Corte 1 - Taller Práctico 1:
Múltiples filtros en imagenes

Por: Guillermo Andres De Mendoza Corrales
Universidad Sergio Arboleda, Febrero 2026

- **Fecha de entrega:** Miércoles 25 de Febrero
- **Objetivo del Laboratorio:** Implementar múltiples (5 ejercicios diferentes) filtros de imágenes mediante la tecnología CUDA

Elementos a evaluar con nota académica

- Algoritmos ejecutan correctamente
- Código limpio
- Informe de laboratorio

---

1. Inversión de Colores (Negativo)

La generación del negativo de una imagen es una operación de **procesamiento puntual**. Se llama así porque el nuevo valor de un píxel depende única y exclusivamente de su valor original, sin verse afectado por los píxeles vecinos.

En el modelo digital **RGB**, cada píxel está compuesto por tres canales: Rojo (Red), Verde (Green) y Azul (Blue). Generalmente, cada canal se representa con 8 bits, lo que permite un rango de valores de: [0, 255]

- **0:** Ausencia total de color (Negro).
- **255:** Intensidad máxima del canal (Blanco).

La inversión de color simula el proceso de los antiguos negativos fotográficos. Para obtener el negativo, se debe calcular la "distancia" entre la intensidad actual y el valor máximo posible.

La fórmula matemática para cada canal es una resta simple:

- R_{negativo} = 255 - R_original}
- G_{negativo} = 255 - G_original}
- B_{negativo} = 255 - B_original}

Visualmente, esto transforma los colores claros en oscuros y viceversa. Por ejemplo, un amarillo intenso (255, 255, 0) se convertirá en un azul puro (0, 0, 255).

---

2. Blur de una imagen

El desenfoque de una imagen es una técnica de procesamiento digital de señales que busca suavizar las transiciones de intensidad entre píxeles, eliminando el ruido de alta frecuencia. En términos matemáticos, esto se logra mediante una operación llamada **convolución discreta**.

A diferencia de la escala de grises, donde el nuevo valor de un píxel depende solo de sí mismo, en el desenfoque el valor del píxel de salida P_out(x, y) es el promedio ponderado de sus vecinos.

Para ello se utiliza una **máscara de convolución** (o kernel), que es una matriz pequeña (generalmente de 3x3, 5x o 7x7).

Es la forma más simple de desenfoque. En este caso, todos los píxeles de la vecindad tienen el mismo "peso". Si tenemos un kernel de tamaño NxN, el valor de cada celda de la máscara es 1 / N^2.

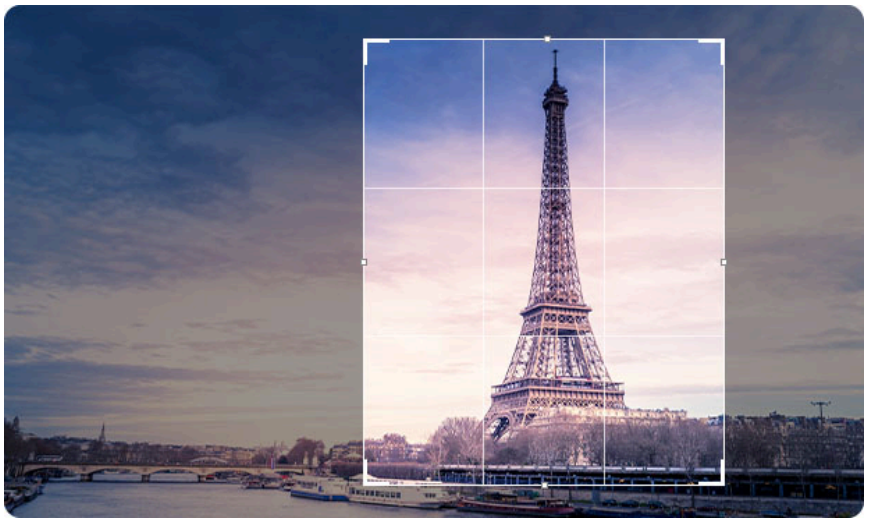
La fórmula para un píxel en la posición (i, j) con un kernel de radio r es:

Ejemplo: Para un filtro de 3 x 3, el radio r = 1. El píxel central se suma con sus 8 vecinos y el resultado se divide por 9.

$$P_{out}(i,j) = \frac{1}{(2r+1)^2} \sum_{m=-r}^{r} \sum_{n=-r}^{r} P_{in}(i+m, j+n)$$

---

3. Recorte de imagen



El recorte de imagen es el proceso de extraer una subregión rectangular de una imagen original. Desde el punto de vista computacional en GPU, consiste en copiar selectivamente datos de una matriz fuente a una matriz destino basándose en un desplazamiento u offset.

