



TRABAJO FIN DE GRADO

**Grado en Ingeniería Informática (Mención
Ingeniería de Computadores)**

**SISTEMA DE RECONOCIMIENTO DE
MICROFIBRAS MEDIANTE PROGRAMACIÓN
PARALELA EN OPENCL**

Autor: José Sánchez Yun

Directores: Ezequiel Herruzo Gómez
Jesús Rioja Bravo

Índice General

I Punto de Partida	10
1. Introducción	11
1.1 Paralelización y sus métodos	13
1.2 Procesamiento de imágenes	16
1.3 Tecnología y medio ambiente	17
2. Definición del problema	19
2.1 Definición del problema real	19
2.2 Definición del problema técnico	20
3. Objetivos	22
4. Antecedentes	25
4.1 Procesamiento de imagen	25
4.2 Unidades de procesamiento gráfico (GPUs)	28
4.3 Algoritmos de detección de bordes	29
4.4 OpenCV	30
4.5 OpenCl	31

II Restricciones y especificaciones	33
5. Restricciones	34
5.1 Factores Dato	35
5.2 Factores Estratégicos	36
5.2.1 OpenCV	37
5.2.2 Nvidia CUDA y OpenCl	38
5.2.3 Python	39
5.2.4 Algoritmo de Canny	40
5.2.5 Algoritmo identificador de fondo	42
5.2.6 Algoritmo identificador de borde	43
6. Recursos	45
6.1 Recursos hardware	46
6.2 Recursos software	47
6.3 Recursos humanos	48
7. Especificación de requisitos	49
7.1 Requisitos de Usuario	50
7.2 Requisitos Funcionales	51
7.3 Requisitos no funcionales	52

7.4 Requisitos de información	53
III Desarrollo	54
8. Análisis Funcional	55
8.1 Código Host	57
8.2 Código Kernel	59
9. Sistemas Hardware	62
9.1 Arquitectura del procesador	62
9.2 Arquitectura de la GPU	65
10. Sistemas software	70
10.1 Preparación del entorno de desarrollo	70
10.2 PyOpenCl	73
10.2.1 Pasos a realizar en un programa OpenCl	77
10.3 OpenCV	82
10.4 Tkinter	83
11. Desarrollo del sistema. Preparación del algoritmo	84
11.1 Código host	85
11.2 Código kernel	89

12. Pruebas	91
12.1 Pruebas de funcionamiento	92
12.1.1 Pruebas de verificación	92
12.1.2 Pruebas de validación	95
12.2 Pruebas de tiempo	108
13. Conclusiones	112
13.1 Conclusiones sobre los objetivos propuestos	112
13.2 Conceptos aprendidos	115
13.3 Conclusión general	116
13.4 Futuras mejoras	118
Bibliografía	119
IV Anexos	125
Anexo I: Manual de Código (host y kernel)	126
Código principal (host)	127
Código kernel	134
Anexo II: Implementación de Canny	136
Código de implementación de Canny	137
Anexo III: Manual de Usuario	139

Instalación	140
Uso	141
Anexo IV: Aplicación de escritorio, código de interfaz	143
Código de interfaz	144

Índice de Figuras

Figura 4.1: 4-adyacencia en imágenes digitales	27
Figura 4.2: 8-adyacencia en imágenes digitales	27
Figura 5.2: Prueba con algoritmo de Canny (imagen original)	41
Figura 5.3: Prueba con algoritmo de Canny (imagen de contornos)	41
Figura 8.1: Diagrama de Flujo (Código Host)	57
Figura 8.2: Diagrama de Flujo (Código Kernel)	59
Figura 8.3: Diagrama de Flujo (Calcula btm)	60
Figura 9.1: Arquitectura CPU	64
Figura 9.2: Núcleo GPU	66
Figura 9.3: Streaming multiprocessor	67
Figura 9.4: TPC GPU en arquitectura Pascal	68
Figura 10.1: Prueba de instalación de Python	71
Figura 10.2: Esquema OpenCl	73
Figura 10.3: Secuencia de un programa OpenCl	75
Figura 10.4: logo de OpenCV	82
Figura 10.5: logo de Tkinter	83

Figura 11.1: Esquema de funcionamiento del algoritmo	89
Figura 12.1: Primera prueba (imagen original)	96
Figura 12.2: Primera prueba (imagen resultado)	97
Figura 12.3: Primera prueba (imagen de contornos)	97
Figura 12.4: Primera prueba (resultado de contornos)	98
Figura 12.5: Segunda prueba (imagen original)	99
Figura 12.6: Segunda prueba (imagen resultado)	100
Figura 12.7: Segunda prueba (imagen de contornos)	101
Figura 12.8: Segunda prueba (resultado de contornos)	101
Figura 12.9: Tercera prueba (imagen original)	102
Figura 12.10: Tercera prueba (imagen resultado)	103
Figura 12.11: Tercera prueba (imagen de contornos)	104
Figura 12.12: Tercera prueba (resultado de contornos)	105
Figura 12.13: Resultado del programa vs real	106
Figura 12.14: Resultados Algoritmo de Canny vs real	107
Figura 12.15: Imagen para prueba de tiempos	108
Figura 12.16: Resultados tiempo paralelo vs secuencial	110
Figura I.1: Interfaz del Menú principal.	136

Índice de Cuadros

Cuadro 6.1: Recursos software	46
Cuadro 6.2: Recursos hardware	47
Cuadro 6.3: Recursos humanos	48
Cuadro 7.1: Requisitos de usuario	50
Cuadro 7.2: Requisitos Funcionales	51
Cuadro 7.3: Requisitos no funcionales	52
Cuadro 7.4: Requisitos de información	53

Parte I

Punto de Partida

Capítulo 1

Introducción

Uno de los objetivos fundamentales que se persigue en la informática a día de hoy es la optimización, o mejor dicho los métodos para determinar los valores de las variables que intervienen en un proceso o sistema para que el resultado sea el mejor posible. Conforme la tecnología ha ido avanzando también lo ha hecho su complejidad y por tanto el factor de la optimización cada vez es más importante puesto que los sistemas tienen que soportar software más complejo en un tiempo de respuesta aceptable.

Existen 6 niveles con respecto a la optimización de software [14]:

- Nivel de diseño: el diseño arquitectónico de un sistema mayoritariamente afecta a su rendimiento.
- Nivel de código fuente: persigue evitar la codificación de mala calidad.
- Nivel de armado: actúa entre el código fuente y el compilador.
- Nivel de compilación: se hace uso de un compilador optimizador para asegurar la eficiencia del programa.
- Nivel de ensamblador: para determinadas plataformas de hardware programar en lenguaje ensamblador puede aumentar el rendimiento del programa.

- Nivel de tiempo de ejecución: con el que los programadores de ensamblador y los llamados compiladores just-in-time pueden realizar la optimización del tiempo de ejecución.

Existen campos de la programación en los que es más importante la optimización debido a que conllevan una gran carga computacional, esto hace que sea necesario conocer con gran profundidad tanto el hardware en el que se está ejecutando como los métodos y protocolos para una correcta codificación. Uno de estos campos es el procesamiento de imagen, en el que a menudo los programas constan de una gran carga computacional debido al procesamiento individual de cada píxel y que a menudo supone un gran reto la creación de programas relacionados con este campo que funcionen de una manera eficaz y con un tiempo de respuesta aceptable.

Una de las técnicas más importantes a la hora de trabajar con estructuras de datos con una gran carga matemática es la programación paralela, el paralelismo se basa en un principio muy sencillo, dividir un gran problema en varios de menor tamaño y resolverlos a la vez, lo que permite ejecutar más instrucciones en un menor periodo de tiempo. Por tanto la principal diferencia entre la programación paralela y la secuencial es que en la primera podemos procesar varias operaciones de manera simultánea mientras que en la programación secuencial el programa ejecuta una sola orden en un mismo periodo de tiempo.

1.1. Paralelización y sus métodos.

Hoy en día todos los computadores son esencialmente paralelos [15], esto significa que dentro de cada ordenador siempre existe un conjunto de operaciones que de una manera u otra se ejecutan de manera simultánea, estas operaciones pueden ser para un propósito distinto entre sí o pueden ser programadas para un fin común que se complete una vez todas hayan finalizado.

Hay dos razones principales por las que se necesita que los ordenadores modernos sean paralelos, la primera porque no es posible que la frecuencia de procesador y de memoria mejoren indefinidamente por lo que para mejorar la potencia se necesitan nuevos conceptos de organización y estructura, la segunda es que el consumo de potencia aumenta conforme lo hace también la frecuencia del procesador, mientras que la eficiencia de la energía disminuye, por lo que supone una gran ventaja el poder trabajar a una velocidad menor del procesador pero ejecutando varias operaciones al mismo tiempo.

En sus comienzos, el paralelismo se encontraba mayormente en supercomputadores dedicados a resolver ciertos problemas científicos, pero hoy en día la paralelización se ha extendido al gran mercado y a la gran mayoría de dispositivos, por lo que podemos distinguir tres tipos principales de paralelización:

- Sistemas de memoria compartida: sistemas con múltiples unidades de procesamiento adjuntos a una sola memoria.
- Sistemas distribuidos: consisten en varios computadores con su propia unidad de procesamiento y su memoria que están conectados mediante redes con alta velocidad de conexión.
- Sistemas de procesamiento gráfico: son utilizados como complemento del procesador para resolver problemas de propósito general con estructuras que requieran gran carga computacional.

A lo largo de este trabajo nos centraremos en el tercer tipo de paralelización, mediante el uso de la programación paralela en GPU se resolverá un problema que conlleva procesamiento de imágenes utilizando las herramientas OpenCl y OpenCV.

1.2. Procesamiento de imágenes

Antes de entrar en términos de procesamiento [16] se debe definir el concepto de imagen, siendo esta una representación de unas dimensiones de altura y anchura determinadas por un número de píxeles. Un píxel es un punto de la imagen con una sombra, opacidad y color específicos, estos pueden ser representados en diferentes formatos, los principales son los siguientes:

-RGB: cada píxel se compone de 3 números enteros desde el 0 hasta el 255, cada uno de ellos representa un color primario (rojo, verde y azul).

-Escala de grises: cada píxel se compone de un solo número entero desde el 0 hasta el 255 que va desde el color negro (0) hasta el color blanco (255) pasando por todas las tonalidades de gris que se encuentran entre ellos.

-RGBA: es una extensión del RGB en la que se añade un campo llamado Alpha que representa la opacidad de la imagen, por lo que cada píxel es representado por 4 números enteros.

-HSV: cada píxel se compone de 3 números enteros desde el 0 hasta el 255, uno de ellos representa el matiz, otro la saturación y otro el valor en el eje blanco-negro.

El procesamiento de imagen es un método que realiza ciertas operaciones en una imagen con el objetivo de mejorar su calidad o de extraer cierta información de la misma para su uso posterior. Como podemos observar viendo las diferentes formas de representar una imagen mencionadas anteriormente, hay una gran cantidad de parámetros que se pueden tener en cuenta a la hora de definir una imagen lo que nos da un campo muy amplio a la hora de modificar o de extraer información sobre ciertos aspectos de la misma.

Existen numerosas librerías de programación que contienen funciones orientadas al tratamiento de imágenes, durante este trabajo se utilizará una de ellas llamada OpenCV la cuál es una de las más extendidas a nivel mundial y que contiene un gran número de herramientas que facilitan la realización de este tipo de tareas.

1.3. Tecnología y medio ambiente.

Con la evolución de la tecnología cada vez se ha ido ampliando el número de campos en los que esta se aplica [17], a tal nivel que la tenemos en casi todos los ámbitos de nuestra vida cotidiana y también ayuda en aspectos como en la medicina, investigación o en el que nos centraremos, el medio ambiente.

La tecnología permite mayores conocimientos técnicos y científicos del medio ambiente contribuyendo a diseñar y crear bienes que favorecen el estado de conservación del medio. Esto incluye el desarrollo de nuevas formas de energía, medios de transporte con combustibles más respetuosos, sistemas que puedan controlar el uso de la energía o sistemas que adquieran información sobre los niveles de cierto material en un medio determinado.

Uno de los temas más candentes con respecto al medio ambiente hoy en día es la contaminación de los mares, lo cuál afecta directamente a la vida silvestre de los hábitats oceánicos e indirectamente a la salud humana, estos problemas incluyen el derrame de petróleo y otros residuos tóxicos o la acumulación de plásticos y otro tipo de fibras contaminantes en los mares.

El desarrollo de este trabajo se centrará en el último de los mencionados. El programa consistirá en el análisis e identificación de microfibras dada una muestra de agua observada mediante microscopio, esto incluye también los dos temas tratados anteriormente, se tratará de un programa que procesa una imagen de muestra de agua mediante la implementación de un algoritmo que utiliza la programación paralela. Con esto se identificarán cierto tipo de microfibras incluyendo los microplásticos y fibras naturales entre otros.

Capítulo 2

Definición del problema

Durante este capítulo se procederá a explicar de manera detallada el problema que se pretende solucionar mediante la realización del proyecto, dicha definición tendrá dos enfoques distintos, el primero es visto desde el punto de vista del problema real que se pretende solucionar, es decir, planteado más desde la vista del cliente que propone un problema sin entrar en ningún detalle técnico de cuál será su solución. El segundo enfoque será desde un punto de vista técnico con todas las necesidades que precisa el proyecto para llegar a una solución válida y eficiente.

2.1. Definición del problema real

Como se ha mencionado anteriormente la tecnología es utilizada para una gran cantidad de propósitos incluido en el de ayuda en el medio ambiente, cubriendo campos desde las energías renovables hasta el control del ecosistema.

El problema que se nos plantea en este proyecto es el de desarrollar un programa que analice una muestra de agua tomada con microscopio y sea capaz de identificar diferentes partículas dentro de la muestra así como su número, con la intención final de llevar un

registro de la cantidad de determinada partícula, como pueden ser los microplásticos en una zona determinada de agua. Esto puede ayudar al análisis y control de zonas acuosas permitiendo aplicar métodos de actuación y prevención que ayuden a tener un ambiente sostenible. El registro deberá incluir también el tamaño de cada partícula identificada a partir del tamaño de la placa de muestras.

2.2. Definición del problema técnico

Si enfocamos el problema desde esta perspectiva el reto principal que se presenta es el de cómo identificar y diferenciar de manera correcta las diferentes partículas que se pretende, mediante el análisis de numerosas muestras se ha llegado a la conclusión de que una característica distintiva entre las partículas observadas es el borde, teniendo unas un borde más suave, como pueden ser las fibras naturales mientras que otras presentan uno más escarpado, como pueden ser los microplásticos.

Teniendo esto se nos plantea un problema de procesamiento de imagen en el que tenemos que analizar ciertos tipos de borde para identificar cada tipo de partícula. Tras el análisis de varias formas de afrontación del problema se ha decidido implementar un algoritmo que analiza cada píxel de la imagen y lo compara con sus adyacentes uno por uno para comprobar si hay una diferencia significativa entre él y el píxel adyacente en cuestión, en el caso de que si la hubiera, estaríamos ante el comienzo del borde de algún elemento presente en la imagen. Para diferenciar los distintos tipos de borde se introducirá un nuevo concepto llamado borde de tipo de material (o btm) que se

define como el número de píxeles necesarios para representar el grosor del borde de un elemento de la imagen.

Este algoritmo se implementará mediante el uso de programación paralela en GPU, concretamente con OpenCl, lo que como se ha mencionado anteriormente ayudará a mejorar la eficiencia del programa y hará que se obtenga un tiempo de ejecución aceptable para solucionar el problema planteado.

Tanto el planteamiento del algoritmo como los conceptos mencionados serán desarrollados en mayor profundidad en los próximos capítulos del proyecto.

Capítulo 3

Objetivos

Como se ha mencionado anteriormente, con este proyecto se pretende desarrollar un programa que sea capaz de identificar y diferenciar diferentes tipos de partículas, en los que se encuentran los microplásticos y microfibras naturales a partir de imágenes de una muestra de agua tomadas con un microscopio electrónico. Para ello se deben realizar una serie de objetivos secundarios que ayudarán mediante su cumplimiento a satisfacer los requisitos necesarios para el correcto desarrollo del programa:

- OB-01. Estudio de los fundamentos teóricos para el procesamiento de imágenes.
 - Se pretende estudiar diferentes tecnologías relacionadas con el proyecto:
 - Formatos de imagen, filtros, software de tratamiento de imágenes, etc.
 - En cada caso, se identificarán las principales ventajas y desventajas de la utilización de cada una de ellas.
- OB-02. Estudio de los fundamentos del procesamiento de imágenes usando la GPU y la programación con OpenCL.

- Este objetivo supone satisfacer varios objetivos parciales como es el conocimiento en profundidad de unidades de procesamiento gráfico (GPU), y el conocimiento de la programación paralela OpenCl.
- OB-03. Estudio de la caracterización de microplásticos y microfibras naturales presentes en una imagen.
 - Permitirá identificar bordes y la posible difuminación de un objeto en la imagen para discriminar si se trata de un tipo de material u otro.
 - Será necesario caracterizar distintos tipos de micropartículas.
- OB-04. Desarrollo de un algoritmo en OpenCL para el procesamiento de imágenes basado en el análisis de cada píxel con sus adyacentes.
 - Se pretende identificar tipos de bordes distintos en los objetos de la imagen para poder discriminar el tipo de material presente en la imagen.
 - Se tendrá en cuenta la caracterización del objetivo anterior.
- OB-05. Optimización del nuevo algoritmo.
 - Se utilizará la programación paralela con OpenCL para hacer un uso eficiente de la memoria y el sistema hardware de GPU's.

- OB-06. Estudio comparativo de la calidad del nuevo algoritmo de detección de fibras naturales y microplásticos.
 - Para evaluar la calidad del nuevo algoritmo desarrollado, se comparará su rendimiento con el del algoritmo Canny.
 - La evaluación tendrá en cuenta el número y tipo de partículas elementos detectados en la imagen por cada algoritmo: desarrollado en este TFG y el de Canny.
 - Para conseguir este objetivo, es necesaria la implementación previa del algoritmo de Canny.
- OB-07: Configuración y documentación del entorno y sistemática de desarrollo.
 - Se pretende proporcionar información útil para la implementación de herramientas de programación en GPU con OpenCl en proyectos con necesidades similares.

