

# DETECCIÓN DE BORDES EN IMÁGENES AÉREAS CON EL FILTRO DE SOBEL Y CUDA

REPORTE DE IMPLEMENTACIÓN  
ARMANDO LEVID RODRÍGUEZ SANTIAGO  
*Tópicos Selectos De Programación*



# ORDEN DE LA PRESENTACIÓN

---

- 1. Introducción
  - Planteamiento del Problema
  - Hipótesis
  - Objetivos
- 2. Marco Teórico
  - 2.1. Extracción de bordes
  - 2.2. Cómputo Heterogéneo con CUDA
- 3. Implementación
  - 3.1. Algoritmo secuencial
  - 3.2. Algoritmo paralelizado
  - 3.3. Filtro de Sobel en OpenCV
  - 3.4. Resultados
- 4. Conclusiones

# INTRODUCCIÓN

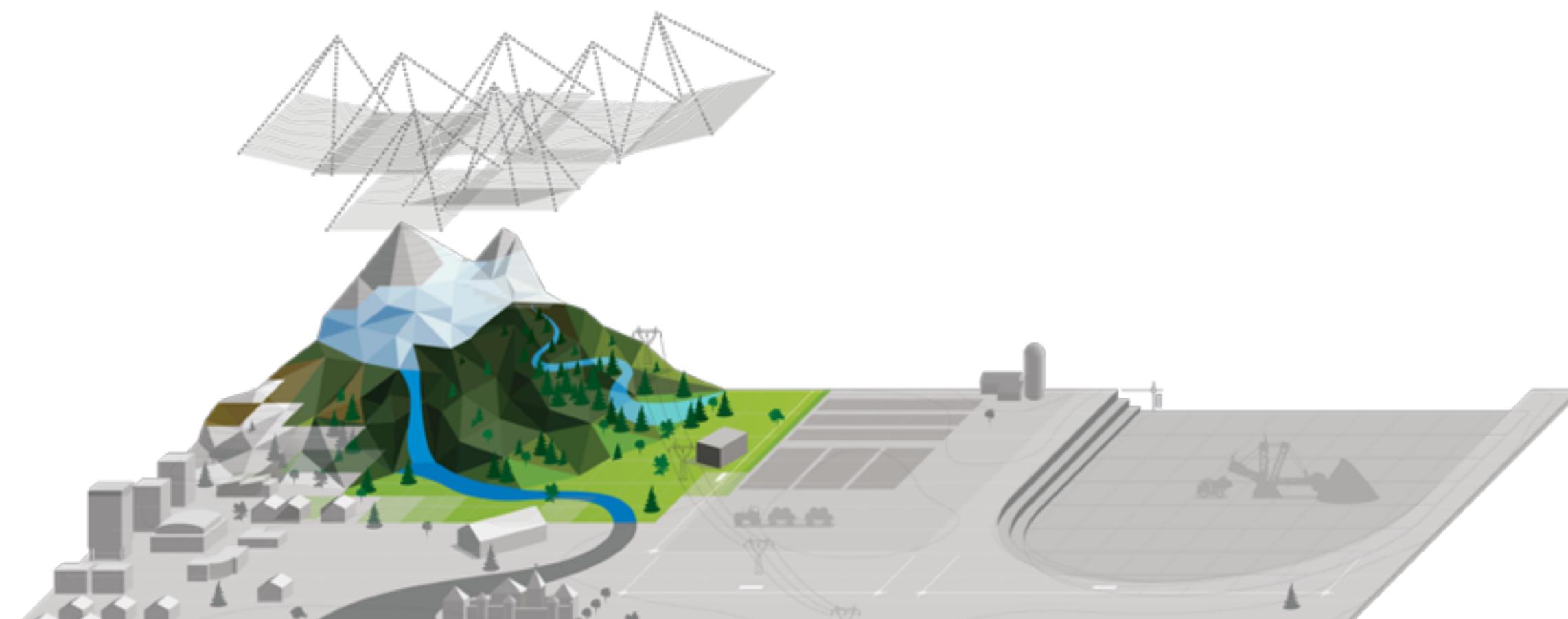
---



# INTRODUCCIÓN

---

- El uso de vehículos aéreos no tripulados UAV (del inglés: Unmanned Aerial Vehicle) mejor conocidos como drones ha ayudado en la solución de diversos problemas.
- Sin embargo, también se han generado nuevos retos para la investigación de manera que puedan realizar nuevas aplicaciones y se resolver algunas otras.
- Por lo que existen diversos algoritmos utilizados para realizar el procesamiento de la información, uno de los más populares es el algoritmo de Sobel.



# PLANTEAMIENTO DEL PROBLEMA

---

- El desarrollo de sistemas que permitan la segmentación, detección o extracción de características de información visual requiere de la implementación de técnicas que permitan la extracción de la información buscada. Un algoritmo suficientemente valido en este tipo de procesos es el filtro de Sobel.
- Sin embargo este es un proceso computacional sumamente costoso en tiempo de ejecución, especialmente cuando se trabaja con imágenes de alta resolución ya que las mascaras son aplicadas a todos los pixeles.
- Por lo tanto, se propone implementar el filtro de Sobel de forma heterogénea de manera que sea posible la reducción del tiempo computacional consumido por el algoritmo para la extracción de bordes en una imagen.



# HIPÓTESIS

---

- Es posible minimizar el tiempo de procesamiento del algoritmo del filtro de Sobel mediante cómputo heterogéneo.



# OBJETIVOS

---

- General:
  - Implementar el filtro de Sobel de forma secuencial y en computo heterogéneo con CUDA C/C++ para comparar los tiempos de ejecución correspondientes a cada implementación.
- Específicos:
  - Implementar en C++ el algoritmo discreto del filtro de Sobel de forma secuencial.
  - Implementar en CUDA C/C++ el algoritmo discreto del filtro de Sobel de forma paralela.
  - Comparar el rendimiento del algoritmo secuencial y paralelo del filtro de Sobel.
  - Determinar el factor de mejora entre los algoritmos secuencial y paralelo del filtro de Sobel.