1. Definición de la Región de Interés (ROI)

Para definir el recorte, necesitamos cuatro parámetros fundamentales:

- (x_start, y_start): Las coordenadas del píxel superior izquierdo en la imagen original donde inicia el recorte.

- Width_crop (Amplitud en X): El ancho de la nueva imagen.
- Height_crop (Amplitud en Y): El alto de la nueva imagen.

En CUDA, lanzamos una malla de hilos que corresponde al tamaño de la **imagen resultante** (la pequeña). Cada hilo tiene una coordenada (i, j) en el espacio de la imagen recortada. El reto es determinar de qué posición de la **imagen original** debe leer la información.

La relación de mapeo es:

$$P_{dest}(i,j) = P_{src}(i + x_{start}, j + y_{start})$$

Un aspecto crítico en este ejercicio es el **manejo de límites**. Si el usuario pide un recorte que se sale de las dimensiones de la imagen original (por ejemplo, empezar en un “x” muy alto), el kernel podría intentar leer memoria no asignada, provocando un error de segmentación en la GPU o datos basura.

Antes de leer, se debe asegurar que tanto las coordenadas del centro del recorte como las amplitudes de “X” y “Y” no sobrepasen los bordes de la imagen original

---

4. Imagen binaria (blanco y negro)

La binarización es un proceso que transforma una imagen (ya sea a color o en escala de grises) en una imagen donde cada píxel solo puede tener uno de dos valores: 0 (negro) o 255 (blanco).

1. El Concepto de Umbral (Threshold)

El núcleo de este algoritmo es la selección de un valor crítico llamado Umbral (T). Este valor actúa como la frontera de decisión para clasificar los píxeles.

La regla lógica para cada píxel es la siguiente:

- Si la intensidad del píxel es mayor que T, el nuevo valor es 255.
- Si la intensidad del píxel es menor o igual que T, el nuevo valor es 0.

$$P_{out}(i,j) = \begin{cases} 255 & \text{si } P_{in}(i,j) > T \\ 0 & \text{si } P_{in}(i,j) \leq T \end{cases}$$

Binarización directa: Calcular el promedio de los tres canales (R+G+B)/3 y compararlo con T dentro del mismo kernel.

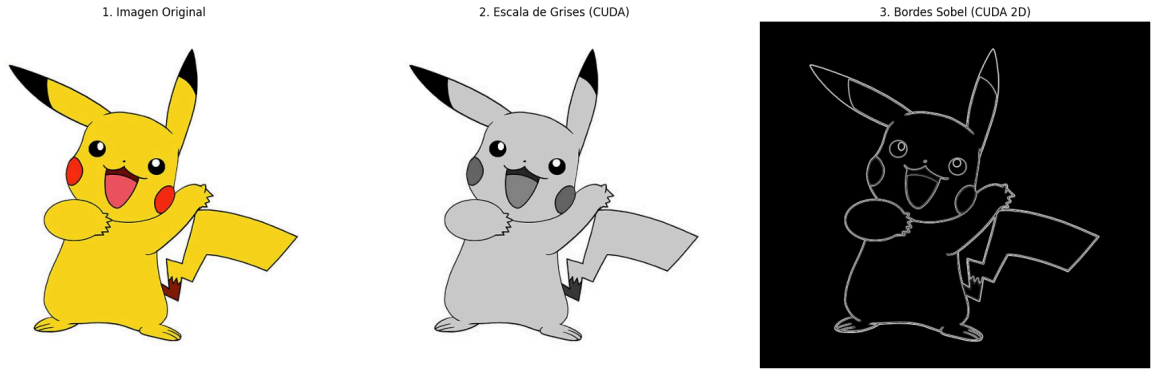
Selección del Umbral

En este taller, el valor de T debe pasarse como un parámetro desde el CPU.

- Un valor típico es **128** (el punto medio del rango 0-255).
- Si el umbral es muy bajo, la imagen se verá mayormente blanca.
- Si el umbral es muy alto, la imagen se verá mayormente negra.

---

5. Detección de Bordes mediante algoritmo Sobel



La detección de bordes es una técnica fundamental en el procesamiento digital de imágenes que busca identificar los puntos de **cambio brusco de intensidad** en una imagen. Estos cambios son esenciales, ya que definen los límites de los objetos.

1.1. El Concepto de Borde como un Gradiente

En matemáticas, el cambio más rápido en una función se describe mediante el **gradiente**. En una imagen (que es una función bidimensional de intensidad I(x, y)), un borde se corresponde con un **alto valor del gradiente**.