Capítulo 4

Antecedentes

En este apartado se comentarán tanto tecnologías que se utilizarán en la realización del proyecto como tecnologías contempladas o relacionadas con la solución al problema planteado en el mismo. Estas abarcan conceptos de procesamiento de imagen, tecnologías de paralelización y algoritmos de detección de bordes. El proyecto se basa en una combinación de estos conceptos para su desarrollo.

4.1. Procesamiento de imagen

El procesamiento de imágenes [18] puede entenderse como el conjunto de técnicas que se aplican a las imágenes digitales [19] con el fin de mejorar la calidad o facilitar la búsqueda de cierta información en las mismas. Si bien la formulación matemática para su realización data de varios siglos atrás, el número de cálculos para su realización es muy elevado por lo que hasta que no se ha dispuesto de sistemas capaces de realizar miles o incluso millones de operaciones por segundo, razón por la que no ha sido posible su desarrollo.

En términos históricos, la utilización de imágenes se ha aplicado a numerosos campos como por ejemplo en las imágenes radiográficas en las que prácticamente desde el descubrimiento de los rayos X en 1895 han sido un medio fundamental para el diagnóstico clínico, sin

embargo las imágenes adquiridas eran sobre films radiográficos o directamente en vivo por lo que el real potencial de su uso no ha sido completamente explotado hasta la aparición de la tecnología que permitía su digitalización.

Dentro de este campo hay una gran variedad de técnicas y especializaciones que persiguen objetivos distintos, como se ha mencionado anteriormente el objetivo de este proyecto es la identificación y conteo de cierto tipo de partículas dentro de una imagen, por lo que a lo largo del algoritmo se le aplicarán tanto transformaciones de filtro, formato y extracción de información. Estas serán vistas en profundidad en los próximos capítulos pero para poder comprender con exactitud lo que se desarrollará más adelante hay que tener claros algunos conceptos que se explicarán a continuación:

- Píxel: es el elemento más pequeño que conforma una imagen, este tiene un color, una saturación y un brillo determinados, lo que da pie a numerosos formatos de imagen como los ya mencionados anteriormente.
- Mapa de bits: conjunto de píxeles que conforman una imagen, estos bits pueden ser triangulares, cuadrangulares o hexagonales, siendo el formato cuadrangular el más extendido y en el que nos centraremos durante el proyecto.
- Resolución: medida de longitud o definición de una imagen que representa la cantidad de filas y columnas de píxeles que conforman un mapa de bits determinado, esta se expresa en píxeles por pulgada.

- Adyacencia: dado un píxel P dentro de un mapa de bits M, se denomina como adyacente a cada píxel definido como vecino de P. Dicha vecindad puede ser de dos tipos:
 - 4-vecindad o 4-adyacencia: corresponde a los 4 píxeles cuyas regiones comparten un lado con P.

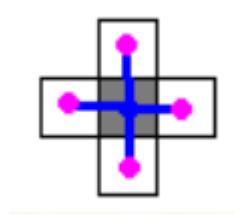


Figura 4.1: 4-adyacencia en imágenes digitales

- 8-vecindad u 8-adyacencia: corresponde a los píxeles cuyas regiones comparten un lado o un vértice con P.

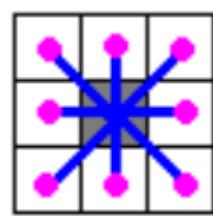


Figura 4.2: 8-adyacencia en imágenes digitales

Estos conceptos han sido de gran importancia a la hora de plantear un algoritmo como el desarrollado.

4.2. Unidades de procesamiento gráfico (GPU'S)

En los primeros ordenadores [20], la CPU era la encargada del proceso y gestión de todo tipo de información, desde datos que el usuario operaba hasta el propio sistema operativo. Por aquella época la interfaz del sistema operativo era basada en texto, pero más adelante con la aparición de las interfaces gráficas y de aplicaciones compatibles con estas interfaces se empezó a requerir de un nivel de exigencia que la CPU difícilmente podía afrontar. Las interfaces gráficas requerían de una alta carga matemática ya que había que mostrar un color, intensidad y saturación concretos para cada píxel de la pantalla en todo momento, para solucionar este problema se añadieron una serie de coprocesadores que quitaban carga a la CPU mediante operaciones en punto flotante, y mientras las exigencias fueron subiendo, estos coprocesadores evolucionaron a lo que llamamos hoy unidades de procesamiento gráfico o GPU'S, término acuñado por Nvidia en 1999.

Estas tenían una arquitectura hardware [21] muy diferente a las utilizadas en la CPU ya que iban orientadas a procesar elementos de gran carga computacional y por tanto debían de estar hechas de tal forma que pudieran realizar una operación sencilla para miles o incluso millones de elementos en un tiempo aceptable.

Esto es posible ya que la GPU a diferencia de la CPU tiene una gran cantidad de unidades computacionales pequeñas y simples mientras que la CPU tiene menor número de unidades pero de una potencia y complejidad superior. Debido a su gran número estas son perfectas para el procesamiento en paralelo ya que tienen un modelo de programación SIMD (Simple Instruction Multiple Data) esto significa que una instrucción se aplica a varios datos, de tal forma que adquiere una gran ventaja a la hora de procesar vectores, matrices y otro tipo de estructuras.

Para el fácil uso de las GPU's en el procesamiento de información existen diversas librerías que pueden ser utilizadas en distintos lenguajes de programación, como Python o C++. Algunas de estas librerías son CUDA (Nvidia) y OpenCL.

4.3. Algoritmos de detección de bordes.

La detección de bordes es una tecnología relacionada con la visión artificial y el procesamiento de imagen que consiste en detectar casos de objetos de una cierta clase tanto en imágenes digitales como en videos.

Aunque hay más desarrollo en ámbitos como la detección de caras o de personas, esta tecnología puede extrapolarse a gran variedad de áreas dentro de la visión artificial, como por ejemplo es el caso de este proyecto, con la detección de partículas microscópicas.

El concepto principal que envuelve a la detección de objetos es el de encontrar una característica común entre objetos del mismo tipo y que sea única con respecto a los demás tipos de objetos que no queremos identificar, una vez se tiene esta característica distintiva se aplican métodos de procesamiento de imagen para su reconocimiento.

Existen numerosos algoritmos de detección de objetos [22] que emplean diferentes técnicas de procesamiento, en este caso nos centraremos en el algoritmo de Canny [23] ya que ha sido una pieza fundamental en versiones tempranas del programa a desarrollar. Este algoritmo se basa en el gradiente de intensidad de una imagen, este término se refiere a la medida del cambio de una imagen en términos de intensidad por cada píxel, la magnitud de este gradiente indica la rapidez con la que la imagen está cambiando de intensidad y la dirección en la que este cambio es mayor, de esta forma se detectan los bordes de cada elemento ya que es ahí donde se produce un mayor cambio de intensidad.

4.4. OpenCV.

OpenCV [7] es una biblioteca libre de visión artificial que fue desarrollada originalmente por Intel y es conocida como la biblioteca más popular de visión artificial, siendo usada actualmente para una gran cantidad de aplicaciones. Esta biblioteca cuenta con una gran cantidad de algoritmos que abarcan varios campos desde la visión artificial clásica hasta mecanismos de aprendizaje automático. Estos se pueden utilizar para propósitos como la detección de rostros,

objetos [24], movimientos de cámara, rastreo de objetos en movimiento entre otros.

Numerosas empresas de alto renombre como Google, Yahoo, Microsoft o Intel utilizan la biblioteca y está integrada en una gran cantidad de campos de la tecnología desde la unión de imágenes en Street View hasta la ayuda a robots de navegación.

4.5. OpenCl.

OpenCl [5][1][2] es una API que permite a los programas paralelizar tanto los núcleos de la CPU y GPU como otros tipos de dispositivos por ejemplo los aceleradores, con el propósito de optimizar los tiempos de ejecución. En visión artificial muchos algoritmos pueden ejecutarse en una GPU de manera mucho más efectiva que en una CPU, por ejemplo en procesamiento de imágenes, aritmética matricial, fotografía computacional, detección de objetos y en general en tareas que requieran de mucha carga computacional.

Este nace en 2008 a manos del grupo Khronos Group, pero fue una propuesta iniciada por varias empresas conocidas como Intel, Nvidia, AMD e IBM entre otras y está preparado para su ejecución en los lenguajes C, C++, Python y java entre otros.

El funcionamiento de OpenCl se basa en la ejecución de un programa secuencial en primera instancia que posteriormente ejecuta una parte del código de forma paralela en la GPU, por tanto el código de cada programa OpenCl está formado por dos partes principales:

- Kernel: corresponde con el código ejecutado de forma paralela en la GPU, este código estará en una función llamada kernel, y será llamado desde la parte secuencial mediante funciones que preparan ciertos parámetros para que el lanzamiento del kernel se realice de manera satisfactoria.
- Host: es la parte del código que se ejecuta de forma secuencial, corresponde con la preparación de todos los elementos necesarios para lanzar el kernel y es la parte que menos varía de un programa OpenCl a otro ya que se repite el empleo de determinadas funciones a lo largo de todos los programas.

Parte II

Restricciones y especificaciones

Capítulo 5

Restricciones

A lo largo de este apartado se expondrán todas las restricciones existentes en el ámbito del diseño del proyecto y que son determinantes a la hora de elegir una alternativa u otra. Se identificarán dos tipos de factores como son los factores dato y estratégicos. Con el término factores dato nos referimos a aquellas restricciones que no son controladas sino que son debidas a la propia naturaleza del problema, la situación del proyecto o la disponibilidad de ciertos recursos. Los factores estratégicos son aquellas restricciones impuestas para cumplir los objetivos definidos en el proyecto, dentro de estos se encuentran decisiones como la elección de un algoritmo en lugar de otro o la elección de la tecnología a usar, todo orientado al correcto desarrollo del proyecto.

5.1 Factores dato.

A continuación se explicarán los factores dato tenidos en cuenta a la hora de realizar el proyecto, estos vienen determinados por la naturaleza del proyecto por lo que durante la realización del mismo se debe tenerlos en cuenta e intentar que afecten lo menos posible al resultado final.

- El programa recibe unas imágenes realizadas mediante microscopio electrónico, esto hace que las imágenes tengan unas características muy concretas siendo en ocasiones complicada la identificación de algunas partículas presentes.
- Las imágenes tienen un formato JPG o JPEG que corresponde con un formato de imagen comprimido por lo que es posible que cierta información se pierda a la hora de procesar la imagen si se compara con un formato sin compresión.
- El sistema deberá realizar la correcta diferenciación de las partículas que se pretendan identificar con respecto a las demás presentes en la imagen.
- El programa precisa de unos parámetros diferentes para cada tamaño de imagen ya que según el número de píxeles habrá diferencias más grandes o más pequeñas, por lo que se debe redimensionar las imágenes recibidas a un tamaño fijo.

5.2 Factores estratégicos.

A continuación se desarrollarán todos los factores estratégicos tenidos en cuenta a lo largo del proyecto, explicando sus características, funcionamiento y resultados para concluir con la indicación de la opción final elegida y la razón de dicha elección.

Antes de comenzar con la explicación de los factores estratégicos se debe dividir el programa desarrollado en tres fases principales:

- Recolección y aplicación de filtros a la imagen: corresponde con la fase en la que el programa recoge la imagen y aplica una serie de filtros que facilitan su procesamiento posterior.
- Identificación de elementos: corresponde con la identificación de los elementos presentes en la imagen diferenciándolos del fondo de la imagen.
- Diferenciación y clasificación de elementos: corresponde con la clasificación de los elementos recogidos en la fase anterior según sus características, en este caso como se ha comentado anteriormente la característica distintiva a la hora de clasificar las partículas será su borde.

5.2.1 OpenCV.

Como se ha mencionado en capítulos previos del proyecto se utilizará la librería OpenCV para realizar las tareas relacionadas con el procesamiento de la imagen, estas serán cuatro tareas principales:

- Recogida de la imagen: se encarga de recoger la imagen de una ruta determinada que se especificará al programa y la carga en forma de matriz de píxeles para su procesamiento.
- Aplicación de filtros: el tamaño de las imágenes procesadas será de 4056 x 3040 de tal forma que si el programa recibe una imagen con un tamaño distinto a este la redimensiona mediante OpenCV, también se aplica un filtro que convierte la imagen a escala de grises, la razón por la que se efectúa este cambio es que en el formato RGB la imagen tiene tres canales por cada píxel, el rojo, el verde y el azul, mientras que en escala de grises cada píxel solo contiene un canal que especifica su valor en la escala entre el blanco y el negro, lo que facilita en gran medida su procesamiento posterior.
- Identificación y conteo de contornos: una vez tenemos una matriz con las partículas a identificar se procede, mediante una función de OpenCV, a guardarlas como contorno con sus coordenadas, número de píxeles y medida de la longitud de cada una.

- Guardado de resultados: mediante OpenCV se guardarán tres imágenes, la imagen original, otra imagen con todos los elementos presentes en la imagen original resaltados y por último, la imagen que contiene el dibujo de los contornos de la partícula que se quiere identificar. Este proceso se realiza para facilitar la visualización de las distintas fases por las que pasa el programa hasta llegar al resultado final.

5.2.2 Nvidia CUDA y OpenCl.

Las principales herramientas para la paralelización en GPU son Nvidia CUDA y OpenCl.

Cuda es un modelo de programación desarrollado por Nvidia para la programación paralela en GPU, este funciona en todas las GPU de Nvidia e incluye una gran cantidad de herramientas que facilitan este tipo de programación. Finalmente este lenguaje no fue escogido por una razón principal, y es que GPU'S de otra marca y arquitectura como puede ser de Intel no tienen soporte para esta herramienta, lo que podía suponer una limitación a la hora de escoger un dispositivo apropiado para el desarrollo de este proyecto.

Por otra parte se encuentra OpenCl, que al igual que CUDA es una herramienta para la programación paralela en GPU que incluye una serie de herramientas que permiten la utilización de este tipo de programación. Hay dos ventajas principales que se presentan a la hora de usar OpenCl con respecto CUDA, la primera es el conocimiento previo que se tenía sobre esta herramienta lo que aporta mayor

sencillez y eficacia a la hora de desarrollar un algoritmo como el que se está viendo. La segunda es que OpenCl es un estándar abierto y libre de derechos lo que permite su ejecución en una variedad más amplia de dispositivos, lo que es un factor determinante a la hora de que el programa funcione en la mayor cantidad de dispositivos posibles.

5.2.3 Python.

El lenguaje de programación utilizado a la hora de desarrollar el proyecto ha sido Python [25], este es un lenguaje de alto nivel que se utiliza para el desarrollo de aplicaciones de todo tipo.

Este lenguaje es muy sencillo tanto de leer como de escribir ya que tiene una gran similitud con el lenguaje humano lo que supone una gran ventaja a la hora de que otros usuarios entiendan de una manera sencilla el código del programa. Además permite el desarrollo de un número muy amplio de aplicaciones ya que facilita el trabajo en campos como la inteligencia artificial, el big data, machine learning y data science entre otros.

Estas características han facilitado en gran medida el desarrollo del proyecto.

5.2.4 Algoritmo de Canny.

Para la fase de identificación de elementos en el comienzo del desarrollo se optó por el algoritmo de Canny, como se ha mencionado anteriormente este es un algoritmo que se basa en el gradiente de intensidad de una imagen para identificar los bordes de los distintos elementos que la integran.

Este algoritmo era muy útil y sencillo de utilizar ya que es posible integrarlo mediante una función presente en OpenCV que devuelve una imagen binaria con los contornos presentes en la imagen. A continuación se mostrará un ejemplo de identificación de contornos mediante el algoritmo de canny a una de las imágenes utilizadas como muestra durante el proyecto:

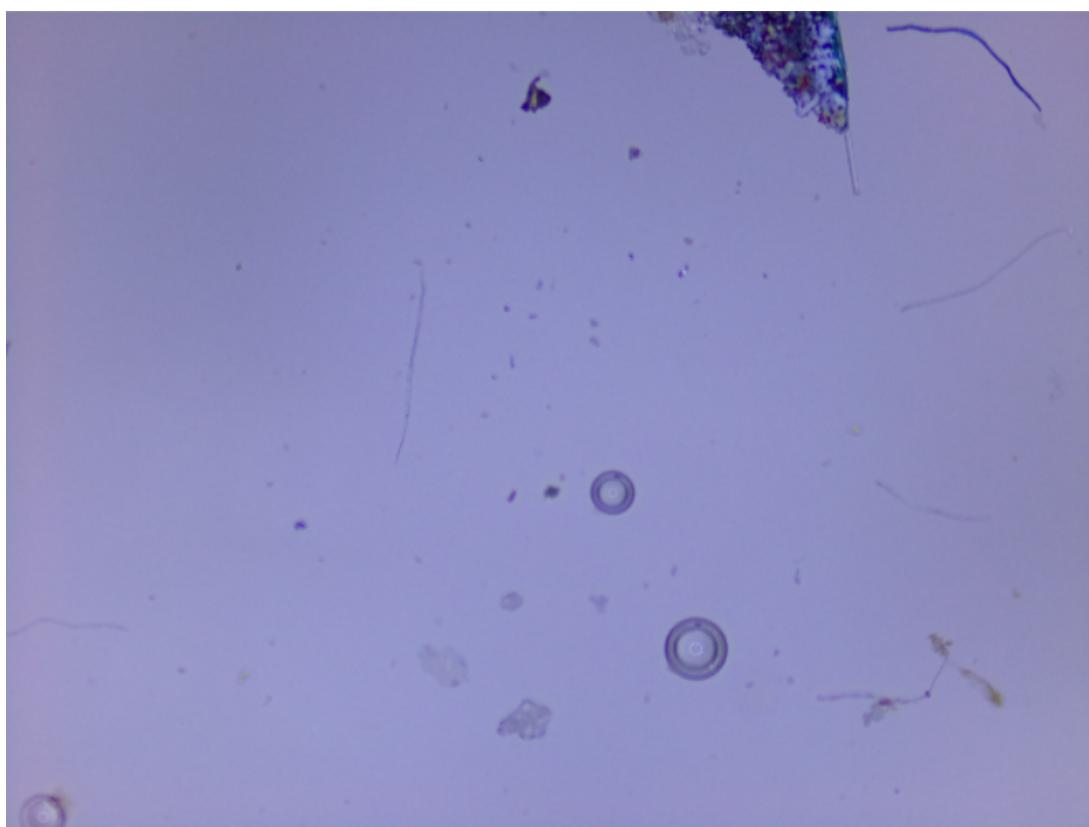


Figura 5.2: Prueba con algoritmo de Canny (imagen original)