# MARCO TEÓRICO

---



# EXTRACCIÓN DE BORDES

---

- El operador Sobel es utilizado en el procesamiento digital de imágenes, cuando se desea realizar la detección de bordes.
- Técnicamente es un operador diferencial discreto que calcula una aproximación al gradiente de la función de intensidad de una imagen para cada punto de la imagen a procesar.

$$G[f(x, y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} f(x, y) \\ \frac{\partial}{\partial y} f(x, y) \end{bmatrix} \quad (1)$$

$$|G| = \sqrt{G_x^2 + G_y^2}; \quad \Phi = \tan^{-1} \frac{G_y}{G_x} \quad (2)$$

$$|G| \approx |G_x| + |G_y| \quad (3)$$

# EXTRACCIÓN DE BORDES

---

- El problema de obtener una magnitud de gradiente absoluta apropiada para los bordes, radica en el método utilizado.
- Para dicha tarea existen 2 mascaras de 3x3, una para el gradiente horizontal  $G_x$  y una para el gradiente vertical  $G_y$ , que se utilizan como operadores de Sobel.

Región de la imagen 3x3	Mascara usada para $G_x$	Mascara usada para $G_y$
$\begin{bmatrix} z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 \\ z_7 & z_8 & z_9 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

*Mascaras de Sobel*

# EXTRACCIÓN DE BORDES

---

- A partir de eso las derivadas basadas en los operadores de Sobel son:

$$\begin{aligned} G_x &= (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \\ G_y &= (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \end{aligned} \tag{5}$$

- La magnitud se puede obtener con la expresión (2) o (3).
- Con estas expresiones obtenemos el valor del gradiente en dichos píxeles, para obtener el siguiente valor, las máscaras se mueven al pixel siguiente, es decir, la nueva posición.

# CÓMPUTO HETEROGÉNEO

---

- El modelo de computación con tarjetas gráficas consiste en usar conjuntamente una CPU y una GPU.
- NVIDIA introdujo a sus tarjetas gráficas la arquitectura CUDA (Compute Unified Device Architecture).
- Una nueva arquitectura para el cálculo paralelo de propósito general con soporte para lenguajes de alto nivel y constituidas por cientos de núcleos que pueden procesar de manera concurrente miles de hilos de ejecución.

# CÓMPUTO HETEROGÉNEO - CUDA

---

- Arquitectura de Dispositivos de Cómputo Unificados
- Asume un sistema compuesto por un host (CPU) y un device (GPU)
- `cudaMalloc()`, `cudaMemcpy()`, `cudaFree()`.

Tipo de Transferecia	Sentido de transferecia
<code>cudaMemcpyHostToHost</code>	Host $\Rightarrow$ Host
<code>cudaMemcpyHostToDevice</code>	Host $\Rightarrow$ Device
<code>cudaMemcpyDeviceToHost</code>	Device $\Rightarrow$ Host
<code>cudaMemcpyDeviceToDevice</code>	Device $\Rightarrow$ Device

*Tipos de transferencias de datos en CUDA.*

# CÓMPUTO HETEROGÉNEO - KERNEL

---

- Básicamente es la función que se le asigna un hilo en CUDA.
- La sintaxis es muy similar a C

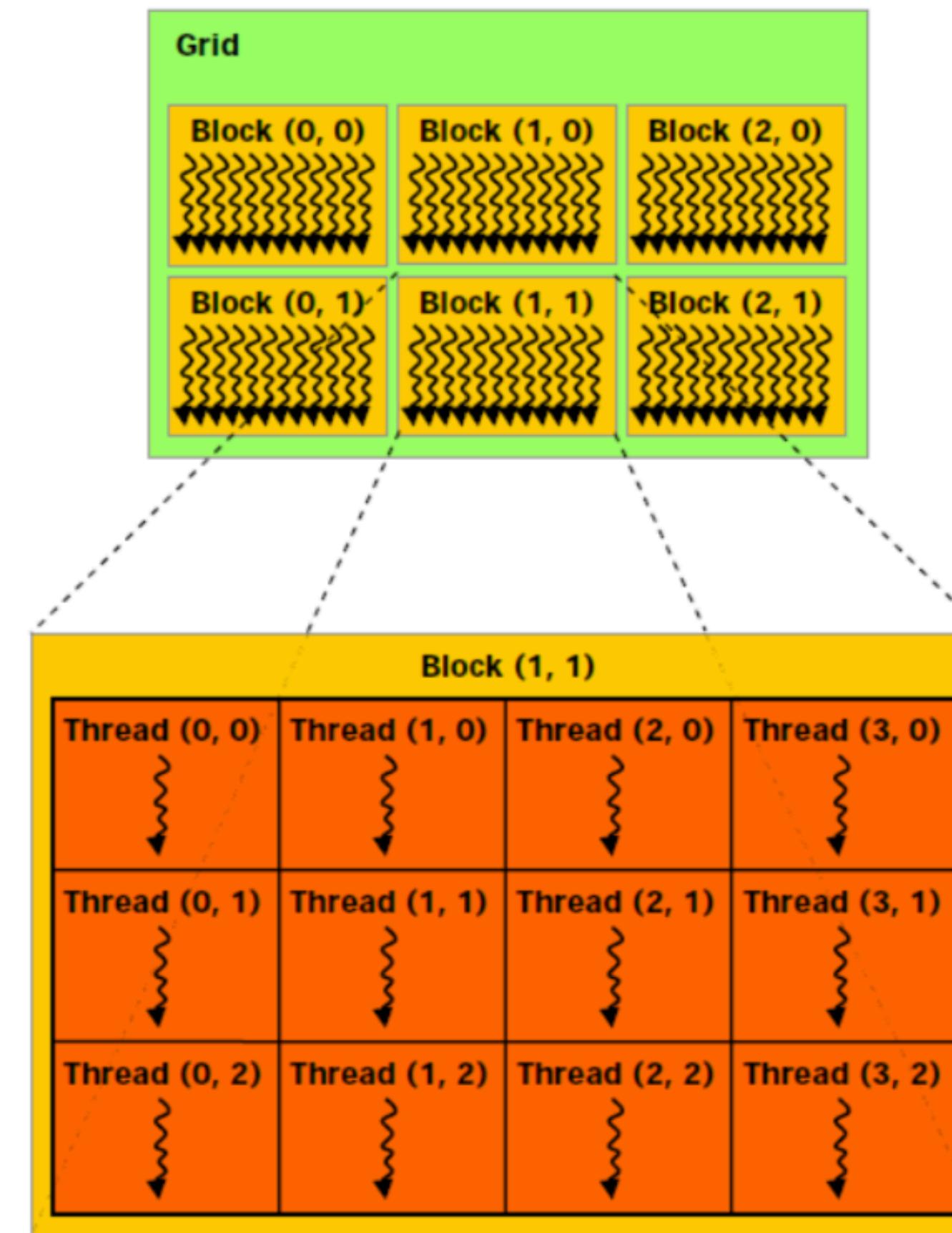
C	CUDA
<pre>1 // Hello World - C 2 void c_hello(){ 3     printf("Hello World!\n"); 4 } 5 int main() { 6     c_hello(); 7     return 0; 8 }</pre>	<pre>1 // Hello World - CUDA 2 __global__ void cuda_hello(){ 3     printf("Hello World from GPU!\n"); 4 } 5 int main() { 6     cuda_hello&lt;&lt;&lt;1,1&gt;&gt;&gt;(); 7     return 0; 8 }</pre>

