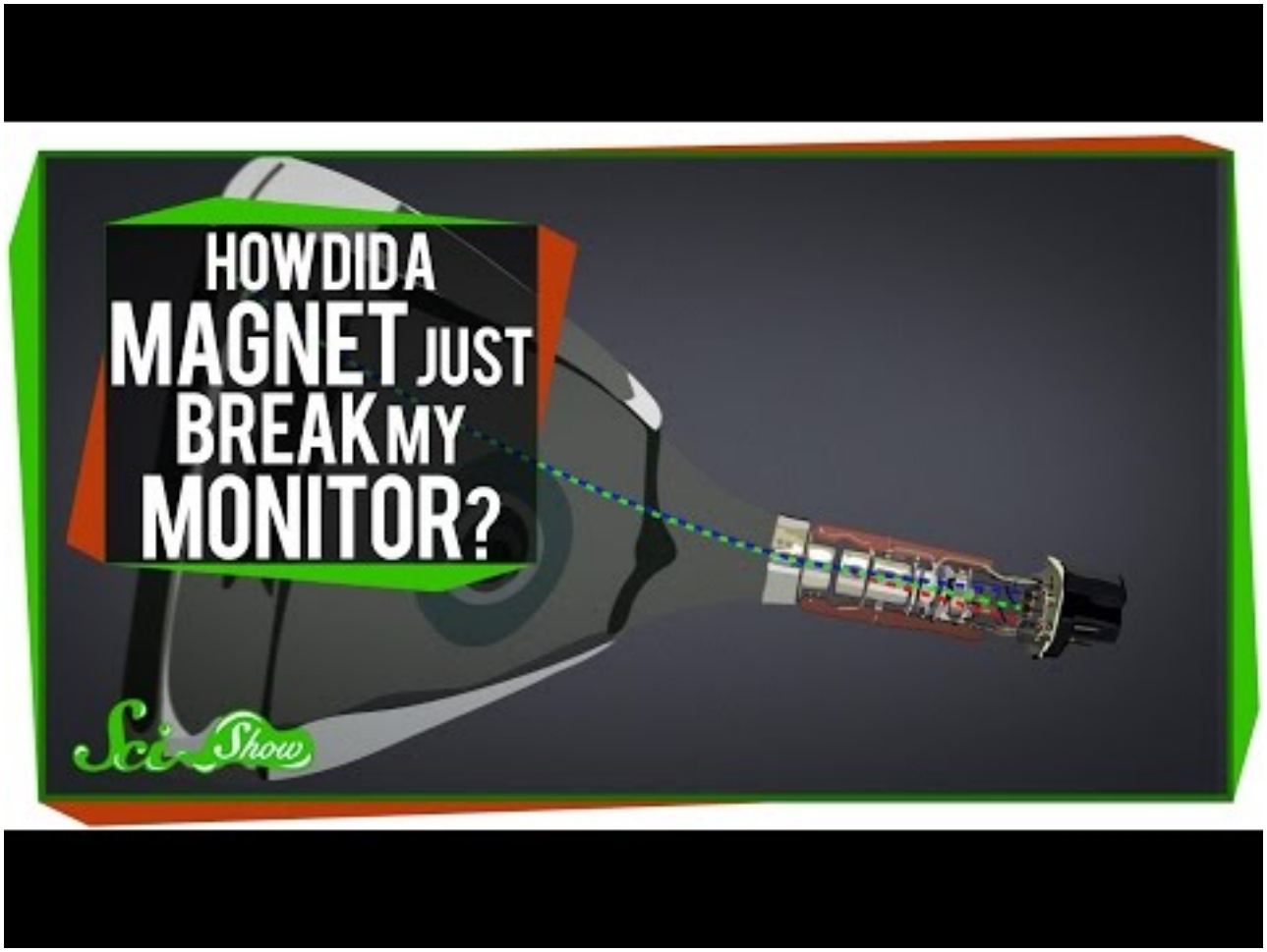


Figura 4: How Did a Magnet Just Break My Monitor?

Gráficas de ráster - Actualidad

Un monitor moderno de pantalla plana no utiliza un ráster en el mismo sentido. No hay un haz de electrones móvil. El mecanismo que controla los colores de los píxeles varía según el tipo de pantalla.

Sin embargo, la pantalla sigue estando compuesta de píxeles, y los valores de color de todos los píxeles aún se almacenan en un buffer de cuadro.

La idea de una imagen compuesta por una cuadrícula de píxeles, con valores numéricos de color para cada píxel, define a las gráficas de ráster.

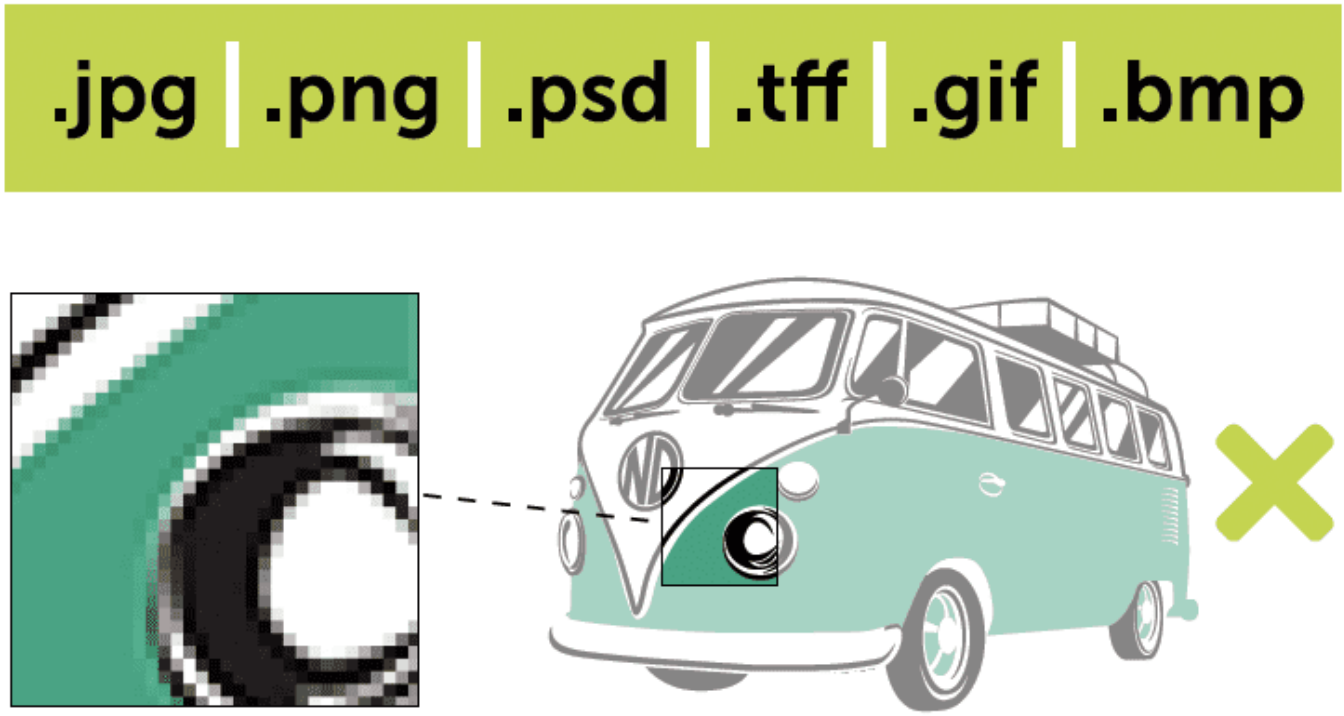


Figura 5: Ejemplo de arte ráster

Imagen de nodinx.com

Gráficas vectoriales

Aunque las imágenes en la pantalla de la computadora se representan utilizando píxeles, especificar los colores de los píxeles individuales no siempre es la mejor forma de crear una imagen.

Otra opción es especificar los objetos geométricos básicos que contiene, como líneas, círculos, triángulos y rectángulos. Esta es la idea que define a las **gráficas vectoriales: representar una imagen como una lista de formas geométricas**.