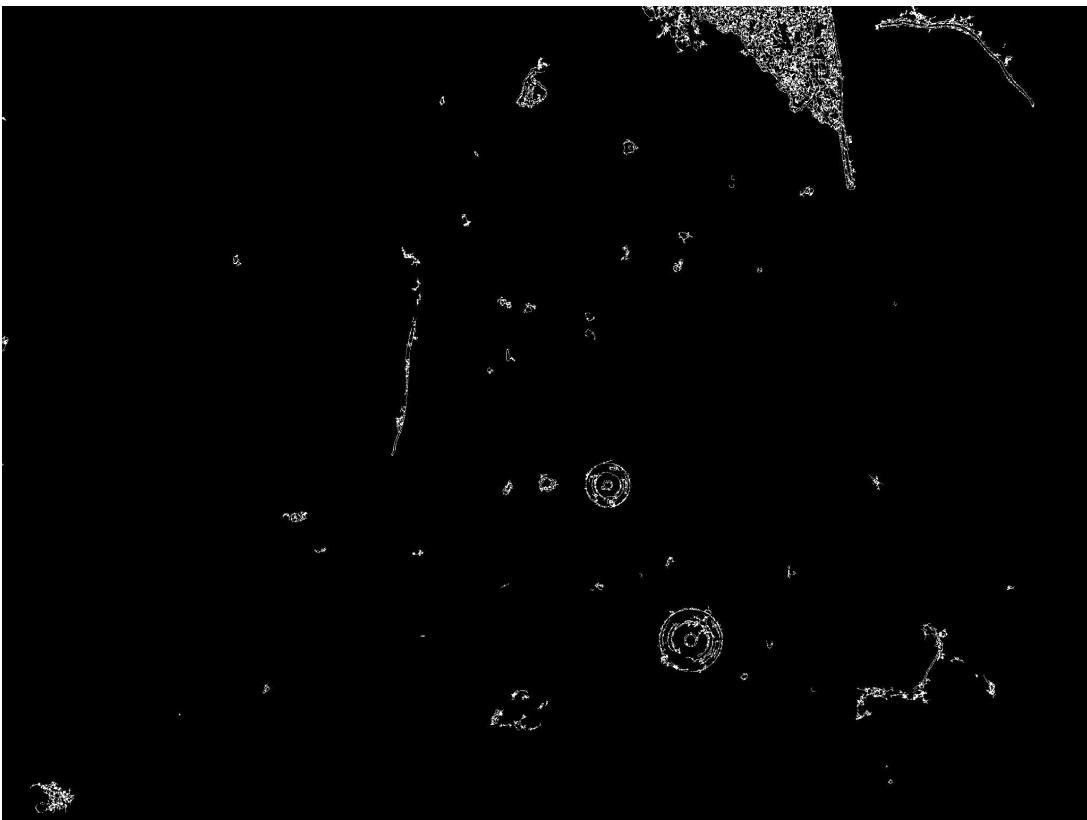


Figura 5.3: Prueba con algoritmo de Canny (imagen de contornos)

Se puede observar que los elementos de la imagen se recogen con gran detalle lo que hace que se identifiquen las partículas en partes divididas, eso supone un problema ya que se pretende identificar los objetos enteros y no separados en varias partes.

Una vez recogidos los elementos se realizaba un procesamiento posterior basado en los cambios con respecto a los ejes X e Y de cada contorno, ya que en un principio la característica distintiva que se propuso para diferenciar los tipos de partículas era su forma.

Finalmente tanto la opción del algoritmo de Canny como la identificación de elementos por su forma no fueron escogidas por los argumentos expuestos anteriormente.

5.2.5 Algoritmo identificador del fondo.

Dado que el algoritmo de Canny no reconocía algunos elementos de la imagen por su parecido con el fondo de la misma, se decidió a realizar un algoritmo que identificara el color de fondo de la imagen, para una vez conseguido, cambiarlo por otro que creara un mayor contraste con respecto a los demás elementos con el fin de facilitar la tarea al algoritmo de Canny de identificar todas las partículas posibles y sin dividirlas en varias partes.

Este algoritmo se realizó mediante OpenCl y su funcionamiento se basa en el recorrido de cada uno de los píxeles de la imagen. Por cada píxel, se analiza su color en escala de gris, se compara con un vector en el que se almacenan los colores de los demás píxeles de la imagen, y si este color no se encuentra en el vector se anota junto con el número de apariciones que ha tenido, en el caso de que este color ya estuviera presente en el vector se incrementa en uno el número de apariciones que ha tenido.

De esta forma el algoritmo devuelve un vector con los diferentes colores presentes en la imagen y el número de veces que aparece cada uno, siendo el color de fondo el que mayor número de apariciones tenga ya que es el color predominante en la imagen.

Finalmente se cambia el color de fondo por otro que cree más contraste y se realiza el algoritmo de Canny. Tras varias pruebas se llegó a la conclusión de que esta opción no era la más acertada ya que

se seguía sin obtener con claridad algunos elementos de las imágenes y tampoco era la opción más eficiente.

5.2.6 Algoritmo identificador de borde.

Tras comprobar que el algoritmo de Canny no era la opción más apropiada a la hora de identificar los contornos de los elementos menos reconocibles en la imagen se pensó en el desarrollo de un nuevo algoritmo que fuera capaz de identificar elementos con un más alto nivel de sensibilidad y que realice a la misma vez la tarea de identificar y diferenciar las partículas que se pretenden.

Como hemos mencionado anteriormente la característica distintiva utilizada para la identificación de las partículas es el borde, esto nos permite identificar elementos de la imagen a la vez que descartamos los que no tengan un borde apropiado.

El algoritmo funciona de tal manera que se analiza cada píxel de la imagen en escala de grises y se compara uno por uno con sus adyacentes, esta comparación verifica si hay un cambio significativo de color entre uno y otro, lo que supondría el inicio de un borde. Si hay un cambio importante, se sigue comparando con el píxel de delante para comprobar de nuevo si sigue este cambio notable, este proceso se repite hasta que deje de haber una diferencia importante entre píxeles, que será el indicador de que se ha analizado el borde completo. Una vez ha terminado el proceso, se dibuja el píxel en una

matriz imagen en formato binario (blanco y negro) para indicar que nos encontramos ante un píxel del borde.

Si se sigue el ejemplo de los microplásticos, estos tienen un borde muy escarpado por lo que a la hora de identificarlos se debe tener en cuenta las partículas que tengan un borde que incluya un número pequeño de píxeles, este será el criterio mediante el que se identifiquen las diferentes partículas de la imagen.

Tras numerosas pruebas se ha comprobado que este algoritmo es más eficiente a la hora de identificar elementos, por lo que lo hace más adecuado para la tarea que se está realizando.

Capítulo 6

Recursos

Durante este capítulo se mencionarán todos los recursos que han sido utilizados para la realización del proyecto, estos serán de tres tipos; recursos software, recursos hardware y recursos humanos.

6.1. Recursos Software

En el siguiente cuadro se enumeran los diferentes recursos utilizados para el desarrollo, documentación y experimentación del proyecto.

Entorno de desarrollo:	<p>El sistema operativo utilizado para el desarrollo es Windows 10.</p> <p>Como lenguaje de programación se ha utilizado Python en su versión 3.9.</p> <p>Como editor se ha utilizado el IDLE de Python.</p> <p>Como librería de procesamiento de imagen se ha utilizado OpenCV.</p> <p>Como librería de programación paralela en GPU se ha utilizado OpenCl.</p> <p>Se ha utilizado la librería Numpy para tareas relacionadas con operaciones aritméticas y el manejo de vectores y matrices en Python.</p>
Entorno de documentación:	<p>Se ha utilizado la herramienta online Google Docs para la documentación del proyecto.</p>

Cuadro 6.1: Recursos Software

6.2. Recursos Hardware.

En el siguiente cuadro se detallarán los recursos hardware que han permitido el correcto desarrollo del proyecto especificando sus características principales.

Entorno de desarrollo y documentación:	Ordenador portátil Lenovo Ideapad 330 : Procesador Intel Core i7 8750-H . Tarjeta gráfica: Nvidia Geforce GTX 1050 Memoria RAM: 16 GB DDR4
--	---

Cuadro 6.2: Recursos Hardware

6.3. Recursos Humanos.

Por último, se enumeran las personas implicadas en el desarrollo del proyecto.

Directores:	Ezequiel Herruzo Gómez y Jesús Rioja Bravo. Departamento de ingeniería electrónica y computadores. Universidad de Córdoba.
Autor:	José Sánchez Yun Alumno en el grado de ingeniería informática en especialidad de computadores. Universidad de Córdoba.

Cuadro 6.3: Recursos Humanos

Capítulo 7

Especificación de Requisitos

A continuación se definirán los requisitos que debe satisfacer el sistema para cumplir con los objetivos propuestos. Dichos requisitos se dividirán en requisitos de usuario, requisitos funcionales, requisitos no funcionales y requisitos de información.

7.1. Requisitos de Usuario.

Los requisitos de usuario son aquellos inherentes a la definición del problema y afectan directamente al diseño del sistema. Estos se muestran a continuación en el cuadro 7.1.

Requisitos de usuario (RU)	
RU-01	Las imágenes tratadas serán en formato JPG o JPEG.
RU-02	Se deberá poder observar la información recopilada.
RU-03	Solo se podrá identificar un tipo de micropartícula al mismo tiempo.
RU-04	Se deberá proporcionar el resultado en un tiempo aceptable.
RU-05	El programa deberá tener una interfaz de usuario sencilla y comprensible.

Cuadro 7.1: Requisitos de usuario

7.2. Requisitos Funcionales.

Este tipo de requisitos definen la funcionalidad del software, indicando las diferentes tareas que el programa debe ser capaz de realizar. Estos se muestran a continuación en el cuadro 7.2.

Requisitos Funcionales (RF)	
RF-01	El sistema deberá poder identificar distintos tipos de partículas.
RF-02	El sistema deberá comprobar si el tamaño de las imágenes es el apropiado.
RF-03	Se deberá poder introducir nuevos tipos de partícula así como sus parámetros de identificación.
RF-04	Se deberá poder modificar los parámetros de identificación de un tipo determinado de partícula.
RF-05	Se llevará un registro de los distintos tipos de partículas disponibles a identificar por el sistema.

Cuadro 7.2: Requisitos Funcionales

7.3. Requisitos no Funcionales.

Los requisitos no funcionales son aquellos que especifican criterios que pueden usarse para juzgar la operación del sistema en lugar de sus comportamientos específicos. Estos se muestran a continuación en el cuadro 7.3.

Requisitos no Funcionales (RNF)	
RNF-01	El sistema deberá de estar dividido en módulos repartidos según las distintas partes que lo componen.
RNF-02	El sistema deberá permitir la escalabilidad mediante la posibilidad de realizar modificaciones y ampliaciones del mismo.
RNF-03	El código del sistema deberá de ser comprensible aportando comentarios que faciliten la lectura del mismo.

Cuadro 7.3: Requisitos no Funcionales

7.4. Requisitos de Información.

Los requisitos de información son aquellos que describen la información que debe almacenar y gestionar el sistema. Estos se muestran a continuación en el cuadro 7.4.

Requisitos de Información (RI)	
RI-01	El sistema deberá proporcionar un archivo con la información de cada partícula encontrada, incluyendo el número y longitud de cada una.
RI-02	El sistema deberá proporcionar una imagen resaltando las partículas identificadas.
RI-03	Los parámetros de identificación de cada partícula así como su nombre deberán estar registrados en el sistema.

Cuadro 7.4: Requisitos de Información

Parte III

Desarrollo

Capítulo 8

Análisis Funcional

En este capítulo, se explicará la idea inicial del algoritmo planteado, se establecerán conceptos para su implementación y se definirá el algoritmo a desarrollar, detallando los diagramas de flujo que lo definen. Este modelado se realizará mediante el uso de UML (Unified Modeling Language).

Como se ha mencionado anteriormente se diseñará e implementará un algoritmo de detección de micropartículas según su borde, analizando la diferencia presente entre píxeles adyacentes para identificar si se trata de un borde o no. Para ello, hay que definir una serie de conceptos que ayudan a entender en mejor manera lo que realiza el algoritmo, entre estos conceptos figuran los siguientes:

- Margen: corresponde con la diferencia mínima, en parámetros de color y/o escala de grises, que se debe de producir entre un píxel y otro píxel adyacente para considerar que se está produciendo un borde.
- Borde de tipo de material (Btm): corresponde con el número de píxeles necesarios para representar el grosor completo de un borde. Este concepto se define mediante un contador que analiza las diferencias entre los píxeles adyacentes, si se produce una diferencia superior al margen entre dos píxeles, el btm se

incrementa en uno y sigue la comparación con el píxel siguiente al ya comparado para analizar si se sigue produciendo esa diferencia, una vez la diferencia termine, habrá terminado el borde, y el btm tendrá un valor determinado por el número de píxeles que han sido necesarios para representar dicho borde completo.

La idea básica es que el borde de una partícula concreta esté determinado por el margen y por el btm mínimo y máximo que puede tener para ser considerada un tipo de partícula u otro.

Como se ha mencionado en capítulos previos, el algoritmo será dividido en dos partes, la parte del host que se ejecuta en la CPU, en la que se prepararán los parámetros necesarios para lanzar la función paralela en la GPU, y la parte del kernel, que corresponderá con el algoritmo propiamente dicho y que se ejecutará en una de las GPU's del sistema. A continuación se detallarán los diagramas de flujo de ambas partes.

8.1. Código Host.

La secuencia de acciones que sigue esta parte del código se muestra en el siguiente diagrama de flujo:

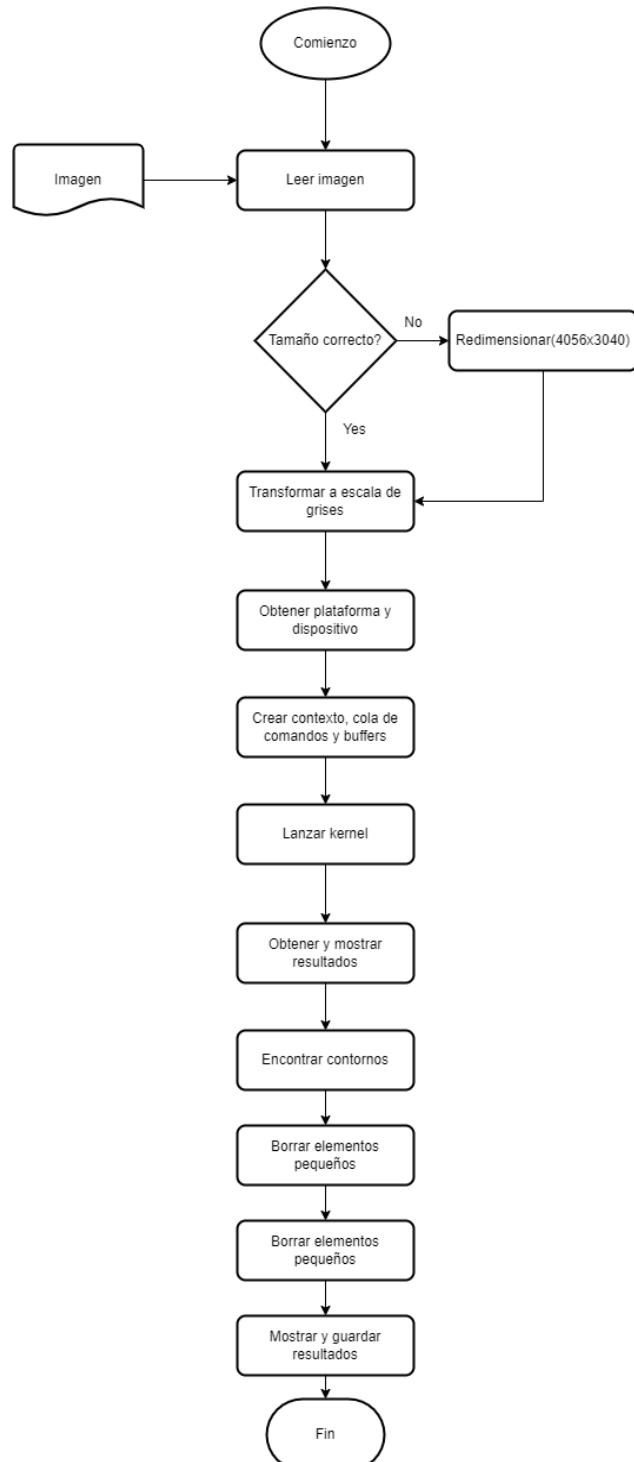


Figura 8.1: Diagrama de Flujo (Código Host)

En la figura anterior 8.1, se aprecia que tras el inicio del programa se necesita la imagen a procesar. Tras la lectura de la imagen y la comprobación del tamaño de la misma se inicia todo el proceso. En primer lugar, la imagen se transforma a escala de grises para que pueda procesar mediante un único canal de color. A continuación, se obtienen las plataformas y dispositivos GPU's y se crea el contexto necesario sobre las mismas antes de lanzar la ejecución del kernel en el dispositivo seleccionado.

Una vez devuelta la ejecución del kernel por parte de la GPU se obtienen y muestran los resultados tras la eliminación de la información no significativa que haya podido devolver la ejecución en la GPU. Por último, se calcula el tamaño de cada partícula y el porcentaje total de partículas presentes con respecto a la imagen completa y se guardan los resultados.