*Similitud de la sintaxis de C y CUDA*

- Kernel\_name<<< M, T >>>(. . .);
- El lanzamiento del kernel

# CÓMPUTO HETEROGÉNEO - KERNEL

- Los hilos son unidades de cómputo paralelo y cada hilo adopta la función definida por el kernel.



*Organización de hilos, bloques y mallas en CUDA*

# IMPLEMENTACIÓN

---



# IMPLEMENTACIÓN

---

- Implementación y pruebas del algoritmo secuencial y paralelo del filtro de Sobel añadiendo también una implementación con OpenCV.
- Tres tipos de imágenes de diferentes resoluciones. Lena y dos son imágenes aéreas de la UTM.
- CUDA Toolkit 10, Ubuntu 18.04.2 LTS y una tarjeta gráfica NVIDIA Testla T4.
- Todo el procesamiento se realizo utilizando le servicio gratuito de Google Colab.

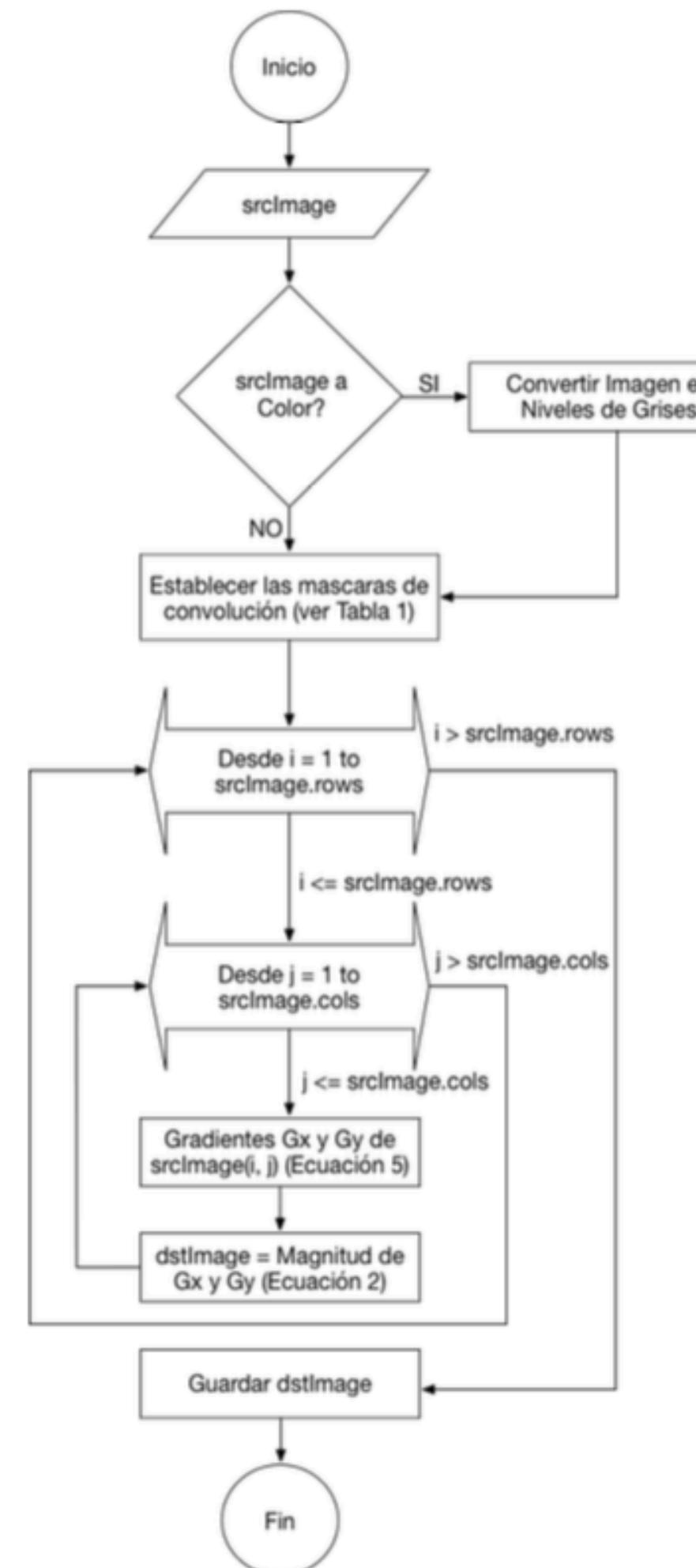


*Google Colab*

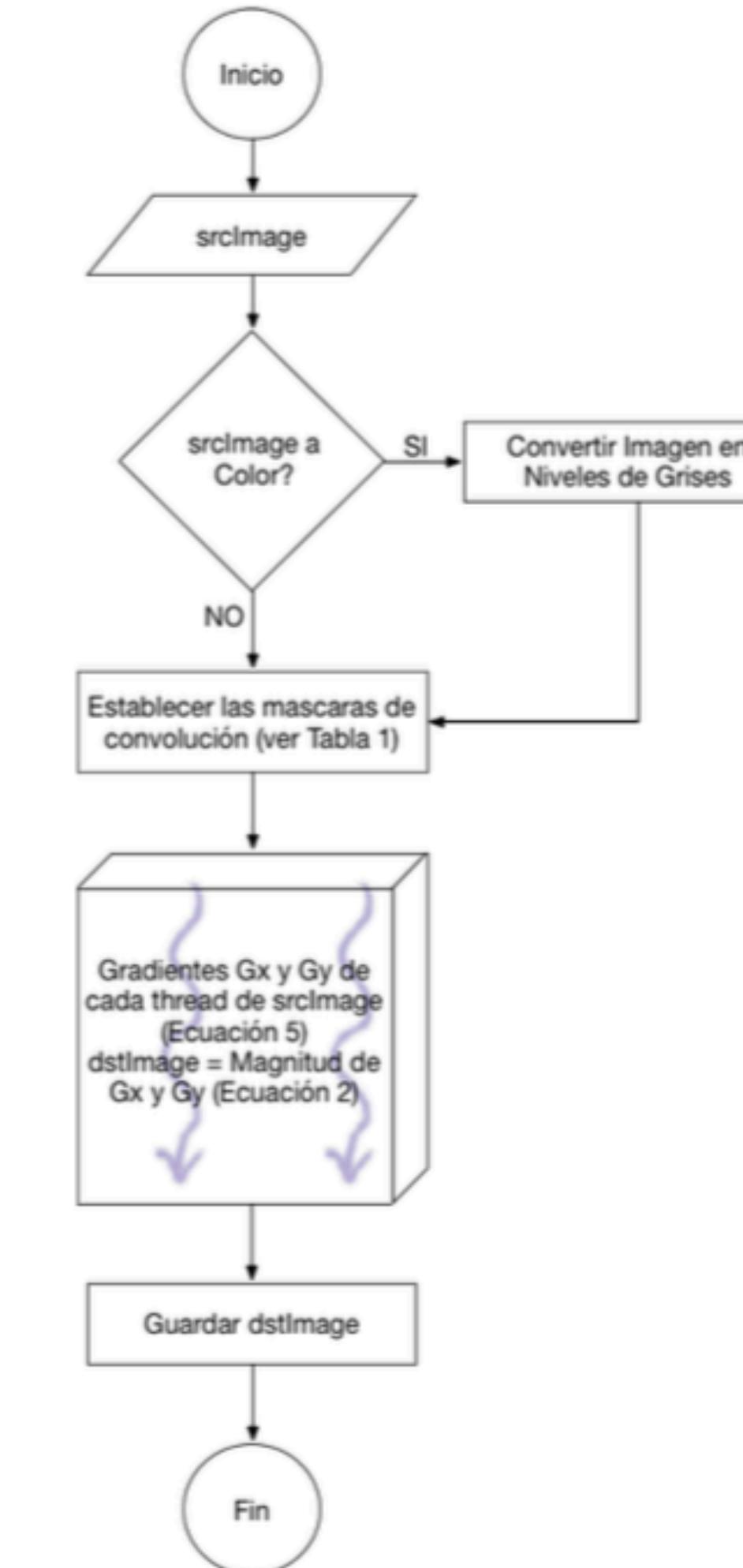


*NVIDIA Tesla T4*

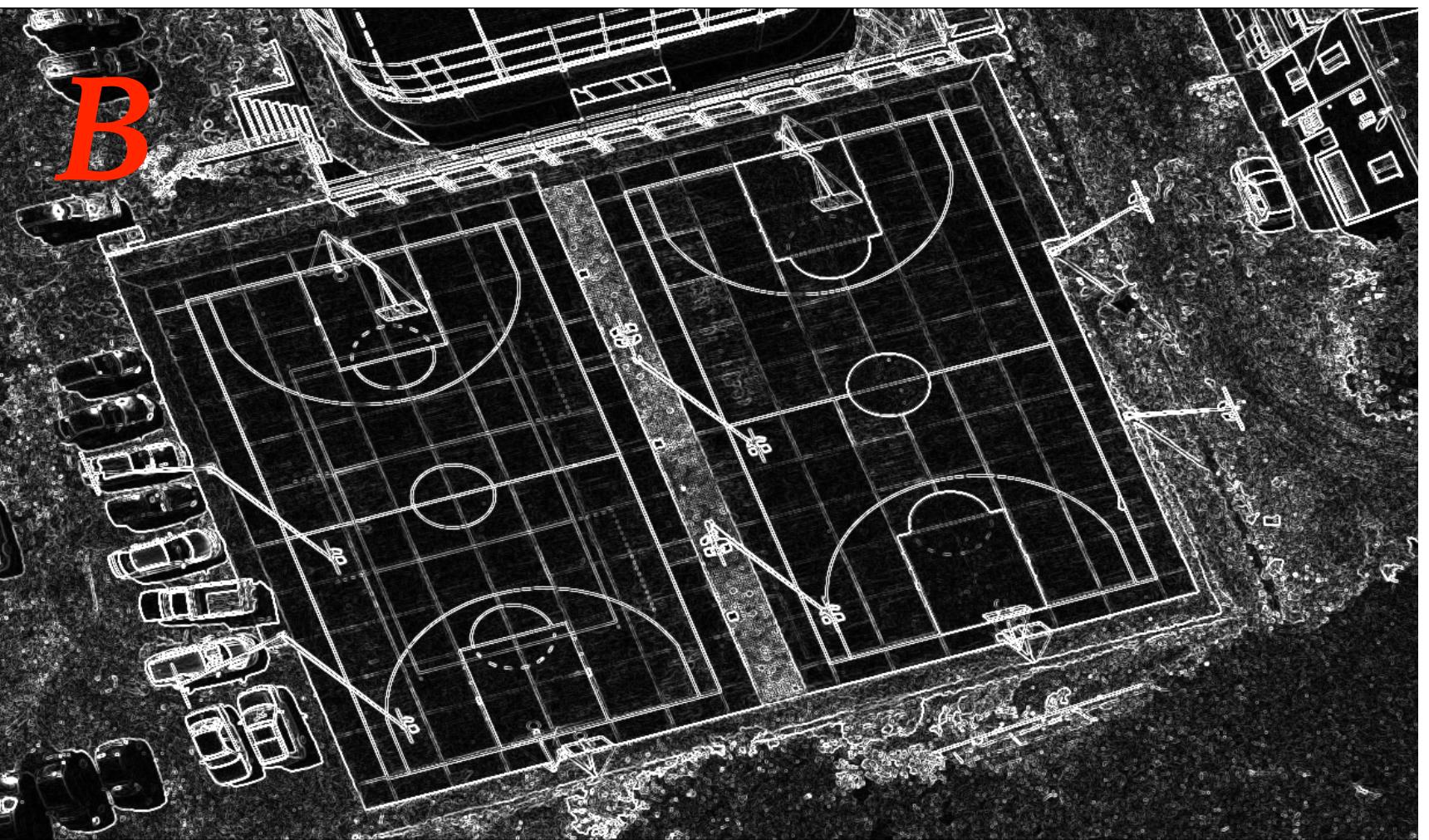
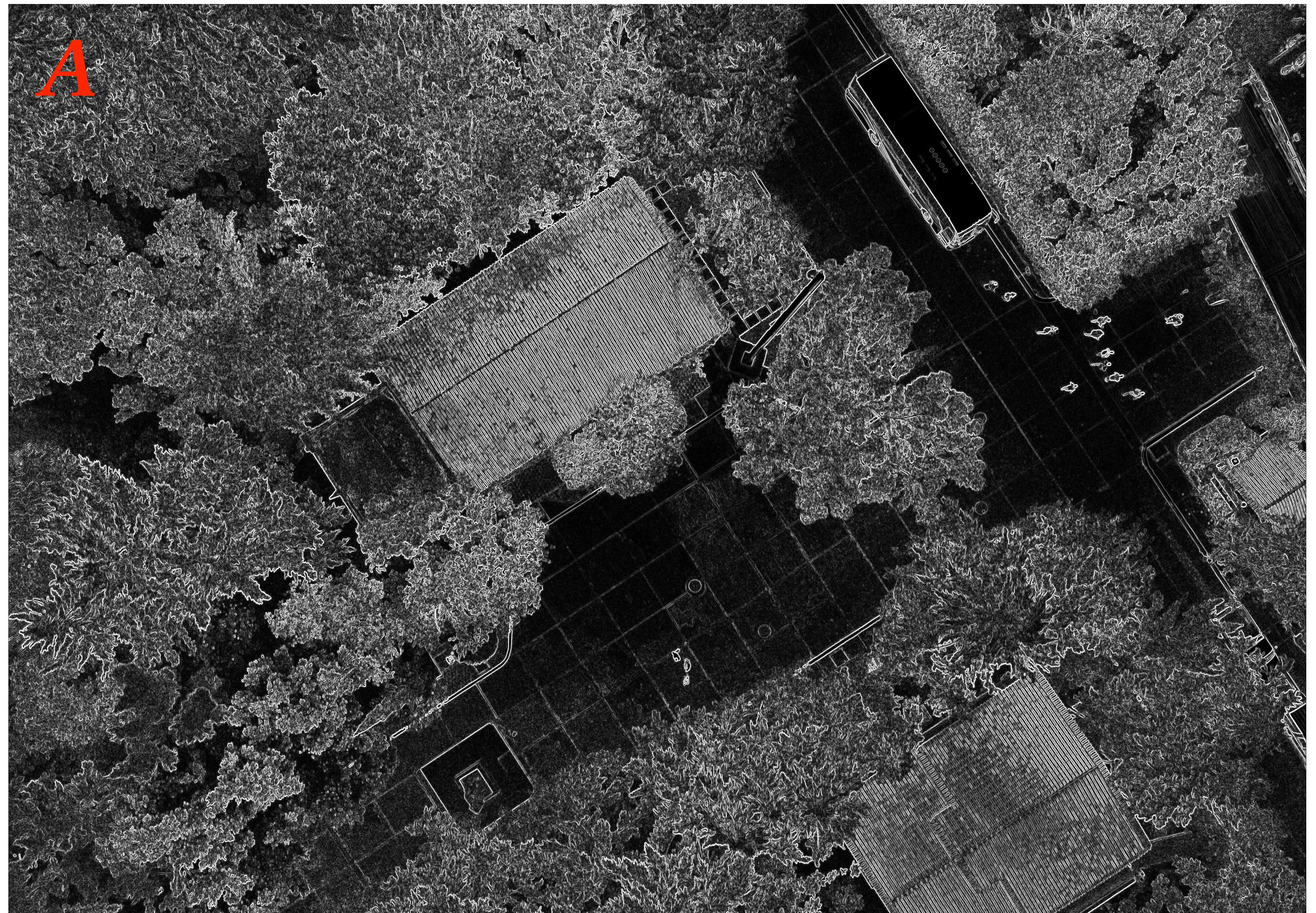
# IMPLEMENTACIÓN - ALGORITMO SECUENCIAL Y CONCURRENTE