Estas formas pueden tener atributos, como el grosor de una línea o el color que rellena un rectángulo. Por supuesto, no todas las imágenes pueden componerse a partir de formas geométricas simples. Este enfoque no funcionaría para una fotografía o una pintura compleja, pero funciona bien para muchos tipos de imágenes, como planos arquitectónicos e ilustraciones científicas.

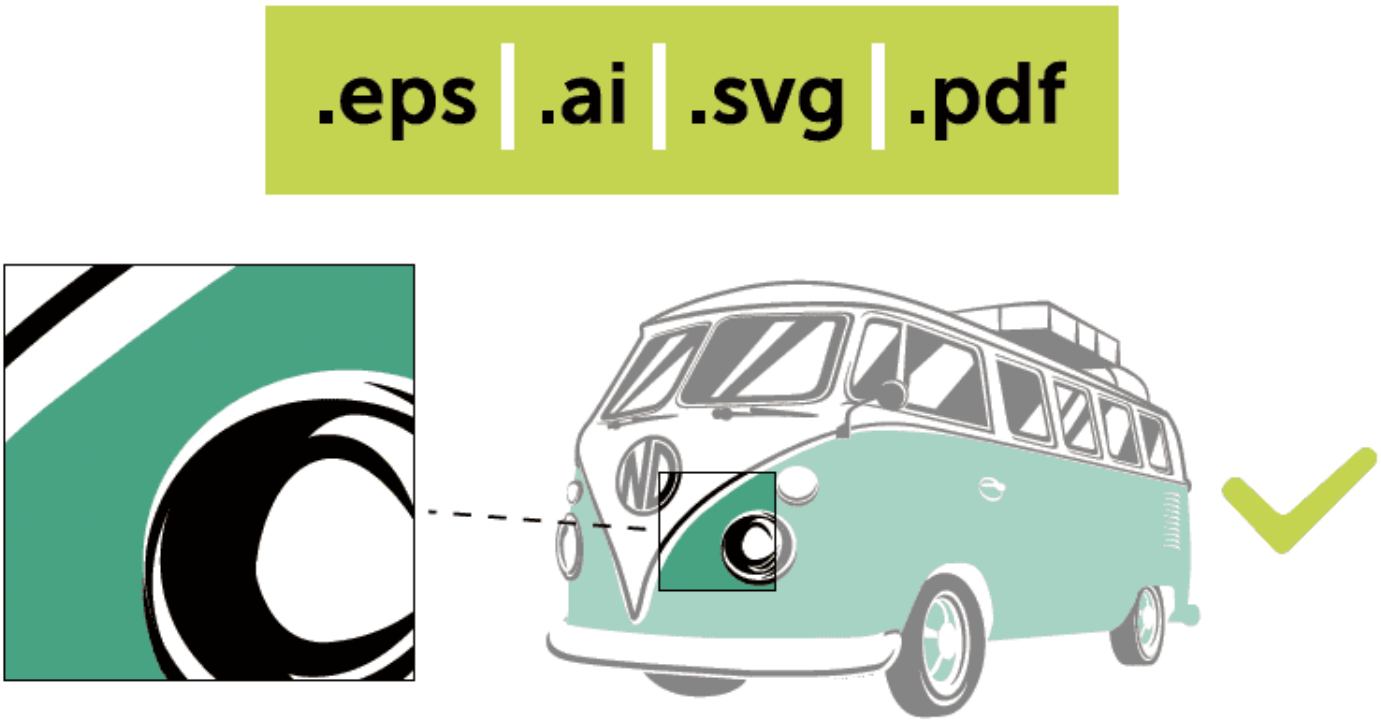


Figura 6: Ejemplo de arte vectorial

Imagen de nodinx.com

Pintar con pixeles vs. dibujar con vectores

- **Programas de pintura:** Representan imágenes como una cuadrícula de píxeles. El usuario pinta asignando colores a píxeles individuales. Ejemplo: Adobe Photoshop.
- **Programas de dibujo:** Representan imágenes como una lista de formas geométricas. Las formas se pueden mover, escalar y editar independientemente. Ejemplo: Adobe Illustrator.

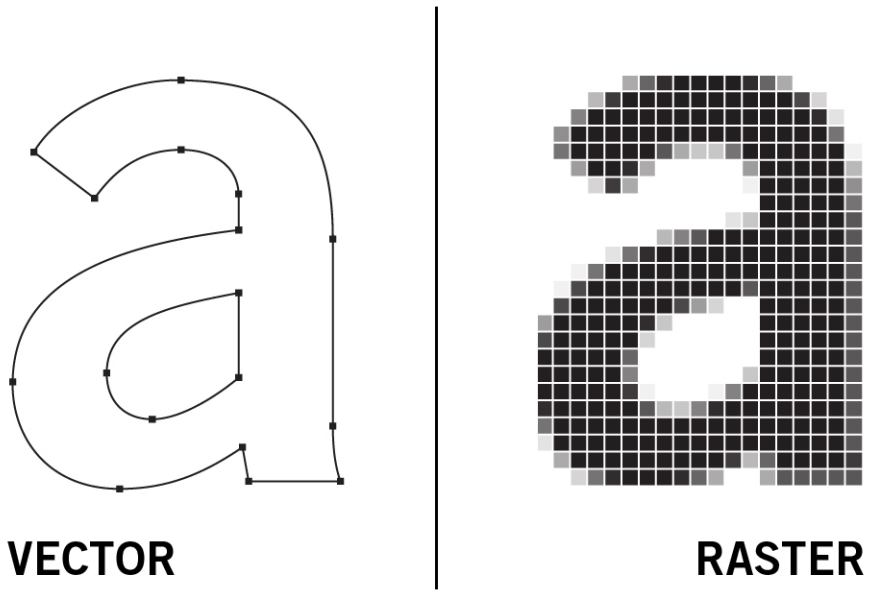


Figura 7: Vectorial vs Ráster

Imagen de ratermanis.com

Rasterización

Introducción a la rasterización

Vimos que las imágenes pueden representarse en dos formas principales:

- **Gráficas vectoriales**, que utilizan formas geométricas (como líneas, curvas y polígonos) para definir imágenes matemáticamente en un plano continuo.
- **Gráficas raster** o imágenes rasterizadas, que representan las imágenes como una matriz discreta de píxeles, donde cada píxel tiene un valor asociado (como color o intensidad).

La **rasterización** es el puente entre estas dos representaciones.

Definición de rasterización

Definición 1: La **rasterización** (también conocida como **scan-conversion**) es el proceso mediante el cual se convierten **primitivas geométricas vectoriales** en una representación discreta de píxeles que pueden mostrarse en dispositivos rasterizados, como monitores o pantallas.

Una **primitiva** es una forma geométrica básica utilizada como bloque de construcción para modelar y representar escenas. Ejemplos comunes de primitivas incluyen:

- **Líneas**.
- **Círculos**.
- **Polígonos**, especialmente triángulos en gráficos 3D.

Estas primitivas se definen mediante vértices en un espacio continuo y se transforman a un espacio discreto durante el proceso de rasterización.

Otra definición de rasterización

Definición 2: En otras palabras, la **rasterización** consiste en determinar qué píxeles de una cuadrícula representan mejor a una primitiva geométrica dada.

Sistema de coordenadas en la pantalla

En matemáticas, las coordenadas se definen en un sistema cartesiano continuo e infinito, utilizando números reales para especificar posiciones. Ej. (-12.4, 10.3)

Una pantalla física organiza los píxeles en una cuadrícula discreta, también conocidas como **coordenadas de dispositivo** o **coordenadas de píxeles**. Ej. (10, 6)

- El espacio de coordenadas está compuesto por valores enteros positivos (X, Y), donde cada posición corresponde a un píxel en la pantalla.
- El origen $(0, 0)$ está generalmente ubicado en la esquina superior izquierda de la pantalla.
- El eje Y apunta hacia abajo (a diferencia de los sistemas cartesianos habituales, donde Y apunta hacia arriba).