8.2. Código Kernel.

La secuencia de acciones que sigue esta parte del código es la siguiente:

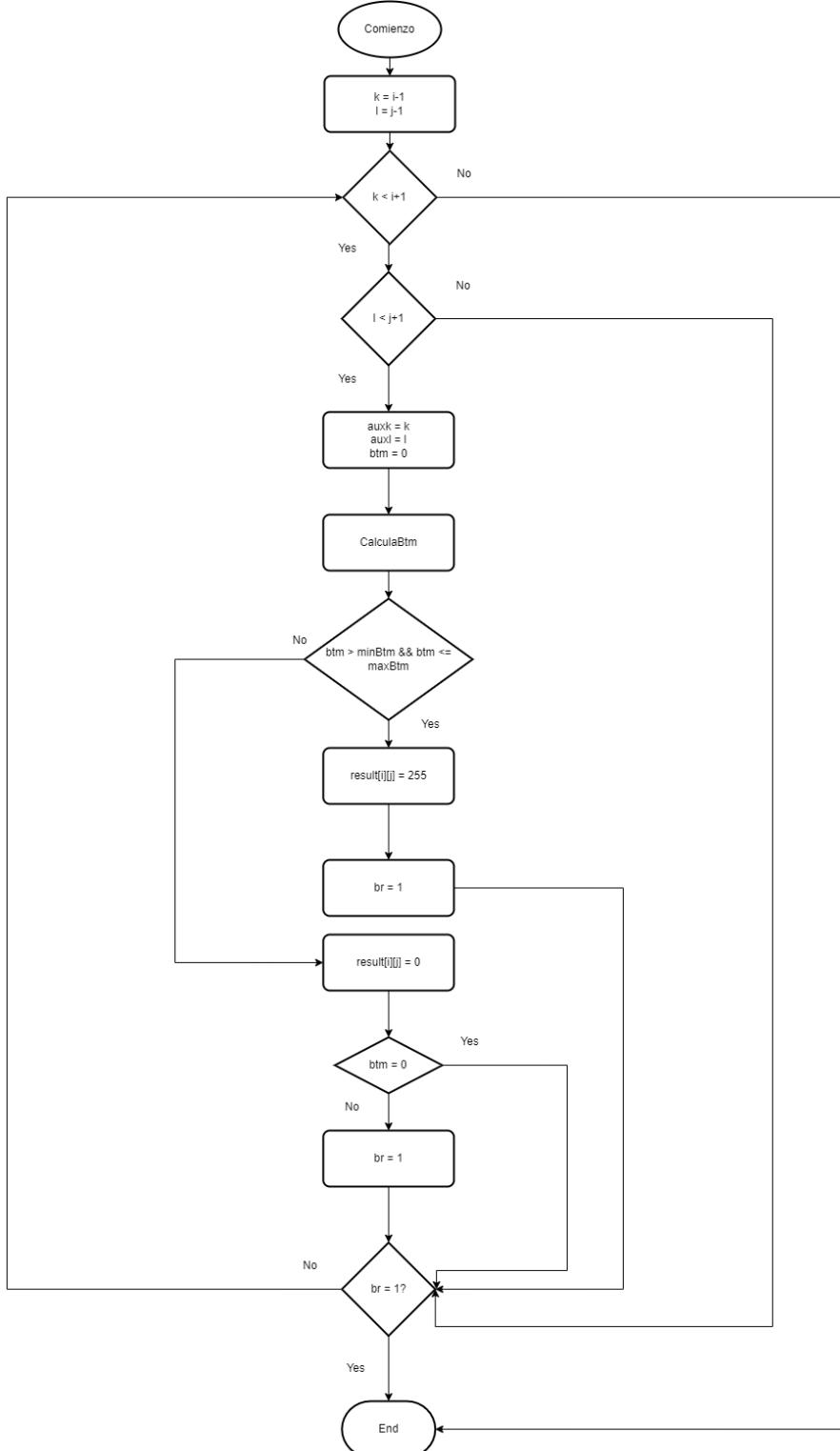


Figura 8.2: Diagrama de Flujo (Código Kernel)

El primer paso es establecer el valor de los cuatro índices, habrá dos para referirse a la posición del píxel central, estos son i y j , mientras que k y l servirán para recorrer a los píxeles adyacentes al central con el objetivo de comparar sus valores y comprobar si se trata de borde o no. Acto seguido comenzarán los dos bucles que recorren dichos píxeles, se establecerá el valor del btm a 0 y se guardará el valor de los índices k y l en unas variables auxiliares para restaurarlos posteriormente ya que cambiará su valor durante el procesado.

Después comenzará el proceso `CalculaBtm`, cuyo diagrama se muestra a continuación:

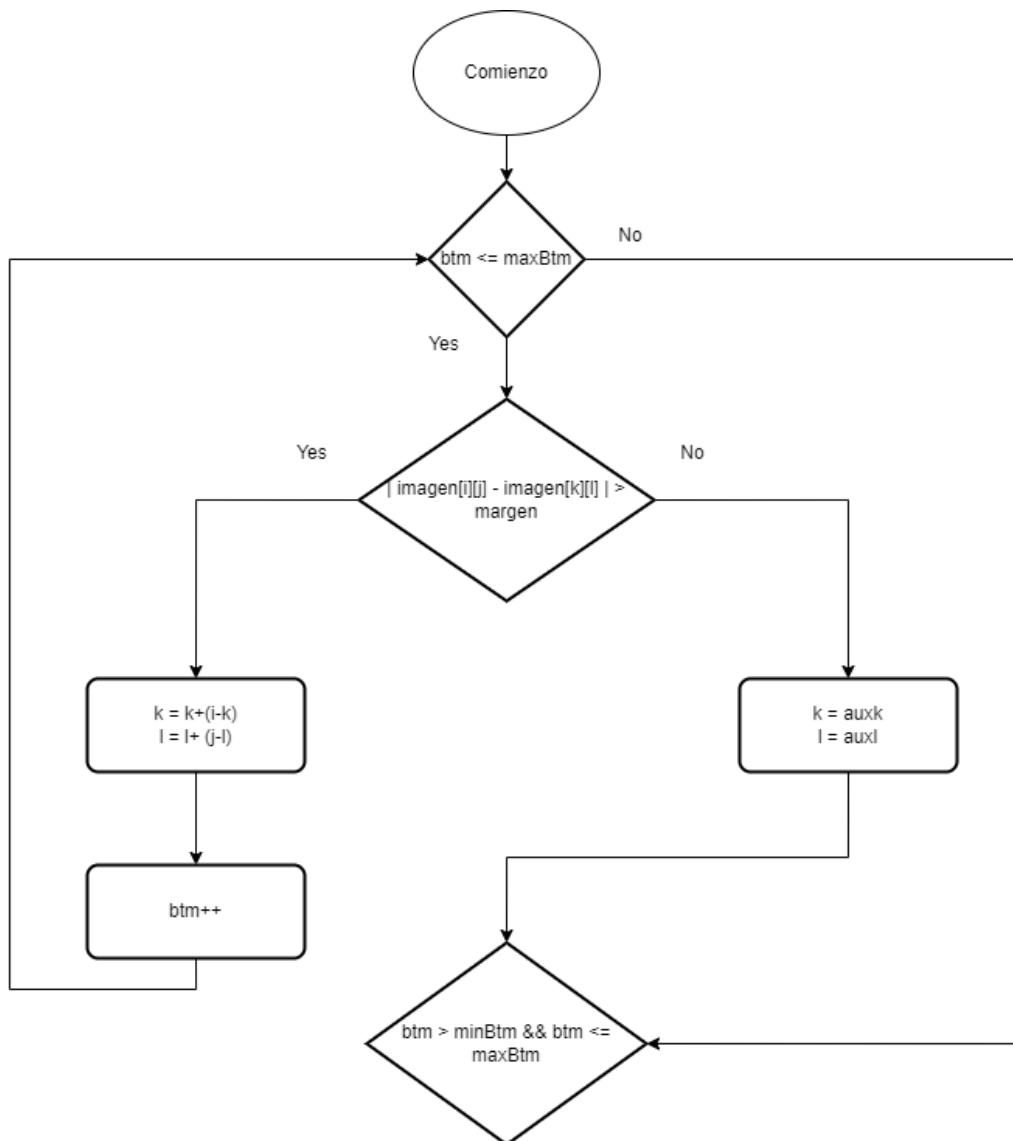


Figura 8.3: Diagrama de Flujo (`Calcula btm`)

Este proceso trata de un bucle en el que se computa el btm del píxel central, para ello se recorre cada píxel adyacente y se comprueba si la diferencia entre su valor con el central es mayor que el margen establecido, si esto se cumple, se repite el proceso con el píxel de delante y se incrementa el btm. Cuando la diferencia no sea mayor que el margen, se restauran los índices k y l y se pasa a la siguiente fase.

El proceso final de este diagrama corresponde con la continuación del diagrama de la figura 8.2 tras computar el btm.

A continuación se comprueba si el btm está dentro de los márgenes de btm mínimo y máximo establecidos, y si esto es así, se marca el píxel central en blanco en la imagen resultado y se sale del bucle estableciendo la variable br a 1 para terminar el proceso. En cambio si esto no se cumple, se marca dicho píxel a 0 y se comprueba si el btm es 0, si esto se cumple se sigue comprobando el siguiente píxel adyacente, y si no se cumple, se establecerá br a 1 y se terminará el proceso.

Capítulo 9

Sistema Hardware

A continuación, se detalla la arquitectura de los dos elementos hardware fundamentales que se utilizan en el sistema desarrollado y que ayudarán a entender mejor el concepto de programación paralela que se ha realizado. Estos dos elementos son el procesador y la GPU, y se explicará de forma resumida las partes que los integran, su estructura y funcionamiento.

9.1. Arquitectura del procesador.

Como ha sido mencionado en el apartado de recursos hardware, el proyecto ha sido realizado mediante el uso de un procesador Intel Core i7 8750-H.

Las principales especificaciones del procesador son:

- 6 núcleos.
- 12 hilos.
- 2.2 GHz de frecuencia base, con una frecuencia máxima de 4.10 GHz.
- 9 MB Intel Smart Caché.
- Gráficos integrados UHD Intel 630.

La arquitectura de un procesador [8] define su estructura interna, determinando la ubicación de las distintas partes que lo componen. El interior de un procesador se compone de una serie de bloques interconectados en el cuál cada uno se encarga de realizar una función específica, su estructura se centra en el diseño de dichos elementos y cómo se interconectan.

En este caso, la arquitectura del procesador es Coffee Lake H, esta fue introducida en 2017 y permitió la integración de un 50% más de núcleos con respecto a su predecesora Kaby Lake.

Esta se compone de 5 elementos principales:

- Núcleos de la CPU [9]: son el elemento principal dentro de un procesador y son los encargados de realizar las operaciones matemáticas necesarias para que el dispositivo funcione correctamente. En este caso se dispone de 6.
- Agente del sistema: este elemento es el encargado de realizar las funciones del microprocesador que no se encuentran dentro de los núcleos.
- GPU Integrada: este elemento fue integrado con el fin de despejar de carga computacional a los núcleos desarrollando tareas de procesamiento gráfico, no poseen la potencia de una GPU dedicada pero es suficiente para sistemas que no precisen de un gran procesamiento gráfico.
- Memoria caché L3 [10]: se trata de una memoria caché específica para cada núcleo del procesador.

- Anillo Interconector: se trata de un bus ubicado entre los núcleos, la memoria caché y la GPU y cuya tarea principal se trata de la conexión de todos los elementos mencionados previamente.

En la siguiente imagen se puede apreciar con claridad la ubicación de los elementos mencionados dentro del microprocesador:

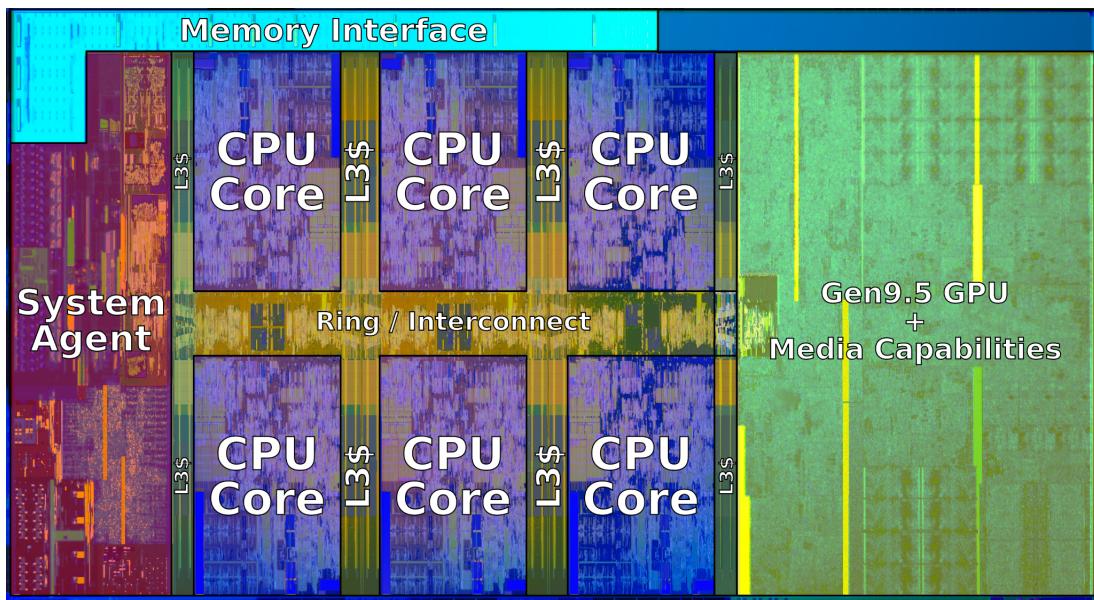


Figura 9.1: arquitectura CPU

Para explorar nuevas formas de optimización se debe conocer a la perfección esta arquitectura, así como la del elemento que se explicará a continuación, la GPU.

9.2. Arquitectura de la GPU.

Una GPU o unidad de procesamiento gráfico es un procesador dedicado al procesamiento de datos relacionados con el vídeo y las imágenes que se están reproduciendo en el ordenador, esto hace que el procesador tenga menos carga por lo que puede dedicar más recursos a otras funciones.

El entendimiento de esta estructura es fundamental para la herramienta que se va a utilizar a la hora del desarrollo del proyecto, OpenCl.

Como se ha mencionado anteriormente en el apartado de recursos hardware, se utilizará una tarjeta gráfica Nvidia GeForce GTX 1050 [11], esta trabaja a una frecuencia base de 1354 MHz, llegando a una frecuencia turbo de 1455 MHz. Este dispositivo tiene una arquitectura Pascal [12], la cuál se encuentra estructurada de la siguiente manera:

Así como los procesadores constan de pocos núcleos que trabajan a altas frecuencias de reloj, las GPU tienden a lo contrario, presentando una gran cantidad de núcleos que trabajan a poca frecuencia, ya que para el procesamiento de gráficos se precisa de operaciones de baja complejidad pero aplicadas a un gran número de píxeles, por lo que supone una gran ventaja el disponer de un número más alto de núcleos.

El primer elemento a destacar en una GPU son los núcleos, que como hemos mencionado anteriormente se encuentran en un gran número, estos pueden realizar operaciones en punto flotante de precisión simple y doble, y son la base de la estructura de la GPU, en la siguiente imagen se puede observar su estructura de manera gráfica:

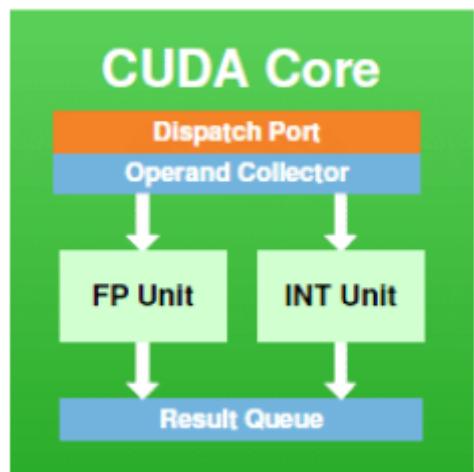


Figura 9.2: Núcleo GPU

Estos núcleos son agrupados en bloques llamados Streaming Multiprocessor o SM, los cuales son procesadores de propósito general que tienen una frecuencia baja y una memoria caché de pequeño tamaño. Dentro de ellos encontramos los siguientes elementos:

- Núcleos.
- Memoria caché.
- Planificadores de warps.
- registros.

La GPU programa bloques de hilos a cada SM de tal forma que todos los hilos presentes dentro de un bloque deben residir en el SM en el cuál se ha programado el bloque. Cada SM contiene 8 núcleos que pueden funcionar simultáneamente, pudiendo ejecutarse hasta 32 hilos al mismo tiempo, estas agrupaciones son llamadas warps y son un concepto muy importante a tener en cuenta a la hora de optimizar el programa. Estos warps tienen un modelo de ejecución SIMD (simple instruction multiple data) ya que cada uno ejecuta una misma instrucción al mismo tiempo en múltiples núcleos.

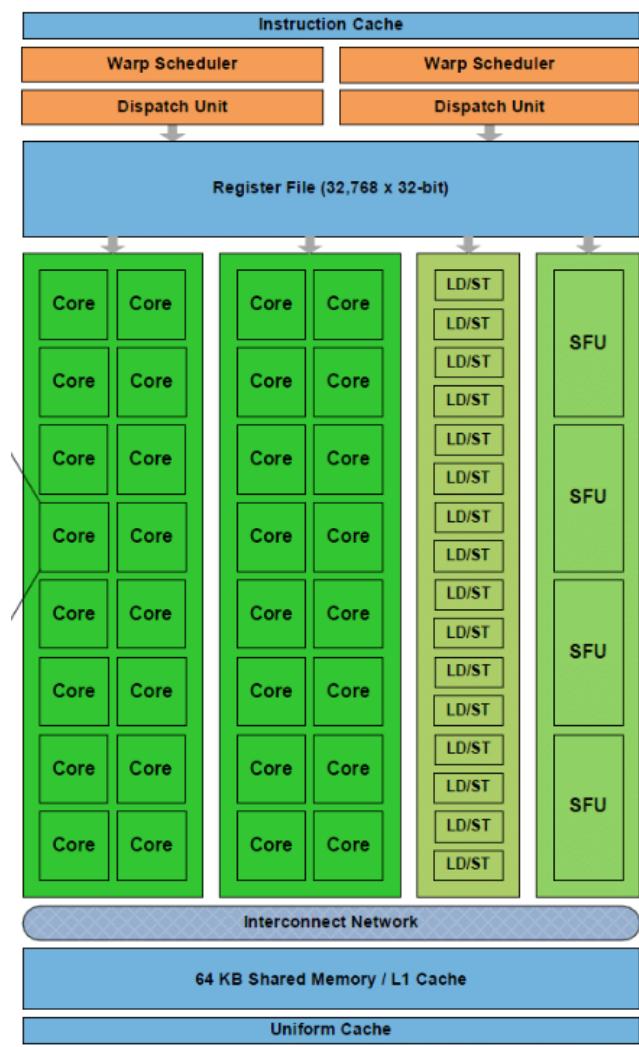


Figura 9.3: Streaming Multiprocessor

Los SM son agrupados a su vez dentro de los llamados TPC, que en conjunto forman la arquitectura completa de la GPU como podemos ver en la siguiente ilustración:



Figura 9.4: TPC GPU en arquitectura Pascal

Otro elemento fundamental a destacar en la arquitectura de una GPU es su modelo de memoria [13][2][6], que puede ser utilizado para optimizar aún más los programas OpenCl. Este modelo se divide en los siguientes elementos:

- Memoria global: es la memoria de mayor tamaño y a su vez la que tiene mayor latencia. Debido a su tamaño se utiliza como contenedor a la hora de copiar datos entre el Host (CPU) y el Device (GPU).