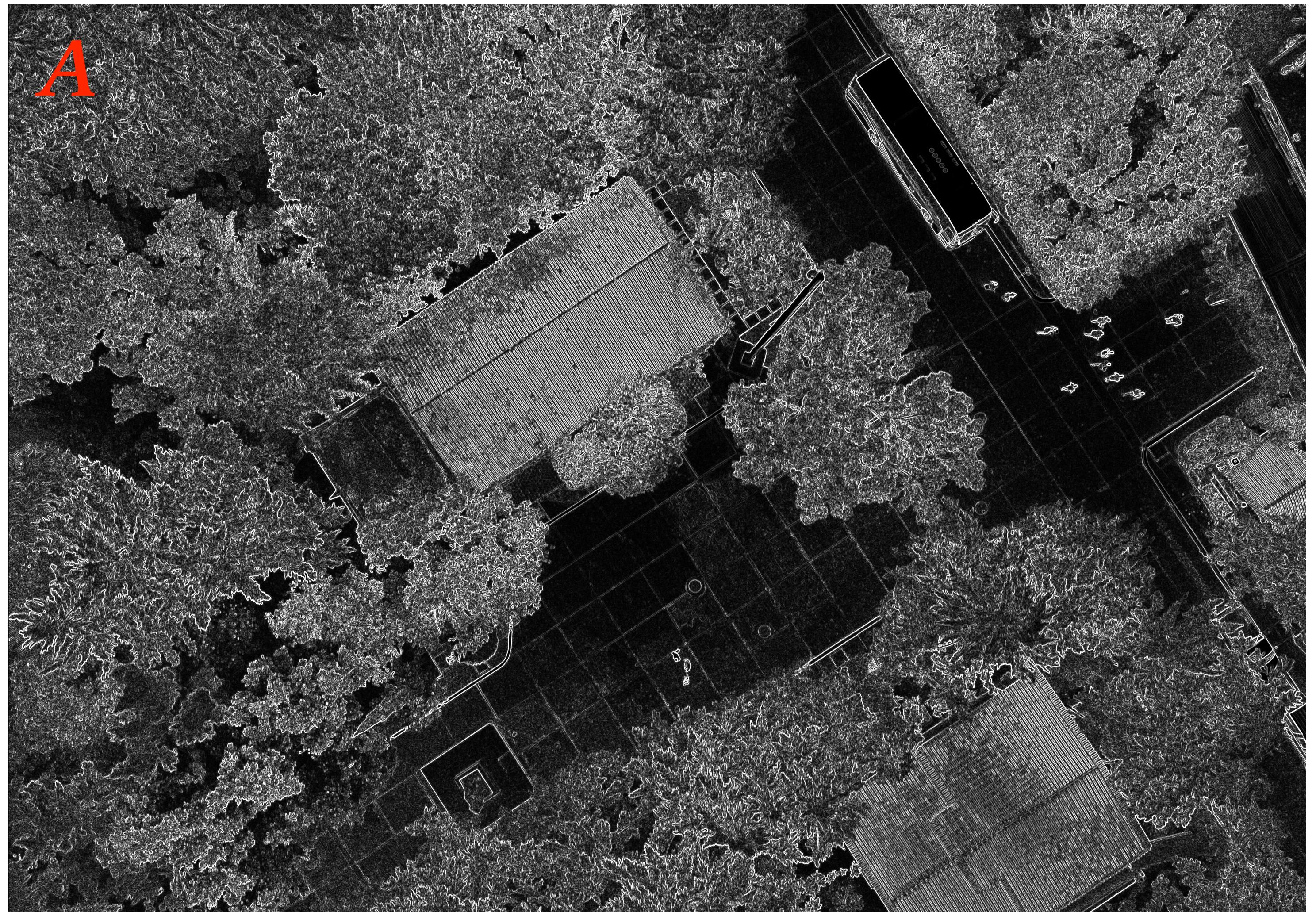
Algoritmo secuencial del filtro de Sobel



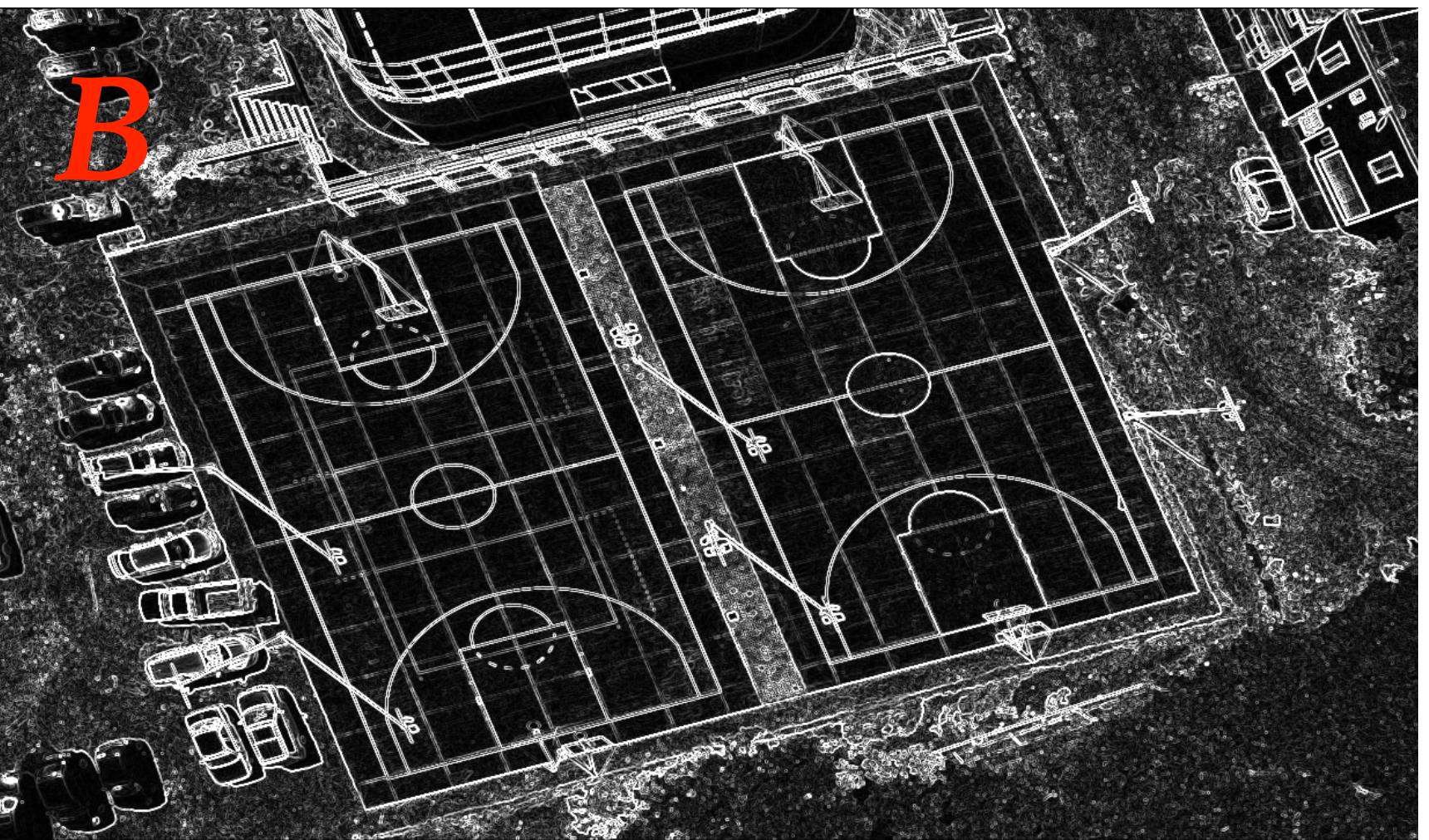
Algoritmo paralelizado del filtro de Sobel