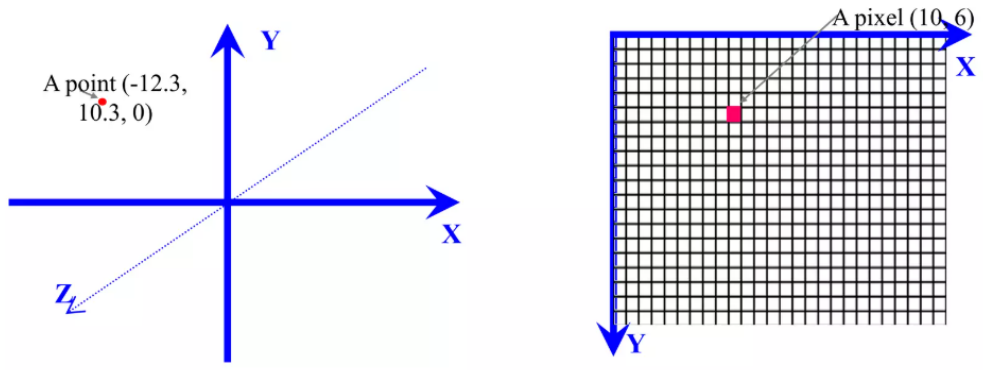


Figura 8: Sistema de coordenadas

Imagen de slideshare.net

¿Cómo dibujar un punto en una pantalla?

Para realizar el mapeo correcto de coordenadas de modelo a coordenadas de pantalla, es necesario conocer el rango de valores del modelo (es decir, el rango en el espacio continuo que queremos representar en la pantalla). Esto se conoce como el **espacio del modelo** o **espacio de mundo**.

1. **Definir los límites del espacio del modelo**
   - x_{min} y x_{max}: los valores mínimo y máximo de x en el modelo.
   - y_{min} y y_{max}: los valores mínimo y máximo de y en el modelo.
2. **Definir la resolución de la pantalla**
   - **Resolución** $(W \times H)$ (ancho y alto de la pantalla en píxeles).
3. **Hacer el mapeo de coordenadas**
   - Para convertir un punto $(x_{\text{modelo}}, y_{\text{modelo}})$ del espacio del modelo al espacio de pantalla $(x_{\text{pantalla}}, y_{\text{pantalla}})$, usamos las siguientes fórmulas:

$$x_{\text{pantalla}} = \frac{x_{\text{modelo}} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \cdot (W - 1)$$

$$y_{\text{pantalla}} = \left(1 - \frac{y_{\text{modelo}} - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}}\right) \cdot (H - 1)$$

Es decir, cada coordenada del modelo se escala para ajustarse al rango de píxeles de la pantalla $[0, W-1]$ para x y $[0, H-1]$ para y.

En las coordenadas de pantalla, el eje Y apunta hacia abajo, mientras que en el modelo normalmente apunta hacia arriba. Por eso usamos $1 - \ldots$ en la fórmula de y.

Ejemplo: Mapeo de punto

Por ejemplo, si tengo los siguientes límites del espacio del modelo y resolución:

- $x_{\text{min}} = -10$, $x_{\text{max}} = 10$
- $y_{\text{min}} = -5$, $y_{\text{max}} = 5$
- $W = 800$, $H = 600$

y deseo mapear el punto: $(x_{\text{modelo}}, y_{\text{modelo}}) = (-5, 2)$.

Cálculo del mapeo:

$$x_{\text{pantalla}} = \frac{-5-(-10)}{10-(-10)} \cdot (800-1) = \frac{5}{20} \cdot 799 = 199.75 \approx 200$$

$$y_{\text{pantalla}} = \left(1 - \frac{2-(-5)}{5-(-5)}\right) \cdot (600-1) = \left(1 - \frac{7}{10}\right) \cdot 599 = 0.3 \cdot 599 = 179.7 \approx 180$$

El punto $(-5, 2)$ en el modelo se mapea al píxel $(200, 180)$ en la pantalla.

¿Cómo dibujar una línea en una pantalla?

Dados dos puntos (x_1, y_1) y (x_2, y_2), necesitamos trazar una línea entre ellos.

Una línea en el espacio del modelo es continua, pero las pantallas están compuestas por una cuadrícula discreta de píxeles. El reto es aproximar esta línea continua seleccionando los píxeles correctos para representar la línea.

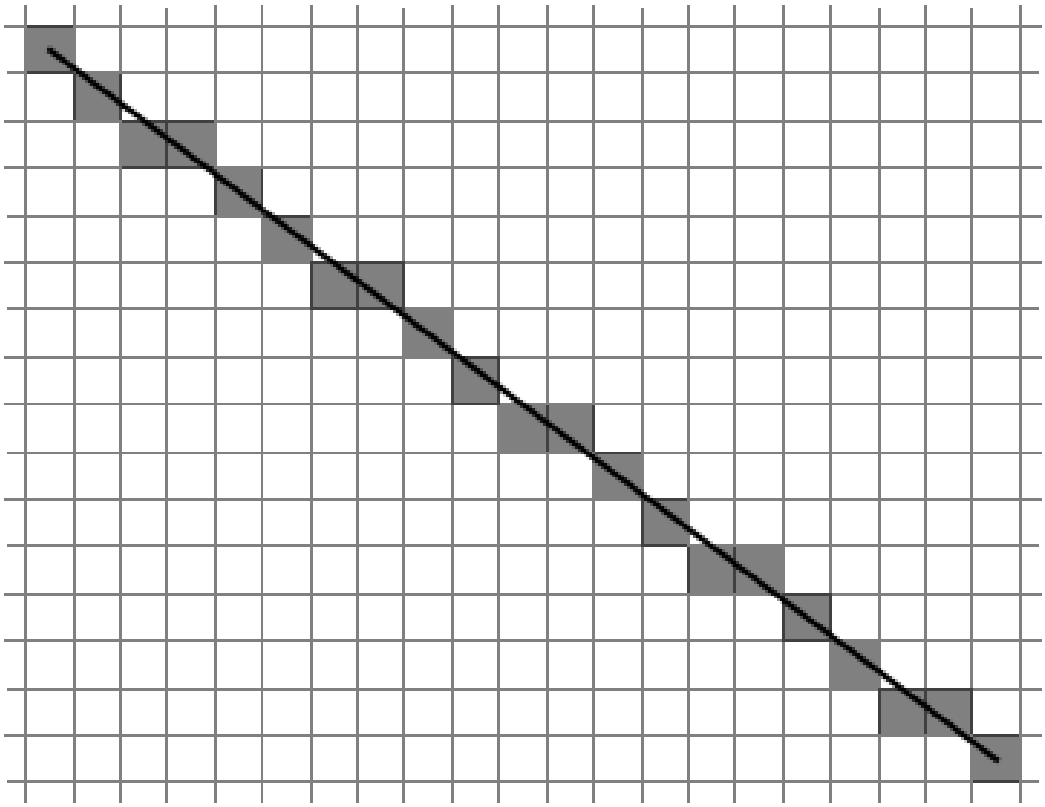


Figura 9: Línea discreta

Imagen de mathematica.stackexchange.com

> ¿Alguna idea de cómo hacer esto?

Enfoque ingenuo

La forma más sencilla sería evaluar cada píxel dentro de la cuadrícula de la pantalla para verificar si está lo suficientemente cerca de la línea.

1. Partimos de la forma implícita de la línea:

$$l(x, y) = ax + by + c = 0$$

   Donde: $a = y_2 - y_1$, $b = x_1 - x_2$, $c = x_2y_1 - x_1y_2$

2. Una curva en **forma implícita** se define mediante una función $f(x, y)$, la cual determina si un punto específico (x, y) se encuentra:
   - **Dentro** de la curva ($f(x, y) < 0$).
   - **Sobre** la curva ($f(x, y) = 0$).
   - **Fuera** de la curva ($f(x, y) > 0$).
3. Debemos visitar toda la cuadrícula, evaluando la función para cada píxel, para determinar cuáles están dentro de la línea y colorearlos.