Estos datos pueden ser copiados posteriormente a memorias más eficientes de forma estructurada.

- Memoria compartida: esta memoria se encuentra dentro de cada SM y consta de una latencia muy baja. Su principal propósito es el de intercambiar información entre los hilos pertenecientes a un mismo bloque dentro de un SM, esta es útil a la hora de copiar datos que van a ser reutilizados dentro del kernel, lo que beneficia la optimización ya que el acceso desde los núcleos a esta memoria es más rápido que a memoria global.
- Memoria local: cuando un hilo ejecutado dentro de un SM ha ocupado el número máximo de registros disponibles en el SM, las variables propias de ese hilo pasan a almacenarse en la memoria local para no afectar al rendimiento.
- Registros: son la memoria más cercana a los núcleos, esta es la de más bajo nivel y menor latencia. El número de registros utilizado por un kernel afecta al número de hilos que pueden ejecutarse en un multiprocesador de forma paralela.
- Caché constante: es una memoria utilizada para accesos de solo lectura de forma simultánea por múltiples hilos en paralelo.
- Caché de texturas: es una memoria utilizada mayormente cuando se tienen datos de solo lectura sobre los que se van a realizar operaciones que requieren accesos no coalescentes a memoria.

Capítulo 10

Sistemas Software

A lo largo de este capítulo se explicarán los elementos software utilizados a la hora de realizar el proyecto así como las instrucciones de preparación del entorno de desarrollo.

10.1. Preparación del entorno de desarrollo

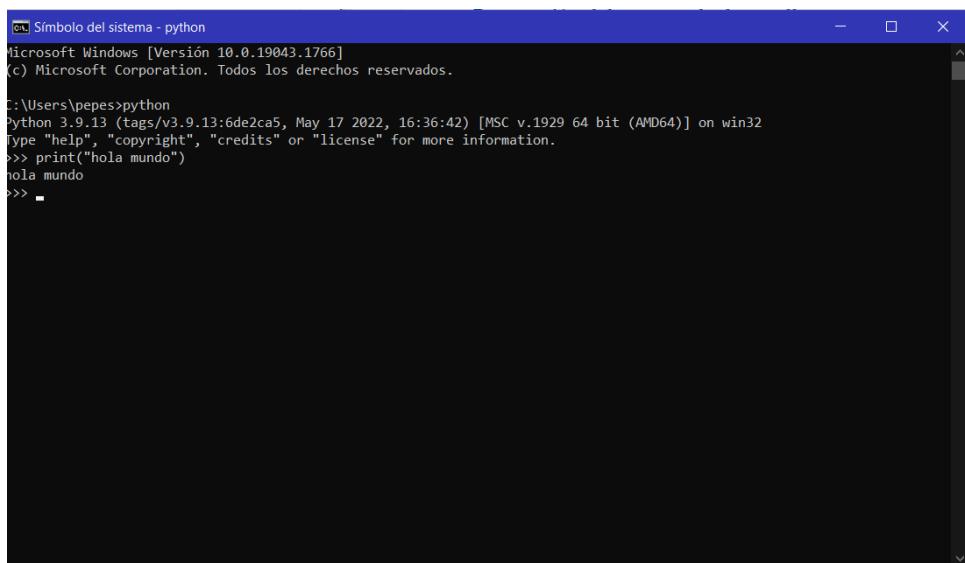
El programa ha sido desarrollado haciendo uso del lenguaje de programación Python en su versión 3.9 en el sistema operativo Windows 10 Home.

Como se ha mencionado anteriormente también se hará uso de las librerías OpenCl para programación paralela y OpenCV para procesamiento de imagen, también se utilizará la herramienta Tkinter para desarrollar la interfaz gráfica por lo que para realizar la correcta instalación del entorno de desarrollo se necesitan cumplir cuatro requisitos, estos son la instalación de Python, las librerías OpenCl, OpenCV y Tkinter.

Junto con la instalación de Python viene incluido un entorno de programación llamado Python IDLE, este goza de un aspecto sencillo, lo que facilita la programación y más utilizando un lenguaje como este, en el que predomina la legibilidad y la simplicidad, por lo que ha sido el elegido a la hora de desarrollar el proyecto.

La instalación de cada uno de los componentes se describe a continuación:

- **Python [26]:** es posible instalar Python para Windows 10 de una manera muy sencilla, en la página oficial de Python si se dirige al apartado de descargas se puede seleccionar la versión de este lenguaje que se precise, una vez descargado se tendrá que ejecutar el archivo resultante de la descarga para comenzar con la instalación. Para verificar que la instalación se ha desarrollado de manera satisfactoria es posible introducir el comando “Python” dentro de la terminal de Windows, apareciendo una terminal como la que se muestra a continuación:



```
C:\Símbolo del sistema - python
Microsoft Windows [Versión 10.0.19043.1766]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\pepes>python
Python 3.9.13 (tags/v3.9.13:6de2ca5, May 17 2022, 16:36:42) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("holamundo")
holamundo
>>>
```

Figura 10.1: Prueba de instalación de Python

- **OpenCL [27]:** una vez se tiene instalado Python, es posible utilizar el comando pip, este es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python, mediante este comando procederemos a la instalación tanto de OpenCl como de OpenCV posteriormente. Para la instalación de OpenCl solo se deberá introducir el siguiente comando en la terminal:

pip install pyopencl

- **OpenCV [28]:** al igual que para la instalación de la librería anterior, se deberá de utilizar el comando pip para la instalación de OpenCV, esto se hará mediante el uso del siguiente comando:

pip install opencv-contrib-python

- **Tkinter [29]:** se utilizará la herramienta tkinter para el desarrollo de la interfaz gráfica del programa, al igual que PyOpenCl y OpenCV puede instalarse mediante el uso de pip con el siguiente comando:

pip install tk

Una vez instalados dichos elementos se podrá desarrollar cualquier programa Python en el que se precise el uso tanto de OpenCl como de OpenCV. Dentro del IDLE basta con incluir las librerías pyopencl y cv2 dentro del archivo de programación.

10.2. PyopenCl

Como se ha mencionado anteriormente se hará uso de PyOpenCl para el uso de la programación paralela en GPU, este lenguaje [1][3] permite parallelizar núcleos de la CPU, GPU y otros tipos de dispositivos como aceleradores, lo que optimiza en gran manera los tiempos de ejecución de los programas.

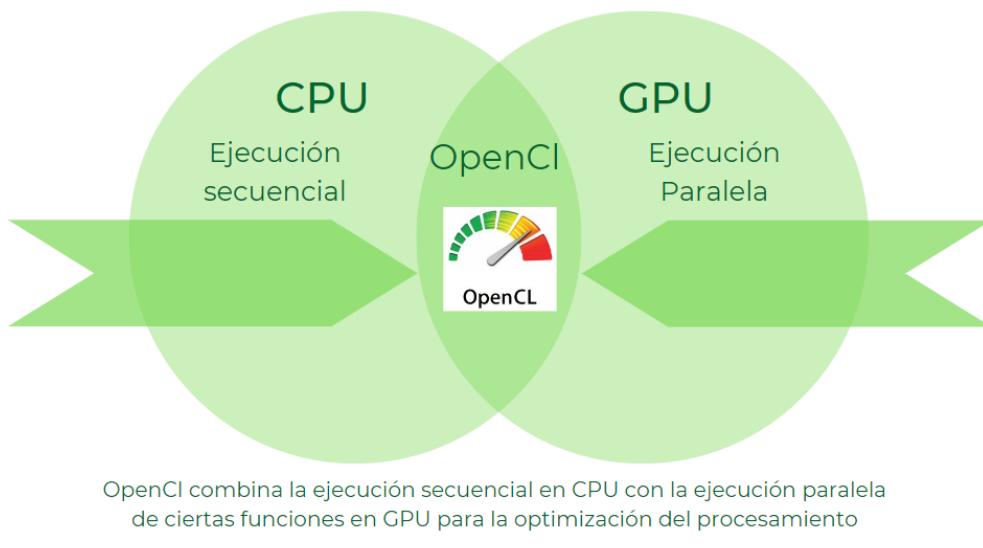


Figura 10.2: Esquema OpenCL

Como se ha indicado en el apartado 9.2, una GPU tiene cientos de procesadores que trabajan con elementos simples y que pueden trabajar de manera simultánea, esto hace que este dispositivo sea ideal para procesar elementos que requieran grandes cargas de cómputo como son los vectores y matrices de grandes dimensiones, o imágenes donde la misma operación puede ejecutarse miles o incluso millones de veces con datos diferentes. Para poder procesar tanta carga de trabajo se debe dividir el programa por parte del programador en

miles de hilos [1][3], y asignarlos a una serie de unidades de cómputo. Dichos hilos en terminología OpenCl son llamados **workers** o **work-items** y se encuentran agrupados en una serie de grupos de trabajo llamados **work-groups**.

Todos los workers [1][3] que se encuentran en un mismo grupo pueden compartir la **memoria local** de la unidad de cómputo, por tanto la sincronización sólo es posible en workers dentro de un mismo grupo de trabajo.

Una vez se ha compilado el programa se asigna un grupo de trabajo a las **unidades de cómputo** y cada una de estas ejecuta workers dentro de un mismo grupo de 32 workers, este concepto ha sido visto con anterioridad en el apartado 9.2 y cuyo nombre son los **warps** [1][3], estos se definen como una agrupación de 32 workers dentro de un mismo work-group que se ejecutan en paralelo. Cuando una unidad de cómputo recibe más de un work-group, esta los separa en warps que posteriormente son programados para su ejecución.

Toda GPU tiene una memoria global [1], que es la de mayor tamaño y todas las transacciones de lectura/escritura con la memoria global son iniciadas por las unidades de cómputo en bloques de 128 bytes de ancho.

Los bloques de memoria que son accedidos en una sola transacción son los llamados **segmentos** [1], este es un concepto muy importante ya que si por ejemplo dos workers del mismo warp acceden a dos datos que están dentro del mismo segmento estos datos son traspasados en una sola transacción lo que ahorra tiempo de cómputo y por tanto ayuda a optimizar mejor el programa.

Por tanto el objetivo que se persigue para tener una máxima optimización es el de que los workers que se encuentren dentro de un

misma warp accedan a posiciones contiguas en memoria de tal forma que se ahorre el mayor número posible de transacciones.

Una vez se han definido estos conceptos se procede a explicar las tres partes principales que componen un programa OpenCl:

- Host: dispositivo en el que se ejecuta el programa de manera secuencial, corresponde con la CPU.
- Device: dispositivo en el que se ejecuta la parte del programa de manera paralela, en este caso corresponde con la GPU.
- Kernel: código que se ejecuta en el device y que es llamado por el Host para su ejecución paralela.

OpenCl es un lenguaje basado en los principios de la programación heterogénea, esto quiere decir que funciona de tal forma que un host se conecta con uno o más dispositivos mediante OpenCl ejecutando en un inicio un programa secuencial que en un momento dado hace una llamada a una función que se ejecuta de forma paralela en la GPU. La siguiente ilustración muestra de manera gráfica este proceso:

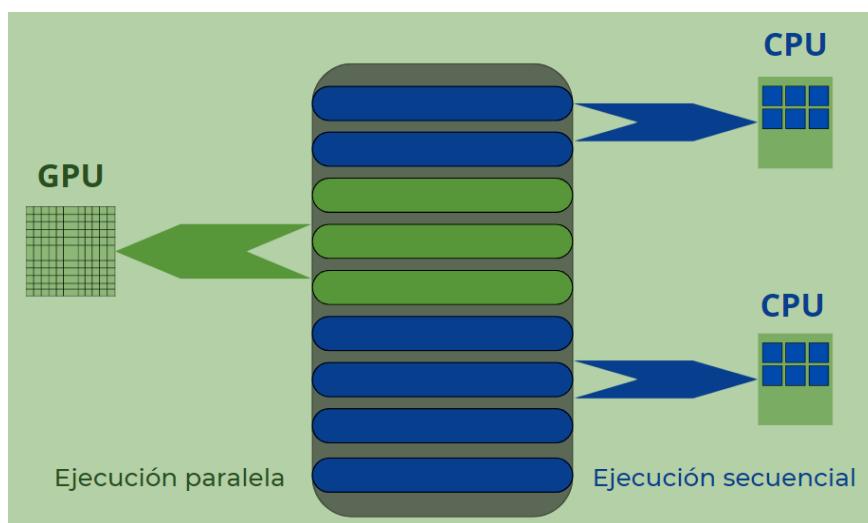


Figura 10.3: Secuencia de un programa OpenCl

Otro concepto fundamental a la hora de trabajar con OpenCl es el **rango N-Dimensional**, este se corresponde con el índice sobre el que se ejecuta la función del kernel, esta función se ejecutará una vez por cada elemento del rango N-Dimensional.

Este puede tener una, dos o tres dimensiones, una dimensión se utilizará a la hora de tratar con estructuras que precisen de un solo índice para recorrer, como pueden ser los vectores, dos dimensiones se utilizan a la hora de tratar con estructuras como las matrices y las tres dimensiones se utilizan a la hora de tratar con elementos de procesado 3D entre otros.

Por último hay dos conceptos a definir a la hora de determinar el número de workers que ejecutarán la función kernel:

- Tamaño global: dicho tamaño se refiere al número de workers que ejecutarán la función kernel, normalmente se establece un worker por cada elemento de la estructura que se esté procesando por lo que si se tiene una imagen de 480x640 píxeles, lo más apropiado será lanzar 307200 workers.
- Tamaño local: este tamaño se refiere al número de workers que se ejecutarán en cada grupo de trabajo, este parámetro es opcional y si no es definido, el propio programa determinará el tamaño más oportuno. Este parámetro es usado cuando se quiere aprovechar al máximo la reducción de transacciones a memoria ya que los elementos de un mismo grupo comparten los elementos de memoria de un mismo segmento.

10.2.1. Pasos a realizar en un programa PyOpenCl

A continuación se mostrarán los pasos a realizar para hacer un programa mediante PyOpenCl [3][4]:

- **Código Host:** el primer paso a realizar en un programa OpenCl es el código del host, en el que se llama a una serie de funciones que inicializan los parámetros necesarios para lanzar un kernel en la GPU. Parte de este código puede ser reutilizable de un programa a otro ya que habrá muy poca variación con respecto a las funciones a llamar.

Para realizar este código se deben seguir los siguientes pasos:

1. Identificación de plataforma y dispositivo en el que se ejecutará la función kernel.

Para ello, se deberá identificar la plataforma, esta normalmente corresponde a un proveedor, responsable de proporcionar la implementación de OpenCl para sus dispositivos, y una vez elegida, se debe seleccionar el dispositivo en el que se quiere ejecutar el programa. Dicha plataforma se puede obtener llamando a la función `get_platforms()`. Esta función no recibe ningún parámetro pero devuelve un vector con las plataformas disponibles para elegir.

Acto seguido tras seleccionar la plataforma adecuada se llamará a la función `get_devices()` desde la que se seleccionará el dispositivo adecuado para ejecutar la función kernel.

2. Creación de un contexto.

Un contexto es un espacio reservado para la gestión de objetos y recursos OpenCl. Este puede ser creado por uno o más dispositivos, y se crea mediante la función **Context()**, los parámetros a definir en esta función son los siguientes:

```
class pyopencl.Context(devices=None, properties=None, dev_type=None,  
cache_dir=None)
```

-Devices: se refiere al dispositivo sobre el que se creará el contexto.

-Properties: es un parámetro opcional que define una serie de propiedades para el contexto.

-dev_type: define el tipo de dispositivo que se está especificando.

-cache_dir: es un parámetro opcional en el que se especifica la dirección de la caché que será utilizada.

3. Creación de la cola de comandos.

Dicha cola permite a los comandos ser enviados a los dispositivos asociados al contexto, estos se colocan en la cola en el orden en el que han sido definidos. Se puede definir mediante la función **CommandQueue()**:

```
class pyopencl.CommandQueue(context, device=None, properties=None)
```

-Context: especifica el contexto creado anteriormente.

-Device: especifica el dispositivo a utilizar para ejecutar la función kernel.

-Properties: es un parámetro opcional que define una serie de propiedades para la cola.

4. Creación de los buffers necesarios para lanzar la función.

Un buffer es el conducto a través del cual se comunican los datos desde la aplicación host hasta los núcleos de la GPU. Estos se crean mediante la función **Buffer()**:

```
class pyopencl.Buffer(context, flags, size=0, hostbuf=None)
```

- Context: especifica el contexto creado anteriormente.
- Flags: se refiere a una serie de propiedades especificadas que dan información sobre el parámetro del que se crea el buffer, en este parámetro se especificará si el buffer es de lectura o escritura.
- Size: es un parámetro opcional con el que se puede especificar un tamaño para el buffer.
- Hostbuf: si este parámetro es especificado, el tamaño predeterminado es el tamaño del buffer especificado.

5. Compilación y lanzamiento de la función kernel.

Una vez se han creado los parámetros necesarios para lanzar una función kernel, se debe de crear y compilar el programa que contiene dicha función. Para ello se utiliza la función **Program()**:

```
class pyopencl.Program(context, src)
```

- Context: contexto creado con anterioridad.

-src: corresponde con una variable que contiene el código de la función kernel.

Para compilar el programa se precisa del uso de la función **Program.build()**, que no recibe ningún parámetro.

Por último se ejecuta la función kernel mediante el uso de **Program.kernel_name()**, esta función recibe los siguientes parámetros:

-CommandQueue: cola de comandos creada con anterioridad.

-GlobalSize: número de workers que ejecutarán la función de manera simultánea.

-LocalSize: parámetro opcional que define el número de workers por cada grupo de trabajo.

-Buffers y variables: corresponde con los buffers creados anteriormente y las variables presentes en la función que no precisen de la creación de un buffer, habrá un parámetro por cada variable que reciba como parámetro la función kernel.

6. Recibir el resultado de la función kernel.

Una vez ha sido lanzada y procesada la función kernel, el resultado obtenido puede recogerse mediante la función **enqueue_copy()**:

```
pyopencl.enqueue_copy(queue, dest, src, **kwargs)
```

-Queue: especifica la cola de comandos creada anteriormente.