CPU Tiempo promedio de ejecución:  $A = 99.521 \text{ ms}$ ,  $B = 6.796 \text{ ms}$ ,  $C = 2.179 \text{ ms}$



**A**



**B**

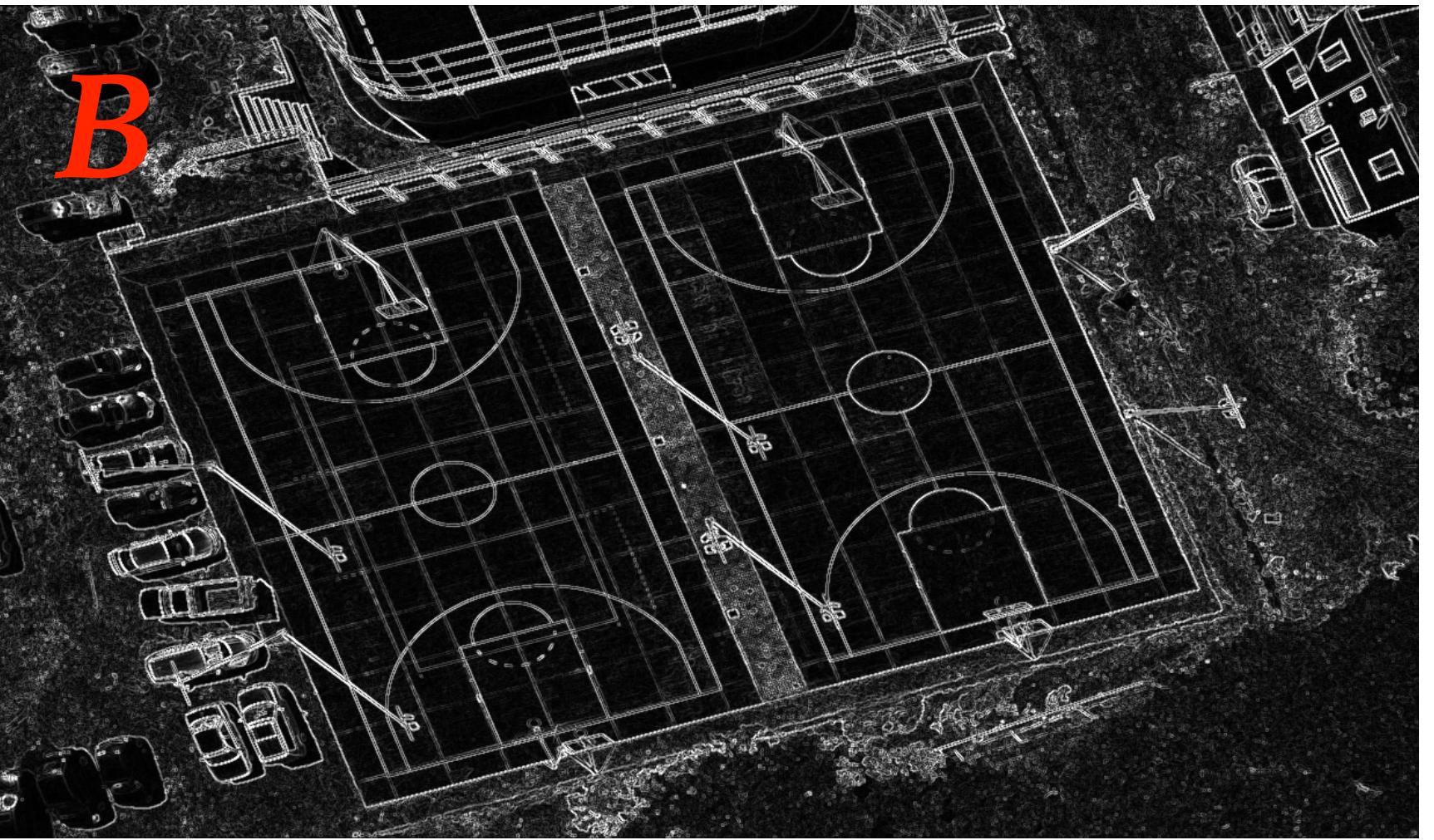


**C**

*GPU Tiempo promedio de ejecución:  $A = 0.889 \text{ ms}$ ,  $B = 0.122 \text{ ms}$ ,  $C = 0.044 \text{ ms}$*