Nota: Este proceso es extremadamente **ineficiente**, ya que revisa cada píxel de la cuadrícula, incluso aquellos que están muy lejos de la línea.

Algoritmos de rasterización de líneas

Este problema se resuelve utilizando **algoritmos para rasterización de líneas**, como:

- **DDA (Digital Differential Analyzer)**: usa incrementos uniformes para calcular los valores intermedios de la línea y determinar los píxeles más cercanos.
- **Algoritmo de Bresenham**: Más eficiente que el DDA, ya que solo utiliza operaciones aritméticas enteras. Es ideal para determinar qué píxel está más cerca de la línea real en cada paso.

Algoritmo DDA (Digital differential analyzer)

El algoritmo DDA calcula los valores intermedios utilizando la ecuación de la pendiente de la línea: $y = mx + b$

Pasos:

Entrada: (x1, y1), (x2, y2)

1. Calcular dx = x2 - x1
2. Calcular dy = y2 - y1
3. pasos = máximo(|dx|, |dy|)
4. Δx = dx / pasos
5. Δy = dy / pasos
6. Inicializar x = x1, y = y1
7. Para i desde 0 hasta pasos:
   - Dibujar el píxel en (redondear(x), redondear(y))
   - x = x + Δx
   - y = y + Δy
8. Fin

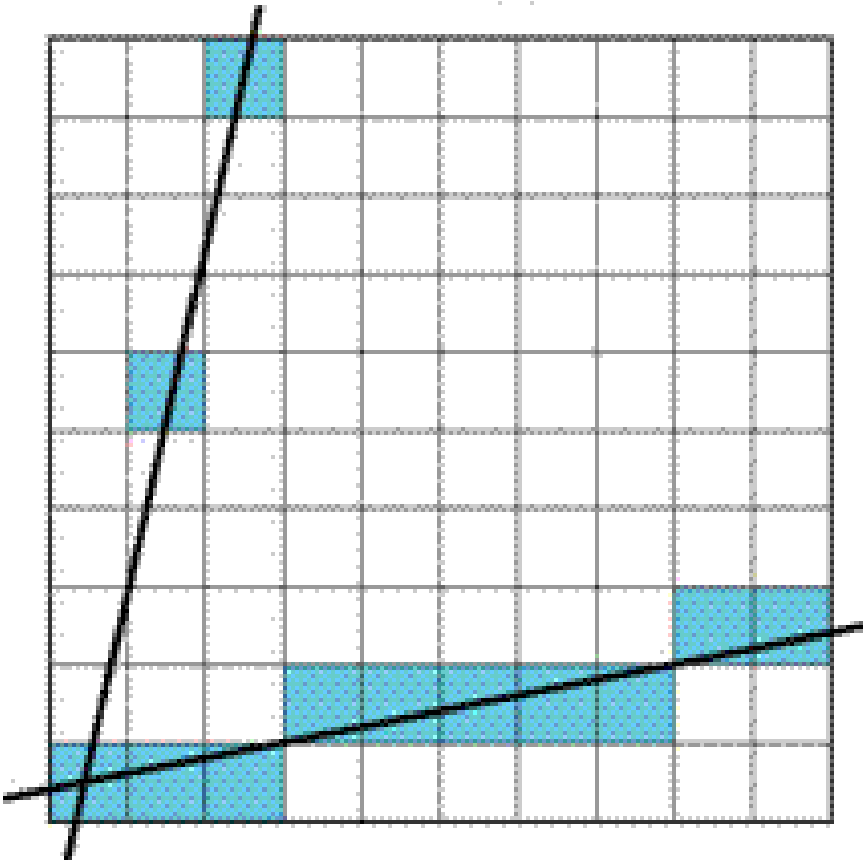


Figura 10: DDA animation

Imagen de jcsites.juniata.edu

Ejemplo:

Supongamos que queremos dibujar una línea desde $(2, 2)$ hasta $(8, 5)$:

Solución DDA

1. dx = 8 - 2 = 6
2. dy = 5 - 2 = 3
3. pasos = max(6, 3) = 6
4. Δx = dx / pasos = 6 / 6 = 1
5. Δy = dy / pasos = 3 / 6 = 0.5
6. x = 2, y = 2
7. **Iteraciones**:
   - Paso 0:
     - Dibujar pixel en $(redondear(2), redondear(2))$
     - x = x + Δx = 2 + 1 = 3
     - y = y + Δy = 2 + 0.5 = 2.5
   - Paso 1:
     - Dibujar pixel en $(redondear(3), redondear(2.5))$
     - x = x + Δx = 3 + 1 = 4

    - $y = y + \Delta y = 2.5 + 0.5 = 3$
- Paso 2:
    - Dibujar pixel en $(redondear(4), redondear(3))$
    - $x = x + \Delta x = 4 + 1 = 5$
    - $y = y + \Delta y = 3 + 0.5 = 3.5$
- ...

Algoritmo de Bresenham

El algoritmo de Bresenham optimiza el proceso de dibujo al evitar cálculos con números decimales. En su lugar, trabaja exclusivamente con números enteros.

El algoritmo utiliza una **variable de decisión** (p) que evalúa si el próximo píxel debe ser dibujado en la dirección principal (por ejemplo, en x) o si también debe ajustarse en la dirección secundaria (en y).

- Si la línea es más horizontal ($|dx| > |dy|$), se avanza píxel por píxel en la dirección x (la dirección principal) y se decide cuándo ajustar y.
- Si la línea es más vertical ($|dy| > |dx|$), se avanza píxel por píxel en la dirección y y se decide cuándo ajustar x