-Dest: corresponde con la variable destino en la que será asignado el valor del resultado de la función kernel.

-Src: corresponde con el buffer en el que se ha introducido el valor de la variable resultado de la función kernel.

-kwargs: corresponde con una serie de parámetros opcionales que indican el comportamiento que debe de tener la función.

- Código kernel: la función kernel debe de realizar operaciones relativamente sencillas y su código no debe de ser excesivamente largo ya que será ejecutado una vez por cada worker que se lance por lo que si esta función es muy larga entonces se verá afectado el rendimiento global del programa. Esta parte es la más complicada de realizar correctamente ya que corresponderá con un algoritmo pensado por el programador y orientado a ejecutarse de forma paralela, por lo que es la parte del código más cambiante de un programa a otro.

10.3. OpenCV.

OpenCV es una biblioteca libre de visión artificial que fue desarrollada originalmente por Intel y es conocida como la biblioteca más popular de visión artificial, siendo utilizada actualmente en una gran cantidad de aplicaciones. Con OpenCV tenemos una gran cantidad de posibilidades para el tratamiento de imágenes como la detección de movimiento, el reconocimiento de objetos o la reconstrucción 3D a partir de imágenes entre muchos otros. Además, esta es compatible con la librería de OpenCL. El principal objetivo de OpenCV es el de proporcionar un entorno de desarrollo muy eficiente y con gran facilidad de uso.

Como ha sido mencionado en capítulos previos del proyecto, OpenCV será utilizado para tareas relacionadas con el procesamiento de imagen.

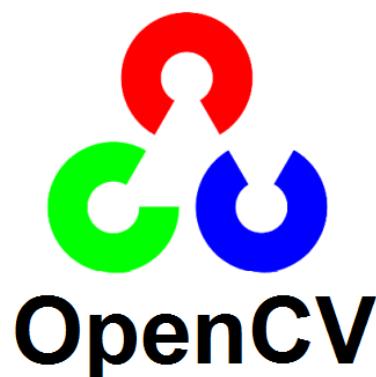


Figura 10.4: Logo de OpenCV

10.4. Tkinter.

Tkinter es una herramienta para el desarrollo de aplicaciones de escritorio multiplataforma. Dicha herramienta está integrada en la biblioteca estándar de Python y dispone de una serie de funciones y una estructura concreta que permite el diseño y creación de interfaces gráficas de una manera sencilla y eficiente.

Las principales ventajas del uso de esta tecnología es que es multiplataforma por lo que el código puede funcionar en distintos sistemas operativos sin dar lugar a ningún error. La siguiente ventaja es, como se ha mencionado anteriormente, su sencillez, permitiendo que programadores que carezcan de gran experiencia en el desarrollo de interfaces gráficas aprendan en un periodo no muy largo de tiempo.



Figura 10.5: Logo de Tkinter

Capítulo 11

Desarrollo del sistema. Programación del Algoritmo.

Durante este capítulo se explicará en detalle el funcionamiento del algoritmo desarrollado y su aplicación a la programación paralela, explicando las distintas fases por las que pasa el programa y su función.

Como se ha mencionado en el capítulo anterior, un programa OpenCl consta de dos partes principales, el código del Host y el código Kernel. Para facilitar el desarrollo modular del programa se ha dividido estas dos partes en archivos distintos, juntándose con un archivo extra que contiene el código de la interfaz gráfica del programa. A continuación se encuentra la explicación detallada de estos archivos:

11.1. Código Host.

Este código se encuentra en un archivo llamado main.py y está formado por diferentes fases que se muestran a continuación:

1. Lectura de imagen y aplicación de filtro. Durante esta fase se lee la imagen mediante la función de OpenCV imread(), se comprueba si tiene el tamaño necesario para su procesamiento (4056*3040), en caso contrario se redimensiona y se le aplica un filtro que convierte la imagen a escala de grises con el fin de tener solo un canal por cada píxel lo que facilita en gran medida su procesamiento.
2. Funciones host de OpenCl. Como se ha explicado en el apartado 10.2.1 para realizar un programa en OpenCl es necesario llamar a una serie de funciones que inicializan los parámetros necesarios para lanzar un kernel. El proceso es el siguiente: se escoge plataforma y dispositivo GPU a utilizar, en este caso una Nvidia GeForce GTX 1050, después se crean contexto y cola de comandos y seguido se procede a crear los buffer, en este caso se crean dos buffers, uno para la imagen en escala de grises que será de solo lectura, y otro para la imagen resultado que será de solo escritura, los demás argumentos de la función no precisan de reserva de memoria. Después se lanza el programa en la GPU y se espera hasta adquirir el resultado.

3. Análisis y conteo de contornos. Durante esta fase se llama a una serie de funciones de OpenCV relacionadas con la identificación y conteo de contornos de una imagen, para ello se utiliza la función `findContours` en la imagen resultado del kernel, esta guarda en un vector los elementos resaltados en la imagen, posteriormente se eliminan los elementos del procesamiento de menor tamaño, lo que borra objetos que son casi imperceptibles en la imagen y que no corresponden con la partícula en cuestión. Por último se llama a una función que computa el tamaño de cada partícula, primero calcula el número de píxeles del interior de cada contorno y a partir de eso y del tamaño de la imagen, averigua el tamaño de cada partícula y el porcentaje total que estas ocupan dentro de la imagen. Esto se guarda en un fichero junto con las imágenes en escala de grises, la imagen resultado y la imagen original con los contornos resaltados.

11.2. Código Kernel.

Este código se encontrará en un archivo llamado kernel.cl, dicha extensión es utilizada para programar los archivos kernel de OpenCl.

Los diferentes inputs que recibe la función kernel son los siguientes:

- **Image**: variable que contiene la imagen a procesar convertida a escala de grises, para disponer en memoria de esta variable se ha creado un buffer de solo lectura.
- **Result**: variable en la que irá el resultado del procesamiento, corresponde con una imagen binaria en la que están resaltados los contornos de la partícula a detectar. Para disponer en memoria de esta variable se ha creado un buffer de solo escritura.
- **Rows**: corresponde con el número de filas de píxeles que contiene la matriz imagen.
- **Cols**: corresponde con el número de columnas de píxeles que contiene la matriz imagen.
- **MaxBtm**: corresponde con el btm máximo asignado a la partícula que se está buscando.

- **MinBtm**: corresponde con el btm mínimo asignado a la partícula que se está buscando.
- **Margin**: corresponde con el margen entre píxel y píxel que se debe tener para considerarse borde.

Como se ha comentado en capítulos previos la funcionalidad que se persigue mediante este algoritmo es la de diferenciar distintos tipos de partículas dentro de una imagen de una muestra de agua tomada con un microscopio electrónico y la característica distintiva que ayudaría a diferenciarlas es el borde, siendo en algunos casos suave y en otros más abrupto.

Para ello se ha creado un algoritmo basado en la comparación de píxeles adyacentes dentro de la imagen y la diferencia entre ellos, de tal forma que hay tres requisitos para considerar a un píxel apto como borde y que tienen relación con los tres conceptos definidos en el capítulo 8, el margen, el btm mínimo y el btm máximo. Dichos requisitos son los siguientes:

- El píxel central debe de tener una diferencia mayor que el margen estimado con al menos uno de sus píxeles adyacentes.
- Si se diera esa diferencia con algún píxel esta debe de mantenerse en los píxeles siguientes hasta alcanzar o superar el btm mínimo.
- El número de píxeles que tienen dicha diferencia no puede ser mayor que el btm máximo.

La siguiente ilustración muestra el proceso de comparación que sigue el algoritmo para cada píxel de manera gráfica, en este caso el píxel central tiene un btm de 2, por lo que es apto para ser considerado

píxel de borde según las circunstancias de margen y btm mínimo y máximo que se muestran en el ejemplo:

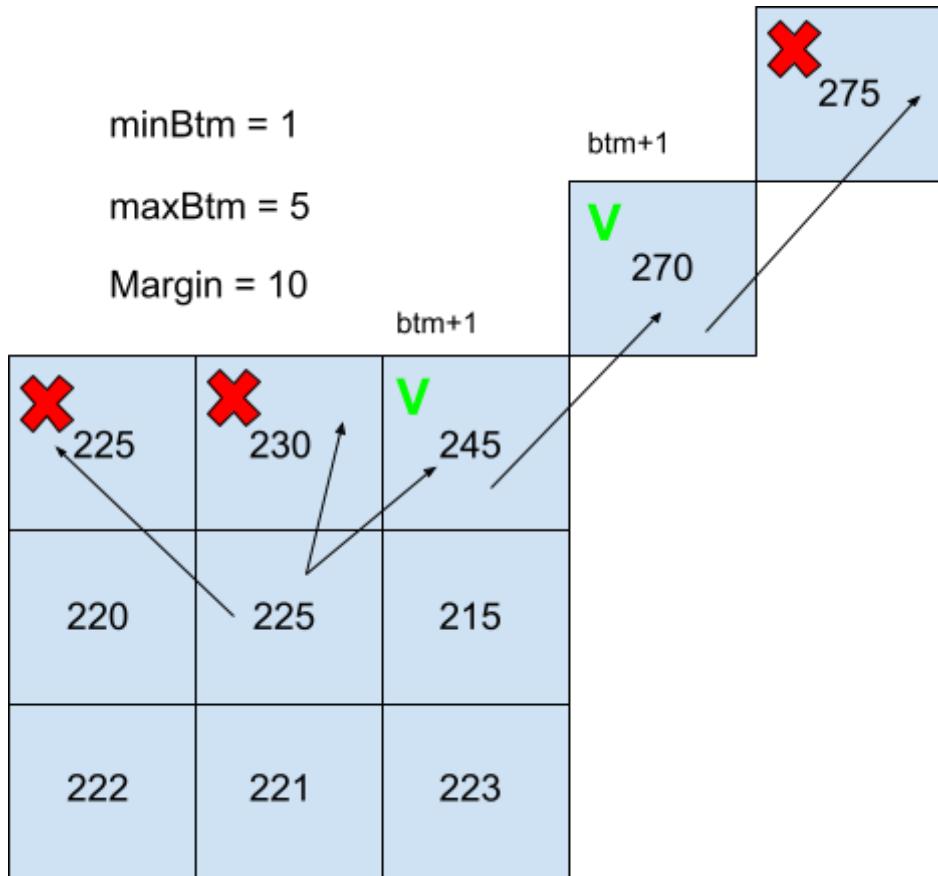


Figura 11.1: Esquema de funcionamiento del algoritmo

Si el píxel supera los tres requisitos para ser apto, se para la comparación y se marca dicho píxel en la matriz resultado, en caso de que no cumpla alguno de los tres requisitos se sigue comparando hasta haber finalizado con todos los píxeles adyacentes. De esta manera al final del procesamiento la función devolverá una matriz binaria (blanco y negro) que resaltará en blanco los píxeles seleccionados.

Los pasos que sigue el código para realizar esto son los siguientes:

1. Se inicializan los dos índices paralelos de los que se dispondrá: cada hilo que entre en la función kernel tendrá dos números asociados que servirán de índice dentro de la matriz, estos se almacenan en dos variables llamadas *i* y *j*, por ejemplo si se quiere acceder a la posición 10,12 de la matriz tendremos que referirnos al hilo que tenga los números *i*=10 y *j*=12. Estos índices se consiguen mediante la función `get_global_id()`.
2. Comienzan dos bucles que recorren los píxeles adyacentes al central.
3. Comienza otro bucle que comprueba si el píxel que estamos comparando tiene una diferencia mayor que el margen estimado con el píxel central, si esto es así se sigue comparando con el píxel de delante a este y se incrementa el btm, una vez no haya una diferencia importante finaliza el bucle y pasa a la siguiente fase.
4. Una vez se tiene el valor del btm por el bucle anterior, este es comparado con el valor de los btm máximo y mínimo, y si este se encuentra entre esos dos valores el píxel es apto para ser considerado borde de la partícula que se está buscando, en caso contrario sigue la comparación con los demás píxeles adyacentes hasta que finalicen los bucles mencionados en el paso dos.

Capítulo 12

Pruebas

Durante este capítulo se procederá a la comprobación del correcto funcionamiento de las características implementadas, para ello se realizarán dos tipos de pruebas, unas que garanticen y verifiquen el correcto funcionamiento del programa y comparando el resultado con otras implementaciones previas y otras que comprueben la optimización de tiempo de ejecución conseguido mediante la programación paralela.

12.1. Pruebas de funcionamiento.

En este apartado se realizarán dos procesos, el primero corresponde con el proceso de verificación, es decir, la comprobación de que el software desarrollado cumple con los requisitos funcionales y no funcionales de su especificación, mientras que el segundo corresponde con el proceso de validación, es decir, comprueba que el software cumple las expectativas esperadas por el cliente.

12.1.1. Pruebas de verificación.

Como se ha mencionado anteriormente durante este apartado se comprobará que tanto los requisitos funcionales como los no funcionales especificados se cumplen en el programa, por lo que se procederá a un repaso de cada uno de ellos comprobando si este está implementado o no en el sistema.

- RF-01: el sistema deberá poder identificar distintos tipos de partículas. Este requisito se encuentra presente en el sistema, puede comprobarse al inicio del mismo ya que al acceder al menú y seleccionar la opción de identificar partícula se dará la opción de seleccionar una de las partículas añadidas al sistema.

- RF-02: el sistema deberá comprobar si el tamaño de las imágenes es el apropiado. Al leer la imagen a procesar el sistema comprueba si el tamaño de la misma es igual a 4056 x 3040 píxeles, si no es así el programa redimensiona la imagen a este tamaño.
- RF-03: se deberá poder introducir nuevos tipos de partícula así como sus parámetros de identificación. Este requisito se cumple mediante la opción presente en el menú principal de añadir partícula, en la que se puede introducir tanto el nombre de la partícula como sus parámetros de btm mínimo, máximo y margen.
- RF-04: se deberá poder modificar los parámetros de identificación de un tipo determinado de partícula. Este requisito se cumple mediante la opción presente en el menú principal de modificar partícula, que permite modificar los parámetros de btm mínimo, máximo y margen de una partícula ya presente en el sistema.
- RF-05: se llevará un registro de los distintos tipos de partículas disponibles a identificar por el sistema. Este requisito se cumple ya que dentro del programa hay un archivo llamado particles.txt dentro de la carpeta files en el que se lleva un registro tanto del nombre de cada partícula añadida al sistema como de sus parámetros de identificación.

- RNF-01: el sistema deberá de estar dividido en módulos repartidos según las distintas partes que lo componen. Este requisito se cumple ya que hay tres módulos principales en el sistema, estos son el archivo principal (main.py) en el que se encuentra tanto el procesamiento de la imagen principal como el código del host, el archivo kernel (kernel.cl) en el que se encuentra el código kernel, y el archivo de interfaz (interface.py), en el que se encuentra el código que muestra la interfaz gráfica del programa.
- RNF-02: el sistema deberá permitir la escalabilidad mediante la posibilidad de realizar modificaciones y ampliaciones del mismo. Este requisito se cumple ya que el código se encuentra dividido en funciones y módulos de tal manera que se puede modificar y ampliar de una manera sencilla.
- RNF-03: el código del sistema deberá de ser comprensible aportando comentarios que faciliten la lectura del mismo. Este requisito se cumple ya que a lo largo del código se encuentran presentes una serie de comentarios que indican la fase en la que se encuentra el programa y la función que tienen determinadas partes del mismo.

12.1.2. Pruebas de validación.

Durante este apartado se realizarán las pruebas de validación, estas comprueban si el software desarrollado cumple con las expectativas esperadas por el cliente, para ello se hará un repaso de la funcionalidad que debe de tener el programa por parte del cliente, se hará un breve repaso del problema planteado y se realizarán una serie de ejecuciones del mismo con diferentes imágenes de la muestra para comprobar si este cumple con las condiciones iniciales. Después se comparará esta implementación con otra realizada anteriormente que identifica los contornos de la imagen mediante el algoritmo de Canny.

Como se ha descrito en capítulos previos, el problema que se nos plantea es el de desarrollar un algoritmo que analice una muestra de agua y realice un conteo del número y la longitud de determinado tipo de partícula. Para comprobar esto nos centraremos en un tipo concreto de micropartícula de gran importancia para el proyecto como son los microplásticos, estos se caracterizan por tener un borde más abrupto si se compara con otro tipo de partículas como pueden ser las fibras naturales.

Se harán tres pruebas, a partir de tres imágenes distintas, la prueba consistirá en hacer una ejecución del programa y comprobar si las partículas identificadas son realmente microplásticos y si hay alguno presente en la imagen que no haya identificado el algoritmo.

- Primera prueba: la imagen escogida para la primera prueba es la siguiente:

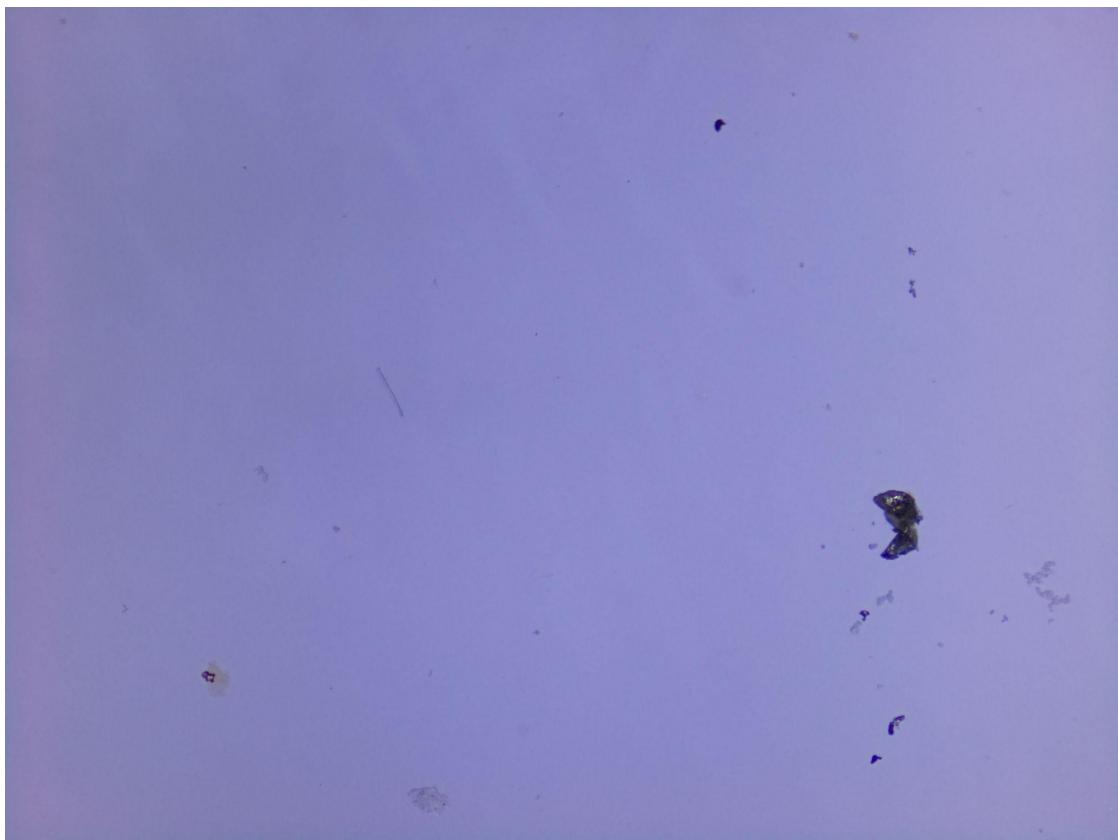


Figura 12.1: Primera prueba (imagen original)

Tras la ejecución del programa se devuelven dos imágenes a parte de la original, una es la matriz resultado del procesamiento en OpenCl, y contiene todos los contornos encontrados por el algoritmo. La segunda imagen contiene los contornos resaltados sobre la imagen original tras la eliminación de las partículas excesivamente pequeñas.