El operador Sobel estima la **derivada** de la intensidad de la imagen en dos direcciones principales:

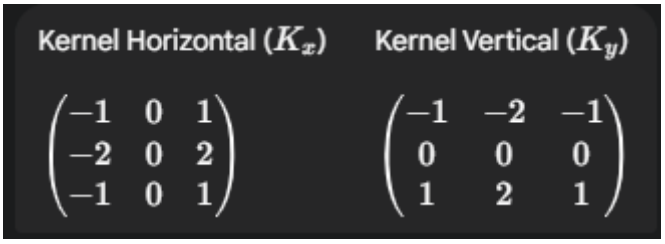
- **Derivada Horizontal (G_x):** Mide los cambios de intensidad en la dirección vertical, detectando **bordes verticales**.
- **Derivada Vertical (G_y):** Mide los cambios de intensidad en la dirección horizontal, detectando **bordes horizontales**.

1.2. La Convolución y el Operador Sobel

Los algoritmos de detección de bordes utilizan la operación de **convolución** (filtro) sobre la imagen. La convolución es la aplicación de una pequeña matriz llamada **núcleo** o **kernel** a cada píxel y sus vecinos.

A. Los Núcleos Sobel 3x3

El operador Sobel utiliza dos kernels 3x3 predefinidos:



Los valores 2 y -2 en el centro de los kernels le dan **más peso** a los píxeles que están justo al lado del píxel central, haciendo el filtro más sensible al cambio inmediato.

B. Cálculo de las Derivadas en el Kernel

El código implementa la convolución para un píxel (i, j) de la siguiente manera:

1. **Cálculo de G_x (Bordes Verticales):** El valor de G_x para el píxel (i, j) se obtiene multiplicando cada píxel vecino en la ventana 3 x 3 de la imagen en escalas de grises por el coeficiente correspondiente en el kernel K_x y sumando los resultados.
2. **Cálculo de G_y (Bordes Horizontales):** Se realiza el mismo proceso, pero utilizando el kernel K_y.

¿Qué Detecta G_x?

Este cálculo de G_x está diseñado para detectar **bordes verticales** (es decir, cambios de intensidad que ocurren en la dirección horizontal):

- **Ponderación Positiva (Columna Derecha):** Los píxeles a la derecha (j+1) tienen un peso positivo (+1 o +2).
- **Ponderación Negativa (Columna Izquierda):** Los píxeles a la izquierda (j-1) tienen un peso negativo (-1 o -2).
- **Ponderación Cero (Columna Central):** Los píxeles en el centro (j) tienen peso cero.

$$G_x = \sum_{m=-1}^{1} \sum_{n=-1}^{1} P_{i+m,j+n} \cdot K_{x,m,n}$$

$$\begin{aligned} G_x &= I_{i-1,j-1}(-1) + I_{i-1,j}(0) + I_{i-1,j+1}(1) \\ &+ I_{i,j-1}(-2) + I_{i,j}(0) + I_{i,j+1}(2) \\ &+ I_{i+1,j-1}(-1) + I_{i+1,j}(0) + I_{i+1,j+1}(1) \end{aligned}$$

Si la intensidad de la imagen cambia bruscamente de **oscura (valores bajos) a clara (valores altos)** al moverse de izquierda a derecha, el resultado de G_x será un **valor positivo grande**. Si cambia de **clara a oscura**, el resultado será un **valor negativo grande**. Este valor grande indica la presencia de un borde vertical fuerte.

¿Qué Detecta G_y?

En este caso es lo mismo que G_x, solo que de forma horizontal

$$G_y = \sum_{m=-1}^{1} \sum_{n=-1}^{1} I(i+m, j+n) \cdot K_y(m,n)$$

$$\begin{aligned} G_y = I_{i-1,j-1}(-1) + I_{i-1,j}(-2) + I_{i-1,j+1}(-1) + \\ I_{i,j-1}(0) + I_{i,j}(0) + I_{i,j+1}(0) + \\ I_{i+1,j-1}(1) + I_{i+1,j}(2) + I_{i+1,j+1}(1) \end{aligned}$$

La Magnitud del Gradiente (El Borde Final)

Los resultados G_x y G_y representan las componentes del gradiente en dos direcciones. Para obtener el valor **final del borde (magnitud)**, que indica la fuerza del cambio de intensidad, se combinan utilizando la fórmula de la norma Euclidiana:

$$\text{Magnitude}(G) = \sqrt{G_x^2 + G_y^2}$$

El valor resultante, asignado a `la imagen resultado`, representa la **fuerza del borde** en ese píxel: un valor alto indica un borde fuerte (como la línea exterior de un objeto), y un valor bajo indica una región suave o uniforme.

Para calcular el gradiente de aproximación sqrt(Gx^2 + Gy^2), utilice el método SQRTF de cuda.libdevice

```
G = cuda.libdevice.sqrtf(Gx**2 + Gy**2)
```

El valor G será el que debemos incorporar en nuestra imagen de salida

```
img_edges[i, j] = G
```