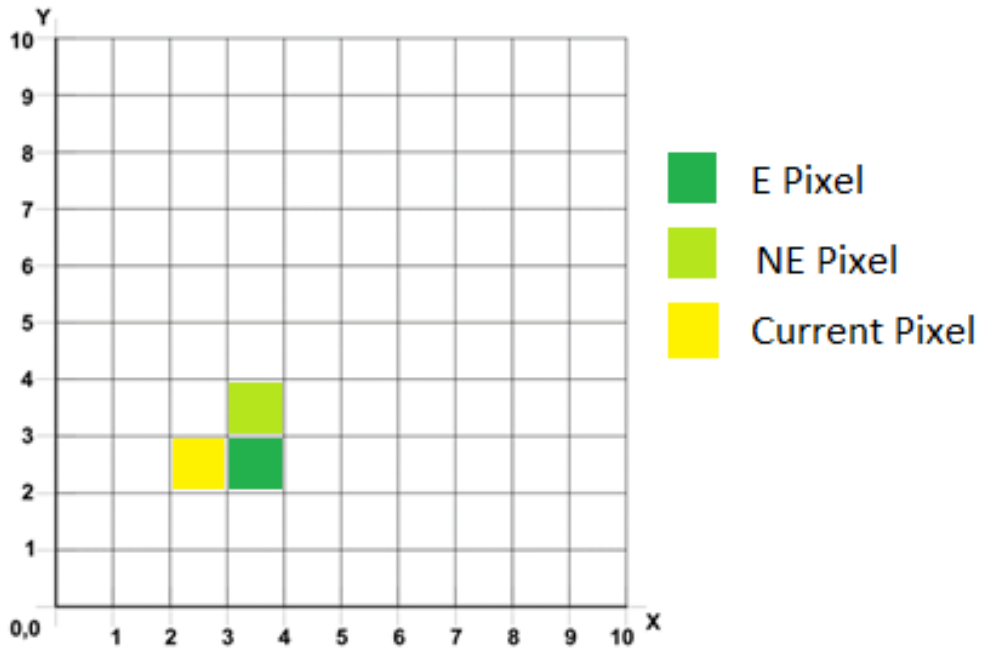


Figura 11: Bresenham

Imagen de medium.com

Video explicativo - Bresenham

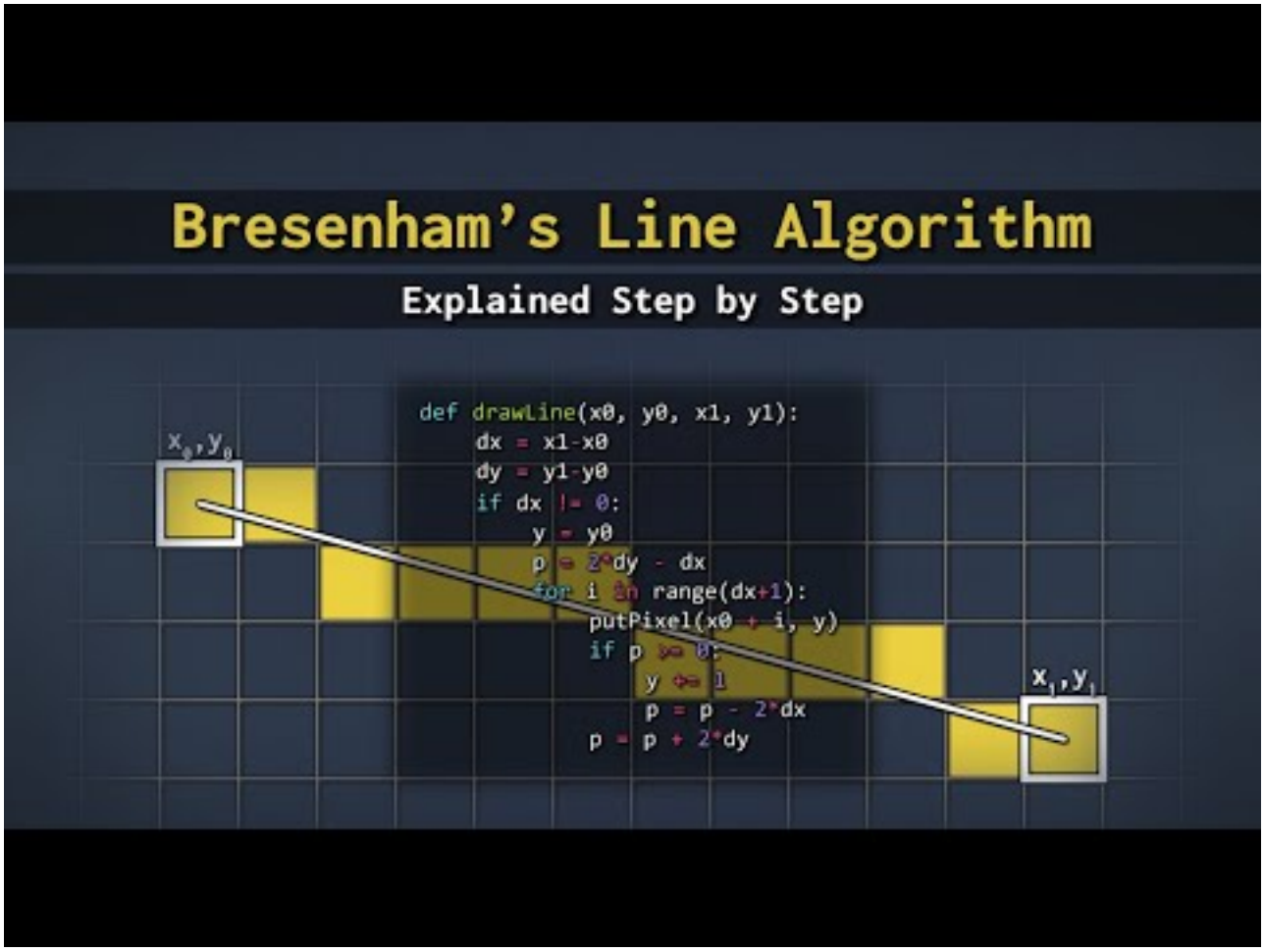


Figura 12: Algoritmo de Bresenham

Pasos del algoritmo de Bresenham

Entrada: (x1, y1), (x2, y2)

1. Calcular dx = x2 - x1, dy = y2 - y1
2. Inicializar x = x1, y = y1
3. Si |dx| > |dy| (línea más horizontal):
   - Si x1 > x2, intercambiar: x1, x2 = x2, x1; y1, y2 = y2, y1; dx = -dx
   - incremento_x = 1
   - incremento_y = 1 si dy>=0, de lo contrario incremento_y=-1
   - p = 2 * dy - dx (término de decisión inicial)
   - Para x desde x1 hasta x2 (en incrementos de tamaño incremento_x):
     - Dibujar el píxel en (x, y)
     - Si p < 0: p = p + 2 * dy
     - En otro caso: p = p + 2 * dy - 2 * dx; y = y + incremento_y
4. Si |dy| > |dx| (línea más vertical):
   - Si y1 > y2, intercambiar: x1, x2 = x2, x1; y1, y2 = y2, y1; dy = -dy
   - incremento_y = 1
   - incremento_x = 1 si dx>=0, de lo contrario incremento_x=-1
   - p = 2 * dx - dy

- Para y desde y1 hasta y2 (en incrementos de tamaño incremento_y):
  - Dibujar el píxel en (x, y)
  - Si p < 0: p = p + 2 * dx
  - En otro caso: p = p + 2 * dx - 2 * dy; x = x + incremento_x

Ejemplo:

Supongamos que queremos dibujar una línea desde $(2, 2)$ hasta $(8, 5)$:

Solución

1. Calcular dx, dy:
   - $dx = 8 - 2 = 6$
   - $dy = 5 - 2 = 3$.
2. Inicializar:
   - x = 2
   - y = 2
3. Determinar incrementos:
   - incremento_x = 1
   - incremento_y = 1
4. |dx| > |dy|, por lo tanto:
   - p = 2 * dy - dx = 2 * 3 - 6 = 0
   - **Iteraciones:**
     - Paso 0 (x = 2, y = 2):
       - Dibujar pixel en $(x, y) = (2, 2)$
       - p = p + 2 * dy - 2 * dx = 0 + 2 * 3 - 2 * 6 = - 6
       - y = y + incremento_y = 2 + 1 = 3
     - Paso 1 (x = 3, y = 3):
       - Dibujar pixel en $(x, y) = (3, 3)$
       - p = p + 2 * dy = -6 + 2 * 3 = 0
     - Paso 2 (x = 4, y = 3):
       - Dibujar pixel en $(x, y) = (4, 3)$
       - p = p + 2 * dy - 2 * dx= 0 + 2 * 3 - 2 * 6= -6
       - y = y + incremento_y = 3 + 1 = 4
     - Paso 3 (x = 5, y = 4):
       - Dibujar pixel en $(x, y) = (5, 4)$
       - p = p + 2 * dy = -6 + 2 * 3 = 0
   - …

Ejercicio:

Dibuja la línea desde $(3, 4)$ hasta $(6, 8)$ utilizando: 1. El algoritmo *dummy* (usando la forma implícita). 2. El algoritmo DDA. 3. El algoritmo de Bresenham.

Para cada paso, calcula los píxeles seleccionados y compáralos. ¿Cuál es más eficiente en términos de operaciones aritméticas?

Tarea (parte 1)

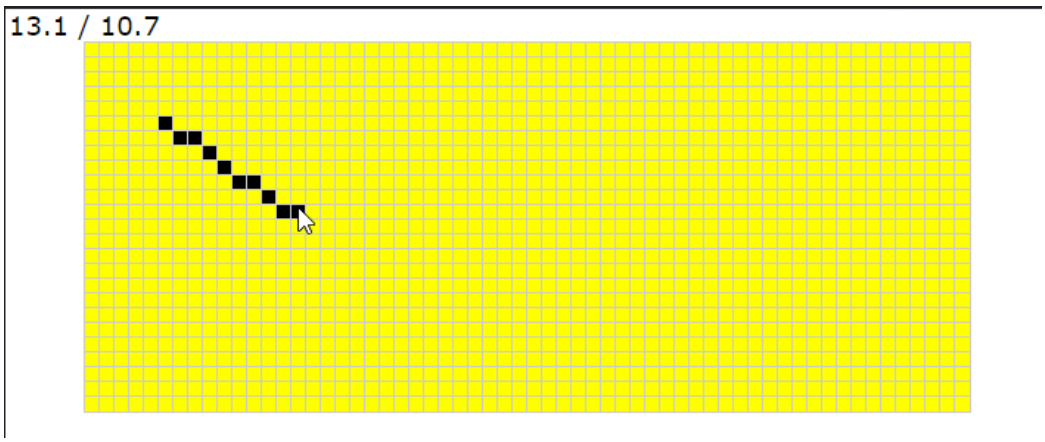


Imagen de medium.com

Rasterización de otras primitivas gráficas

Rasterización de Círculos

- La rasterización de un círculo también utiliza la idea de calcular qué píxeles deben encenderse para aproximar la forma curva.
- Para esto, podemos usar el **algoritmo de Bresenham para círculos**, que evita cálculos con decimales y trabaja con enteros de manera eficiente.