-Matriz resultado:

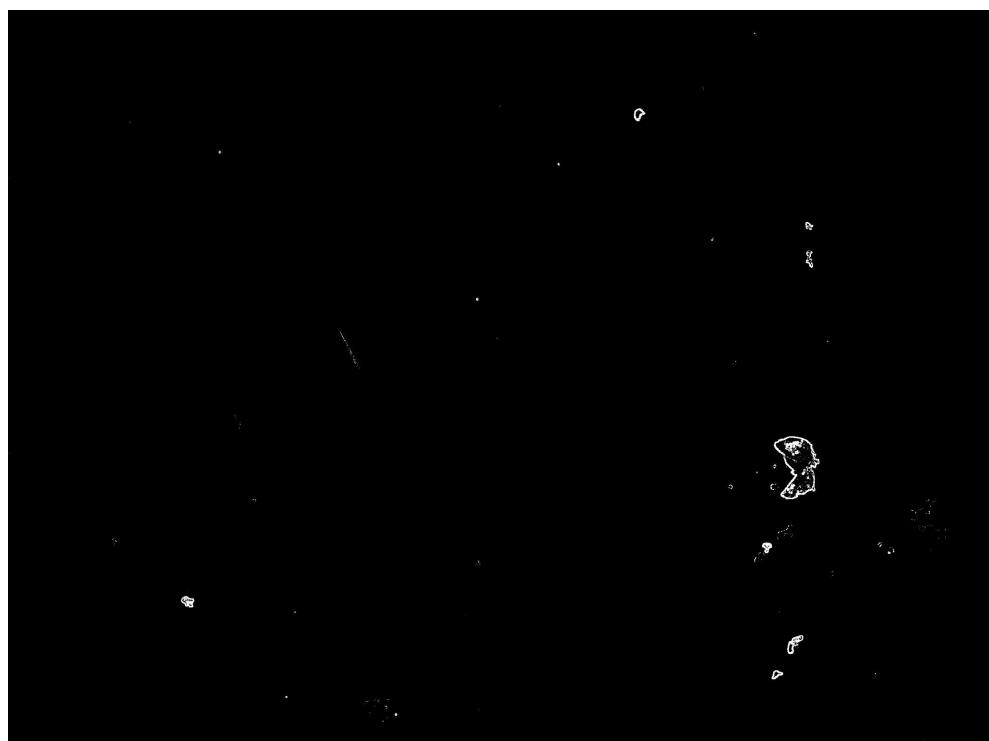


Figura 12.2: Primera prueba (imagen resultado)

-Matriz de contornos:

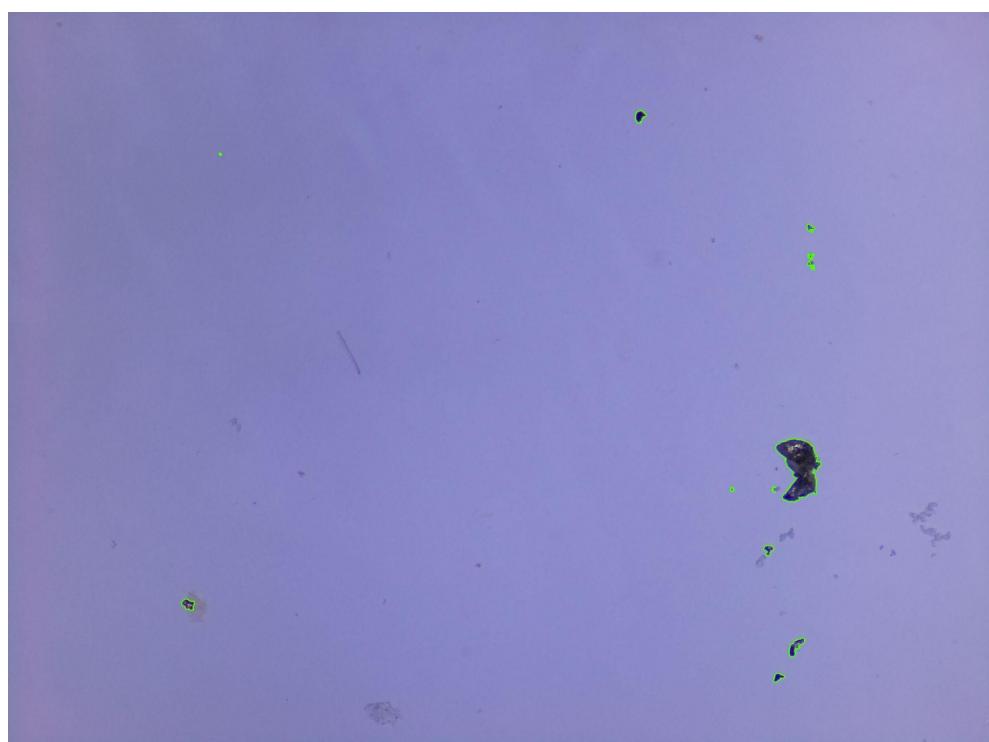


Figura 12.3: Primera prueba (imagen de contornos)

El programa también devuelve un fichero con el número de partículas presentes en la imagen así como el tamaño de cada una.

```
CONTORNO 0: con 1.154 milímetros cuadrados
CONTORNO 1: con 7.035 milímetros cuadrados
CONTORNO 2: con 2.218 milímetros cuadrados
CONTORNO 3: con 1.779 milímetros cuadrados
CONTORNO 4: con 0.969 milímetros cuadrados
CONTORNO 5: con 1.183 milímetros cuadrados
CONTORNO 6: con 140.28 milímetros cuadrados
CONTORNO 7: con 1.308 milímetros cuadrados
CONTORNO 8: con 0.888 milímetros cuadrados
CONTORNO 9: con 0.599 milímetros cuadrados
CONTORNO 10: con 1.902 milímetros cuadrados
CONTORNO 11: con 3.057 milímetros cuadrados
Número de partículas: 12
Total superficie partícula: 162.372 milímetros cuadrados
Porcentaje partículas en imagen: 2.03%
```

Figura 12.4: Primera prueba (resultado de contornos)

Como se puede observar en la imagen el programa indica que hay 12 partículas presentes en la imagen. Previamente a la ejecución del programa se ha realizado un conteo exhaustivo en el que se han identificado 12 partículas de microplásticos y tras ejecutarlo el programa ha identificado correctamente las 12 partículas con éxito por lo que en la primera prueba el porcentaje de éxito es del 100%.

- Segunda prueba: la imagen escogida para la prueba es la siguiente:

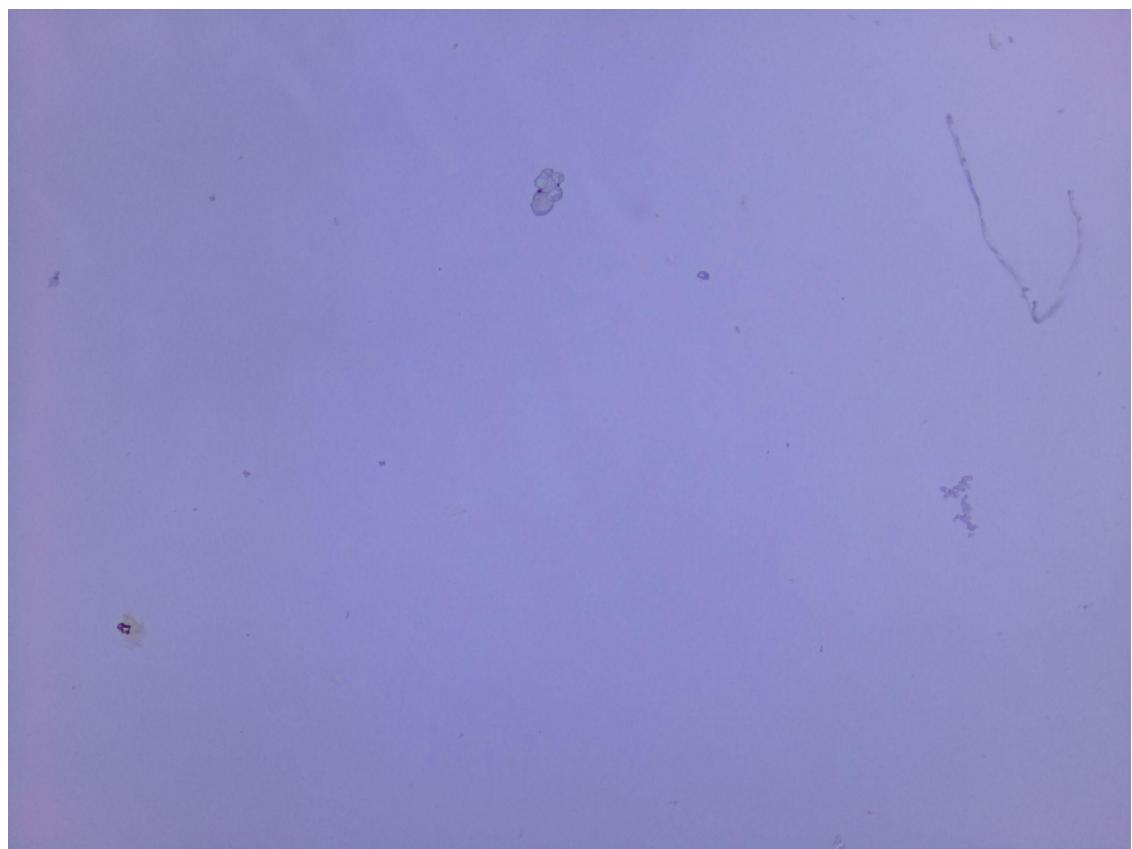


Figura 12.5: Segunda prueba (imagen original)

Tras un conteo exhaustivo han podido ser identificados 4 microplásticos, mientras que la salida del programa es la siguiente:

-Matriz resultado:

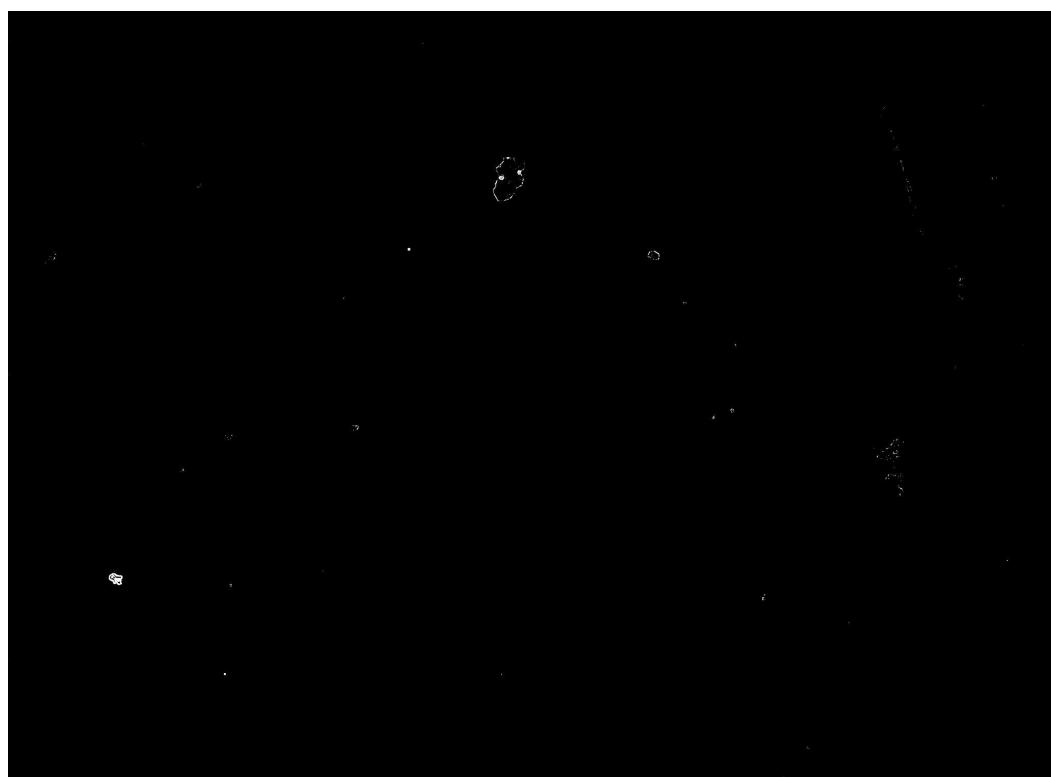


Figura 12.6: Segunda prueba (imagen resultado)

-Matriz de contornos:

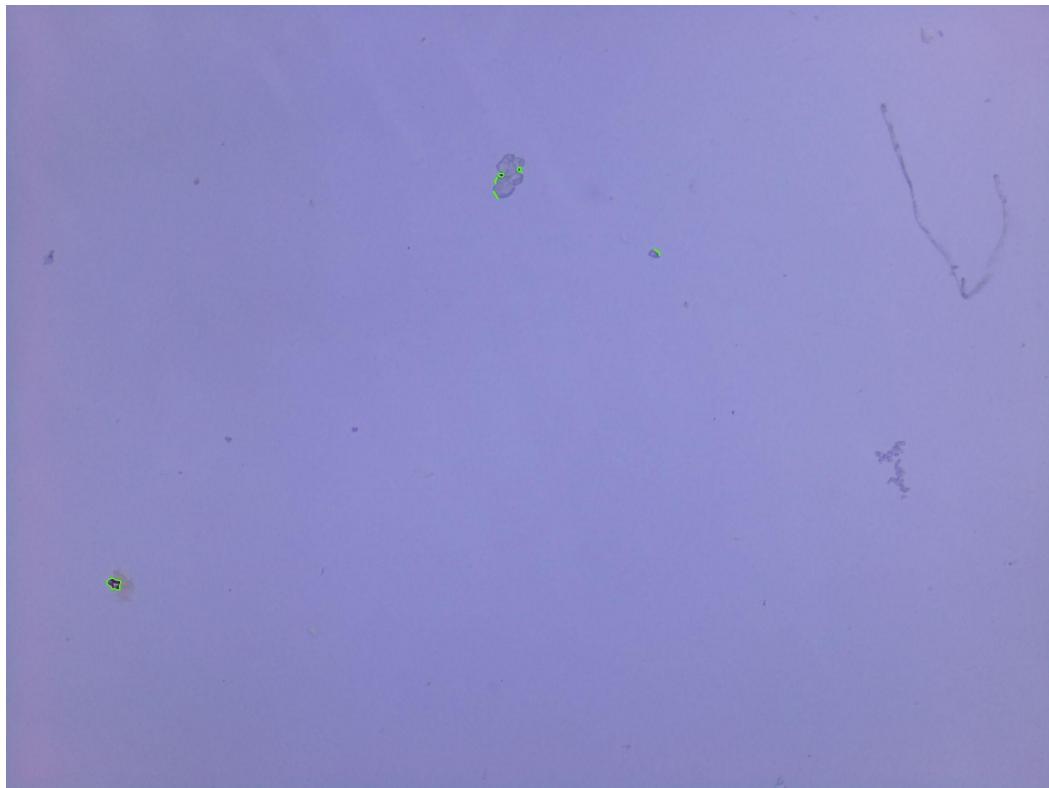


Figura 12.7: Segunda prueba (imagen de contornos)

La salida del fichero de contornos es la siguiente:

```
CONTORNO 0: con 2.388 milímetros cuadrados
CONTORNO 1: con 0.282 milímetros cuadrados
CONTORNO 2: con 0.162 milímetros cuadrados
CONTORNO 3: con 0.82 milímetros cuadrados
CONTORNO 4: con 0.426 milímetros cuadrados
CONTORNO 5: con 0.691 milímetros cuadrados
Número de partículas: 6
Total superficie partícula: 4.769 milímetros cuadrados
Porcentaje partículas en imagen: 0.06%
```

Figura 12.8: Segunda prueba (resultado de contornos)

Como se puede observar se han reconocido dos microplásticos más de los presentes.