**A**



**B**



**C**

*OpenCV Tiempo promedio de ejecución: A = 54.641 ms, B = 8.746 ms, C = 2.179 ms*

# IMPLEMENTACIÓN - RESULTADOS

---

Imagen	Dimensiones	Dispositivo	Tiempo [ms]	Factor de Mejora
Lena	512 x 512 pixeles	CPU	2.179	49.5
		OpenCV	0.865	19.7
		GPU	0.044	
UTM aérea deportiva	1280 x 720 pixeles	CPU	6.796	55.7
		OpenCV	8.746	71.7
		GPU	0.122	
UTM plaza principal	4000 x 3000 pixeles	CPU	99.521	111.9
		OpenCV	54.641	61.5
		GPU	0.889	

*Comparación de tiempos de procesamiento y mejor respecto a CUDA*

# IMPLEMENTACIÓN - PERFIL DE LA APLICACIÓN

---

Time( % )	Time	Calls	Avg	Min	Max	Name
45.89 %	2.6871ms	1	2.6871ms	2.6871ms	2.6871ms	[CUDA memcpy HtoD]
37.68 %	2.2062ms	1	2.2062ms	2.2062ms	2.2062ms	[CUDA memcpy DtoH]
16.40 %	960.38us	1	960.38us	960.38us	960.38us	sobelFilterGPU()
0.02 %	1.2160us	1	1.2160us	1.2160us	1.2160us	[CUDA memset]

*Perfil de la aplicación*

# CONCLUSIONES

---



# CONCLUSIONES

---

- Se ha implementado el filtro de Sobel para la detección de bordes en imágenes aéreas de alta resolución.
- El algoritmo se ha implementado de tres diferentes formas, secuencial, heterogénea y con OpenCV. Con resultados suficientemente buenos en la extracción de bordes de las imágenes de ejemplo.
- Los tiempos de procesamiento con CUDA quedo demostrado que son reducidos considerablemente.
- Se ha obtenido un factor de mejora de hasta mas de 100 veces mejor con respecto a un procesamiento en CPU y poco más de 70 veces mejor que una aplicación con OpenCV.
- La mejora en cuanto al tiempo de procesamiento es evidente, con lo cual la hipótesis plateada queda demostrada de forma afirmativa.

# CONCLUSIONES

---

- En trabajos futuros, implementar el calculo de los gradientes en kernels independientes y con ayuda de los streams de CUDA se espera el tiempo de procesamiento pueda disminuir aun más.
- De igual forma se plantea la integración de este algoritmo en uno dedicado a la extracción de características.
- Este tipo de programación a sistemas más complejos como los son las redes neuronales profundas, donde el tiempo de procesamiento juega un papel muy importantes y este tipo de optimizaciones son requeridas.

# REFERENCIAS

---

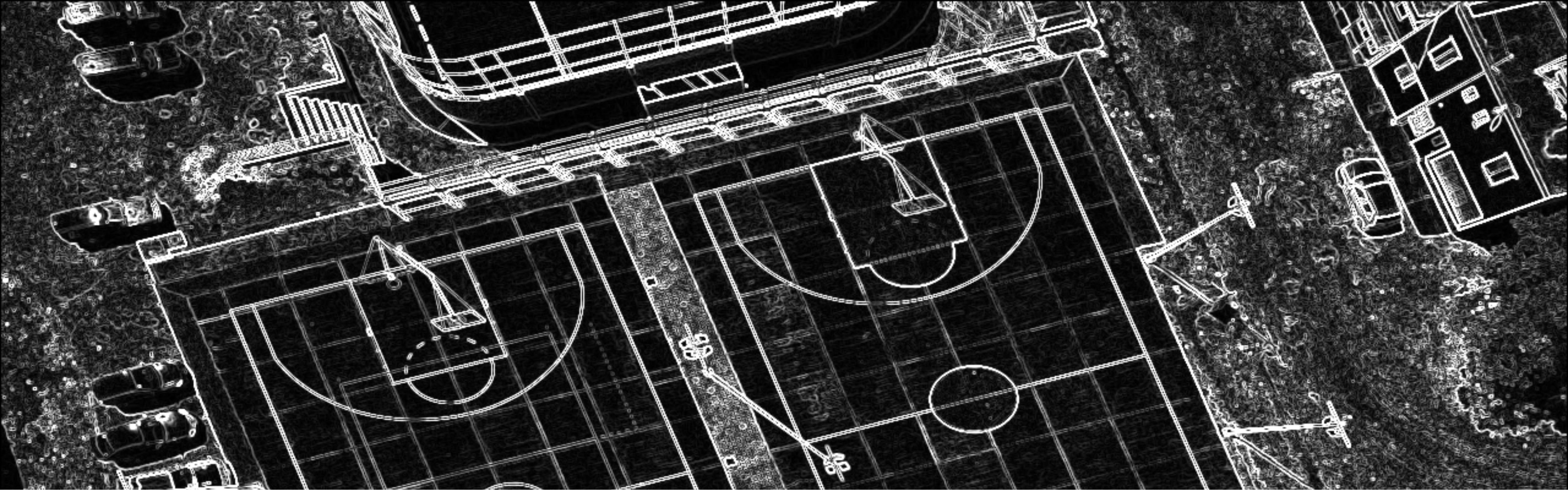




## REFERENCIAS

---

- [Harris, 2017] Harris, B. M. (2017). An even easier introduction to cuda. Online: [<https://devblogs.nvidia.com/even-easier-introduction-cuda>].
- [Represa Pérez et al., ] Represa Pérez, C., Cámera Nebreda, J., and Sánchez Ortega, P. Introducción a la programación en cuda, universidad de burgos, área de tecnología electrónica (2016).
- [Google, 2019] Google (2019). Google colaboratory. Online: [<https://colab.research.google.com/notebooks/welcome.ipynb>].
- Tech, A. (2019). Online: [<https://www.youtube.com/watch?v=n7RdjB9bDKo>].



**GRACIAS POR SU ATENCIÓN**

[https://github.com/LevidRodriguez/Sobel\\_with\\_OpenCV-CUDA.git](https://github.com/LevidRodriguez/Sobel_with_OpenCV-CUDA.git)