Algoritmo de Bresenham para círculos

Entrada: Centro del círculo (xc, yc) *y su radio* r.

1. Inicializa:
   - $x = 0$
   - $y = r$
   - $p = 1 - r$ (**término de decisión inicial**).
2. Dibuja el primer octante (los otros puntos se reflejan automáticamente aprovechando la simetría del círculo). Mientras $x \leq y$:
   - Dibuja los **ocho píxeles reflejados** utilizando la simetría del círculo:
     - $(xc + x, yc + y)$
     - $(xc - x, yc + y)$
     - $(xc + x, yc - y)$
     - $(xc - x, yc - y)$
     - $(xc + y, yc + x)$
     - $(xc - y, yc + x)$

- $(xc + y, yc - x)$
- $(xc - y, yc - x)$

- Si $p < 0$ (el siguiente píxel está más cerca del borde):
  - $p = p + 2x + 3$
- Si $p \geq 0$ (se necesita ajustar hacia adentro):
  - $p = p + 2x - 2y + 5$
  - $y = y - 1$
- Incrementa $x = x + 1$.

Tarea (parte 2)

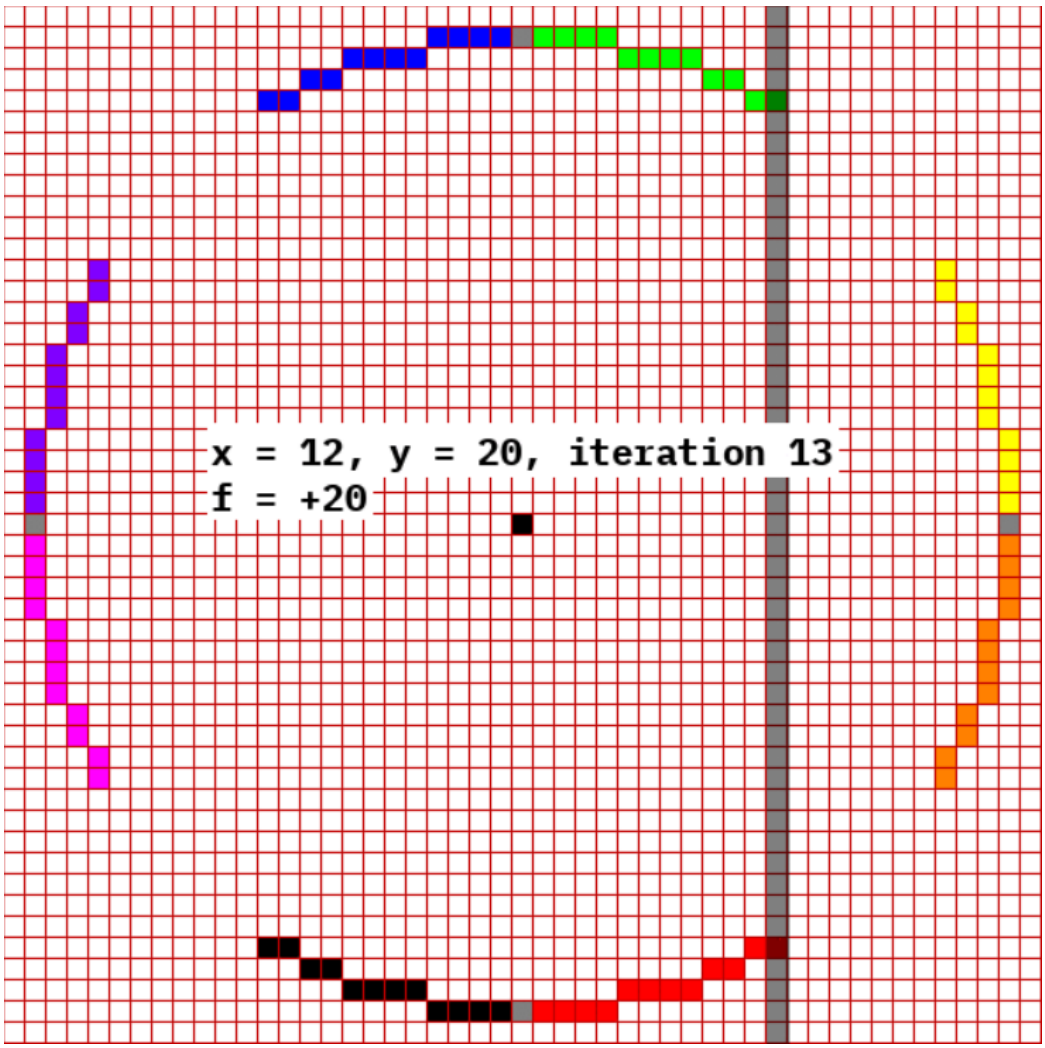


Imagen de wikipedia.org

Rasterización de Triángulos y polígonos

- Los triángulos y polígonos son fundamentales en gráficas por computadora, ya que casi todas las formas complejas se descomponen en triángulos.

Algoritmo de escaneo de triángulos

Para rasterizar un triángulo, el enfoque más común es el **algoritmo de escaneo (scanline)**:

1. Ordena los vértices del triángulo por su coordenada y.
2. Encuentra los bordes activos (puntos donde la línea del triángulo cruza cada escaneo en y).
3. Para cada fila y entre los límites superior e inferior:
   - Calcula las intersecciones de los bordes con esa fila.
   - Llena los píxeles entre las intersecciones.

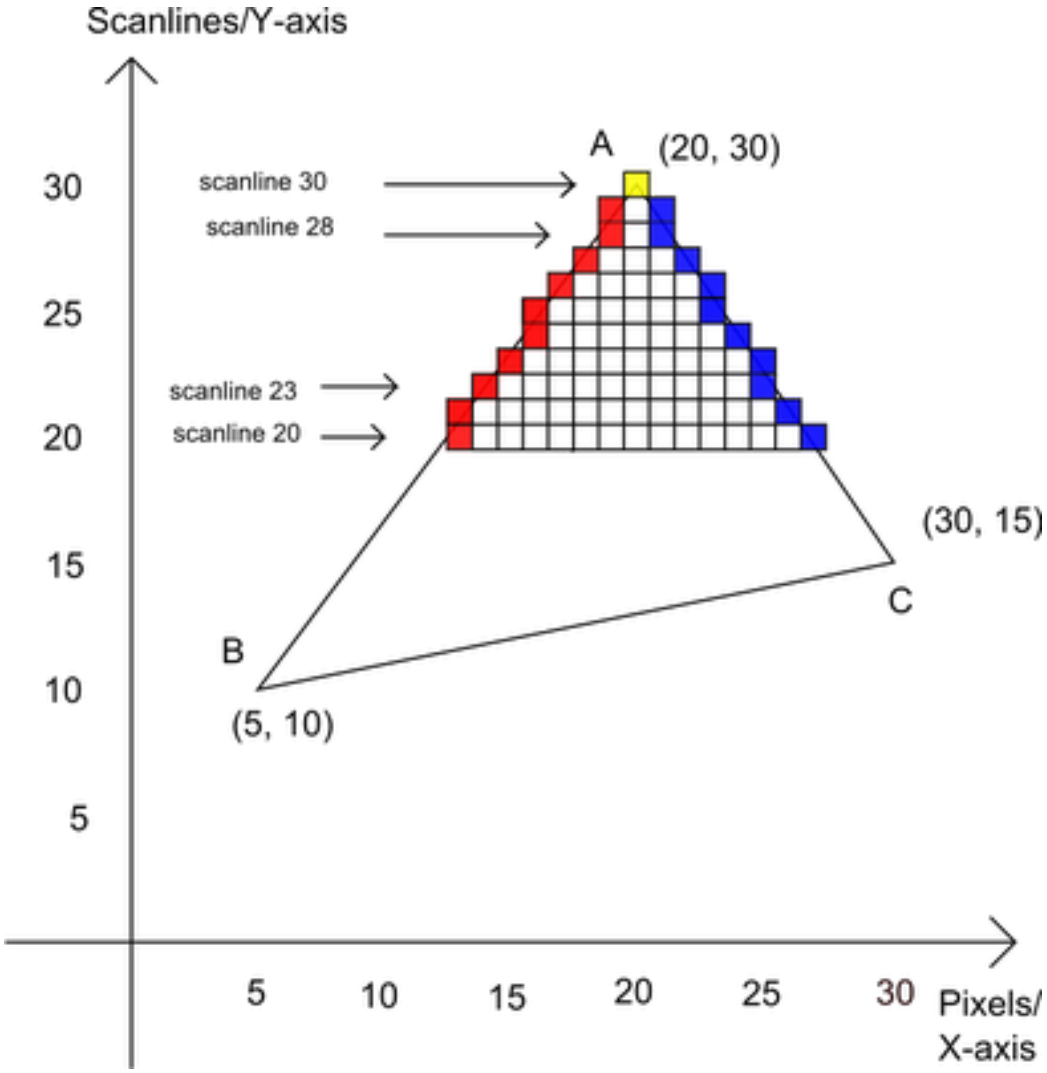
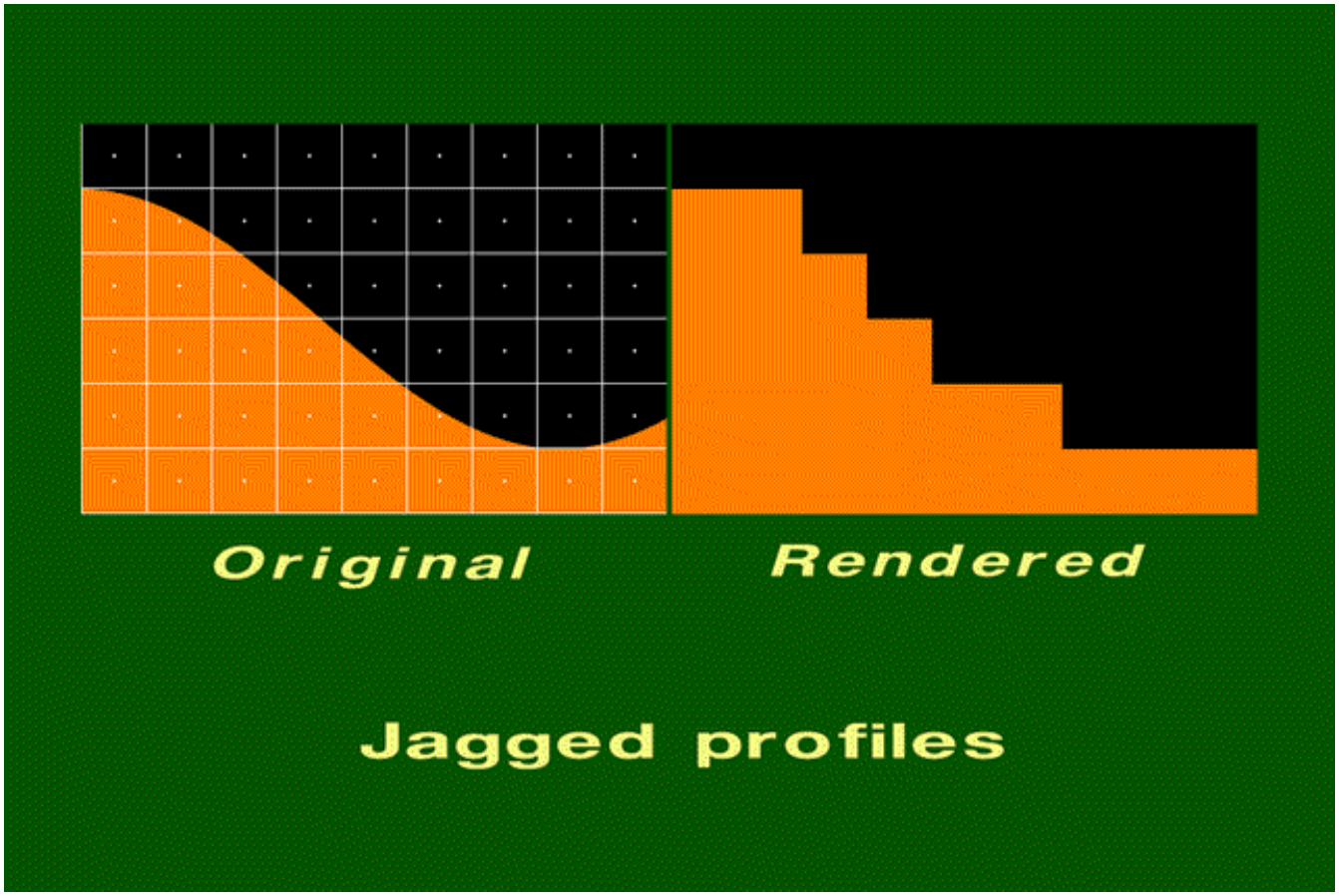


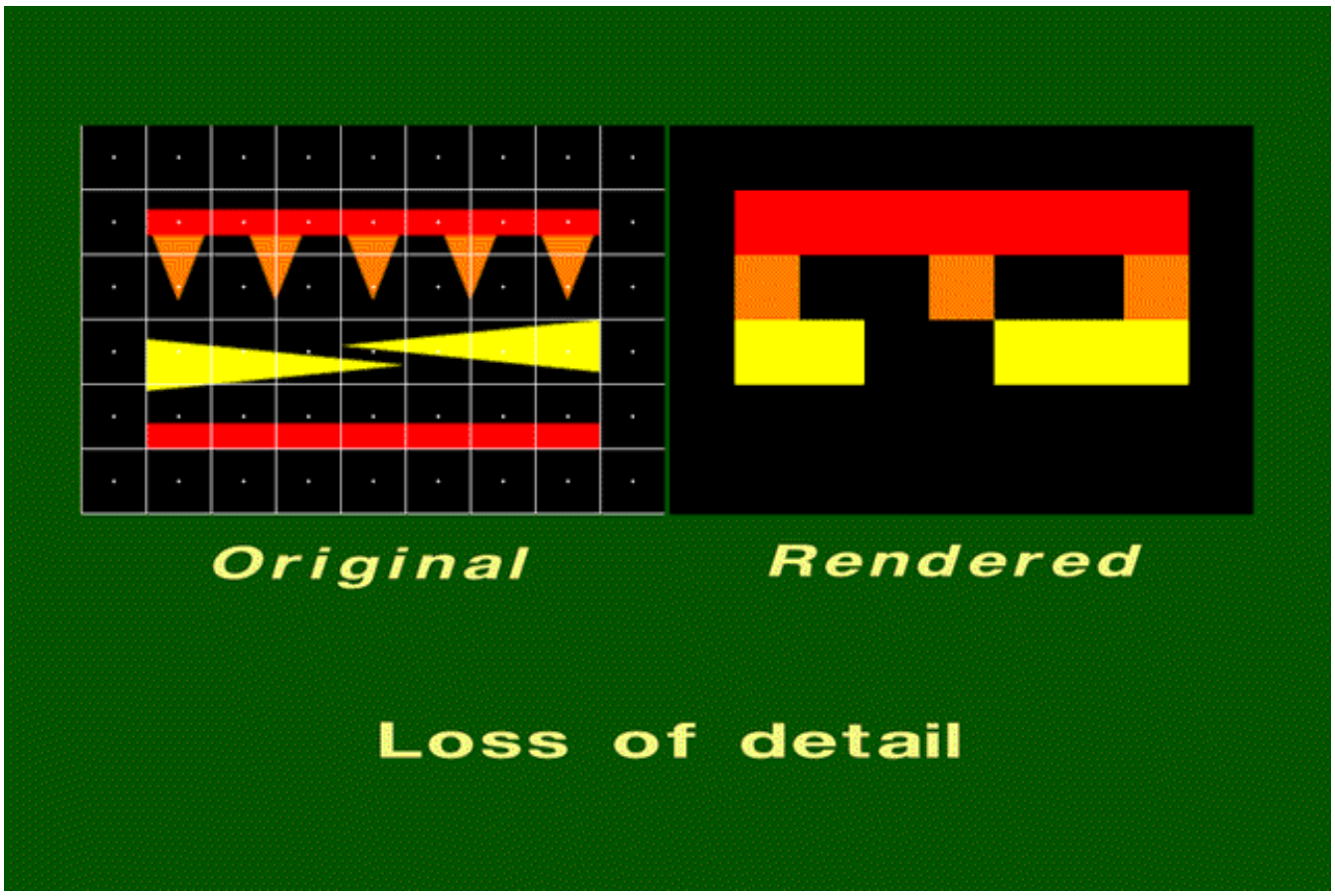
Imagen de github.com

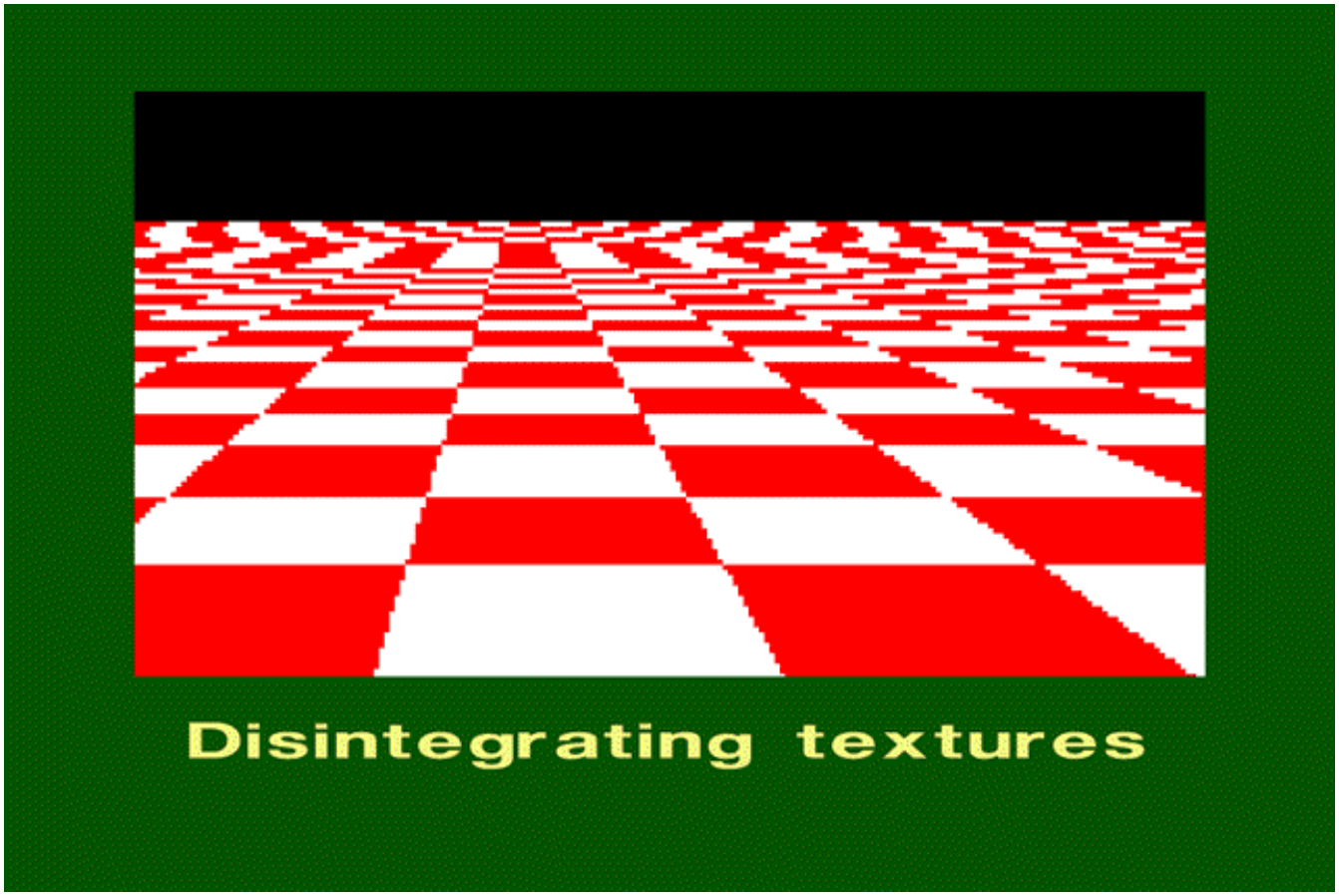
antes de terminar este tema...

Antialiasing

- El **aliasing** es un problema que ocurre cuando una imagen digital es representada con una resolución insuficiente, lo que causa **artefactos visuales** como bordes dentados, detalles desaparecidos o texturas distorsionadas.


Original
Rendered
Jagged profiles

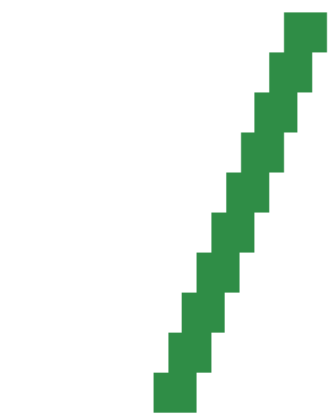

Original
Rendered
Loss of detail



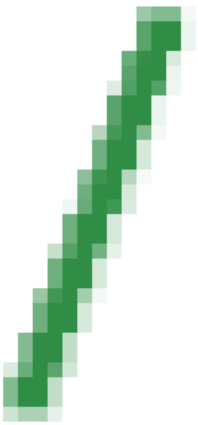
Imágenes de web.cs.wpi.edu

- El **antialiasing** es una técnica que busca reducir el efecto del aliasing aplicando algún tipo de **suavizado o difuminado** sobre la imagen. Existen dos enfoques principales en los algoritmos de renderizado:
  - Algoritmos analíticos: aplican un **prefiltrado** a la imagen antes de muestrearla, eliminando las **frecuencias altas** que causan aliasing. Se basan en principios matemáticos como la **transformada de Fourier**, que permite identificar y reducir los componentes de alta frecuencia. E.g. Supersampling Antialiasing (SSAA).

  - Algoritmos discretos: la imagen se genera directamente en puntos de muestreo regulares (píxeles), sin una función continua previa. Son más simples y fáciles de implementar, pero más propensos al aliasing. E.g. Multisampling Antialiasing (MSAA).



Without Antialiasing



With Antialiasing

Imágenes de electronicshub.org y geeksforgeeks.org

Referencias bibliográficas

- Eck, D. J. (2023). *Introduction to computer graphics* (Version 1.4). Hobart and William Smith Colleges. Recuperado de https://math.hws.edu/graphicsbook
- Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2013). *Computer graphics: Principles and practice*

(3rd ed.). Addison-Wesley.

- Shirley, P. (2020). *Fundamentals of computer graphics* (5th ed.). A K Peters/CRC Press.
- NVIDIA. (2021). *Antialiasing*. NVIDIA Developer Blog. Recuperado de [https://www.nvidia.com/content/Control-Panel-Help/vLatest/en-us/mergedProjects/nv3d/antialiasing.htm)