- Tercera prueba: la imagen escogida para la prueba es la siguiente:

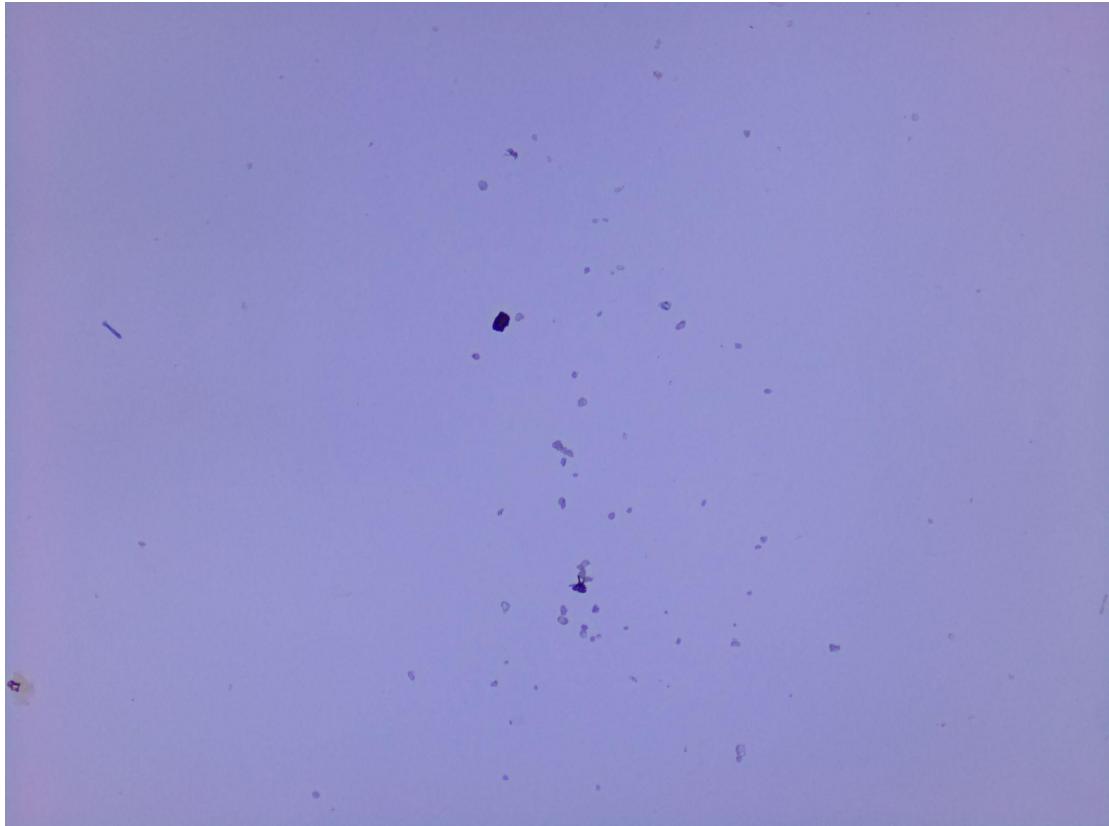


Figura 12.9: Tercera prueba (imagen original)

Tras un análisis exhaustivo han sido identificados 32 microplásticos en la imagen.

El resultado de la ejecución del programa es el siguiente:

-Matriz resultado:

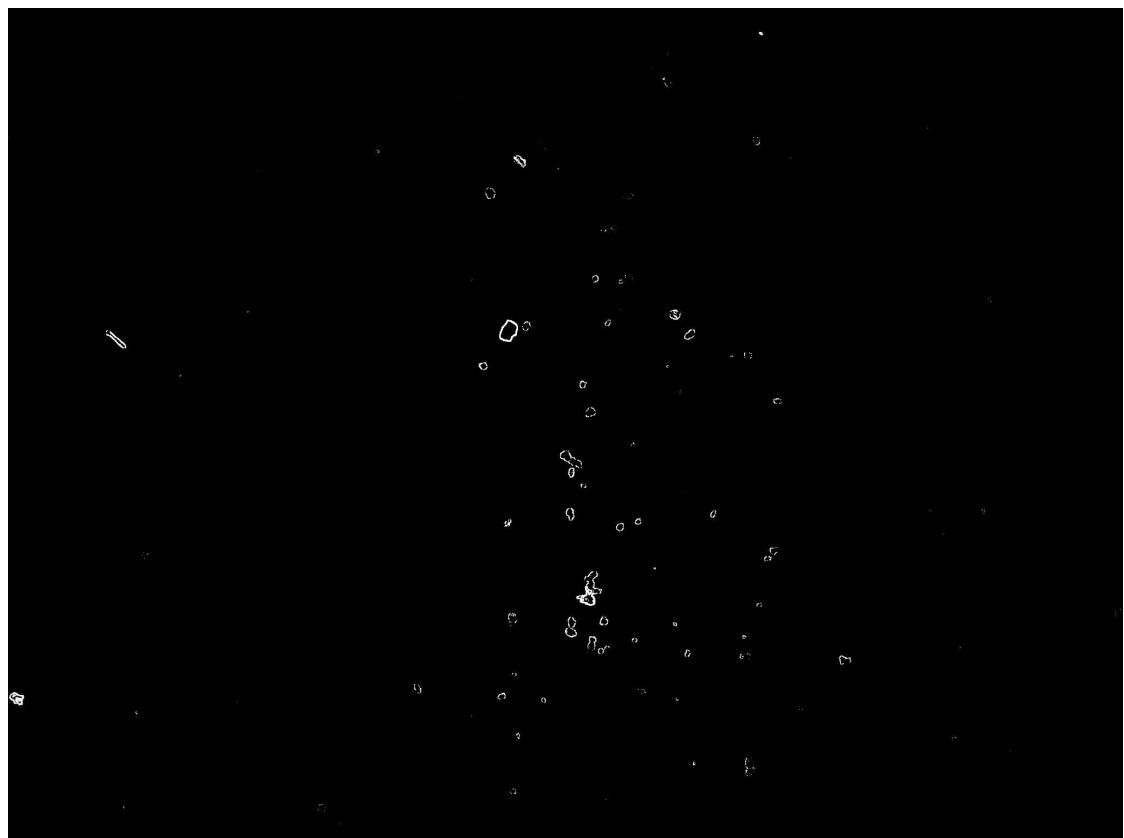


Figura 12.10: Tercera prueba (imagen resultado)

-Matriz de contornos:

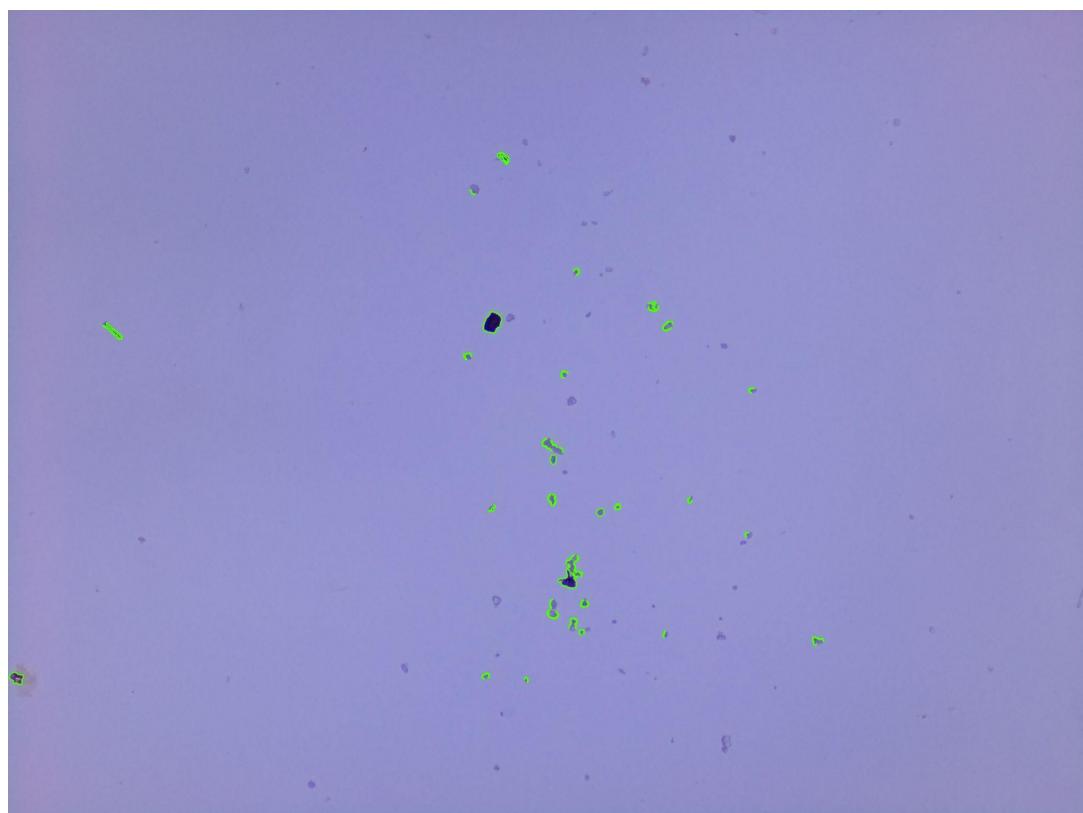


Figura 12.11: Tercera prueba (imagen de contornos)

El contenido del fichero de contornos es el siguiente:

CONTORNO 0: con 0.577 milímetros cuadrados
CONTORNO 1: con 1.093 milímetros cuadrados
CONTORNO 2: con 2.36 milímetros cuadrados
CONTORNO 3: con 3.022 milímetros cuadrados
CONTORNO 4: con 0.747 milímetros cuadrados
CONTORNO 5: con 1.182 milímetros cuadrados
CONTORNO 6: con 4.969 milímetros cuadrados
CONTORNO 7: con 2.651 milímetros cuadrados
CONTORNO 8: con 0.732 milímetros cuadrados
CONTORNO 9: con 1.411 milímetros cuadrados
CONTORNO 10: con 2.894 milímetros cuadrados
CONTORNO 11: con 0.699 milímetros cuadrados
CONTORNO 12: con 11.607 milímetros cuadrados
CONTORNO 13: con 10.047 milímetros cuadrados
CONTORNO 14: con 0.391 milímetros cuadrados
CONTORNO 15: con 0.577 milímetros cuadrados
CONTORNO 16: con 1.193 milímetros cuadrados
CONTORNO 17: con 0.712 milímetros cuadrados
CONTORNO 18: con 1.826 milímetros cuadrados
CONTORNO 19: con 0.812 milímetros cuadrados
CONTORNO 20: con 6.424 milímetros cuadrados
CONTORNO 21: con 1.34 milímetros cuadrados
CONTORNO 22: con 0.248 milímetros cuadrados
CONTORNO 23: con 0.439 milímetros cuadrados
CONTORNO 24: con 2.192 milímetros cuadrados
CONTORNO 25: con 0.773 milímetros cuadrados
CONTORNO 26: con 1.31 milímetros cuadrados
CONTORNO 27: con 1.183 milímetros cuadrados
CONTORNO 28: con 4.174 milímetros cuadrados
CONTORNO 29: con 3.55 milímetros cuadrados
CONTORNO 30: con 5.298 milímetros cuadrados
CONTORNO 31: con 1.105 milímetros cuadrados
CONTORNO 32: con 0.684 milímetros cuadrados
CONTORNO 33: con 1.463 milímetros cuadrados
CONTORNO 34: con 0.22 milímetros cuadrados
CONTORNO 35: con 6.262 milímetros cuadrados

Número de partículas: 36

Total superficie partícula: 86.167 milímetros cuadrados

Porcentaje partículas en imagen: 1.077%

Figura 12.12: Tercera prueba (resultado de contornos)

Por tanto el programa identifica cuatro microplásticos más de los presentes en la imagen.

Se han realizado 4 pruebas más con diferentes imágenes en las que el programa ha demostrado una conducta similar a la vista en las anteriores, el siguiente gráfico muestra una comparación entre el número real de microplásticos presentes en las imágenes probadas y el que muestra el programa:

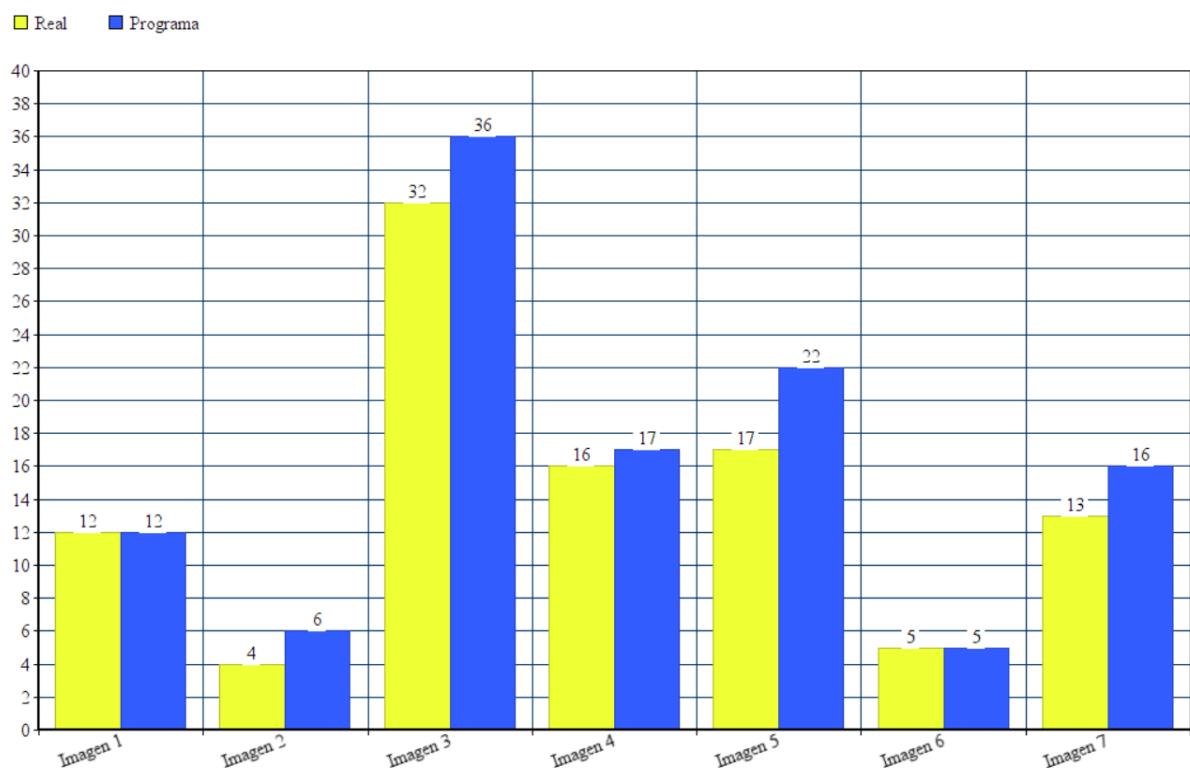


Figura 12.13: Resultado del programa vs real

Se puede observar que en la mayoría de los casos el programa identifica algunas microfibras de más, esto se debe a que algunas fibras que no se encuentran tan claras en la imagen son divididas en partes por lo que se identifica como más de una partícula.

Tras realizar una media de los resultados obtenidos se ha determinado que el programa identifica un 8.34% más de fibras que las que se

encuentran presentes en la muestra, basándonos en la muestra anterior de 7 imágenes.

A continuación se compararán los resultados obtenidos con esta distribución del programa con otra que utiliza el algoritmo de Canny para la identificación de elementos, para ello se mostrará un gráfico como el anterior con los resultados obtenidos por el algoritmo de Canny comparados con el número real de microplásticos presentes en la imagen:

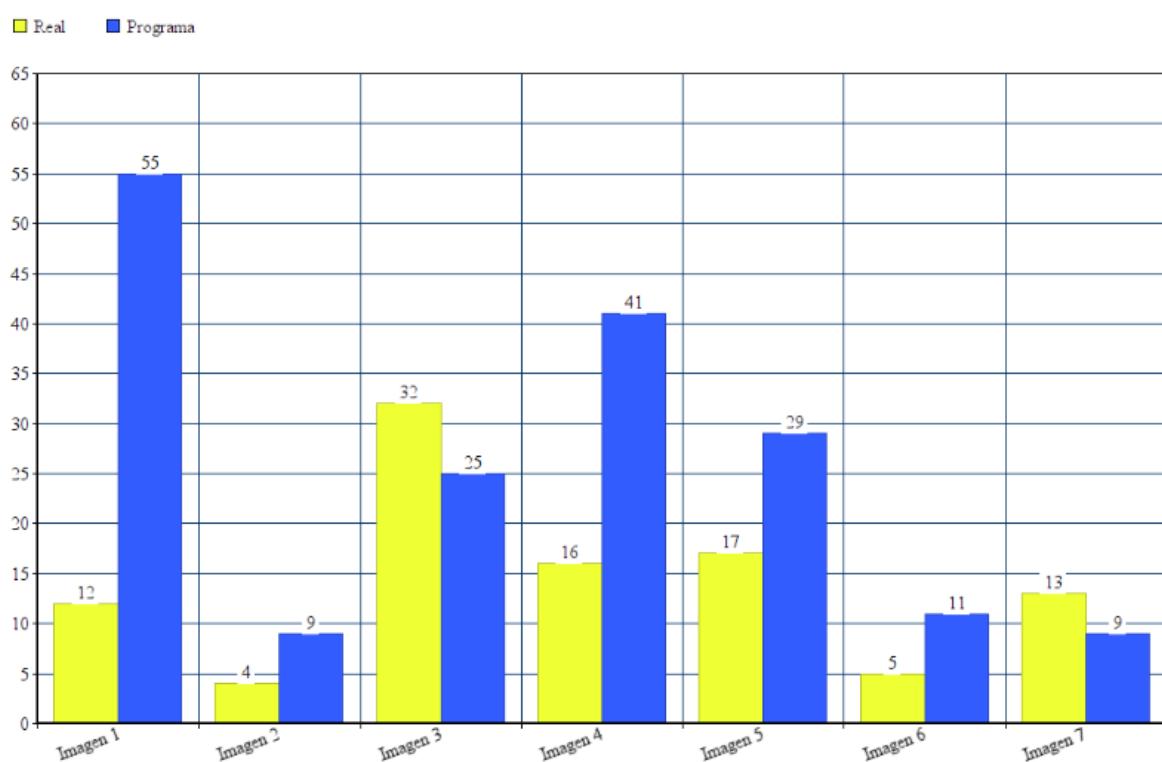


Figura 12.14: Resultados Algoritmo de Canny vs real

Tras realizar una media de los resultados obtenidos se puede observar que esta implementación del programa identifica un 81% más de fibras que las que realmente están presentes en la imagen, esto se produce por el exceso de elementos identificados por este algoritmo al no hacer ninguna distinción de borde por lo que la primera implementación es mucho más precisa a la hora de realizar esta tarea.

12.2. Pruebas de tiempo.

Durante este apartado se realizarán una serie de pruebas que medirán el tiempo de ejecución. Estas serán realizadas únicamente al algoritmo ya que lo que se pretende es comparar la implementación del algoritmo programado de manera secuencial con la realizada mediante programación paralela y analizar las diferencias entre las mismas.

Para ello se realizarán 5 pruebas a cada implementación, y se medirán los tiempos de ejecución para posteriormente mostrar la tendencia de los tiempos de cada una según se sube el tamaño de la imagen a procesar.

La imagen escogida para realizar las pruebas es la siguiente:

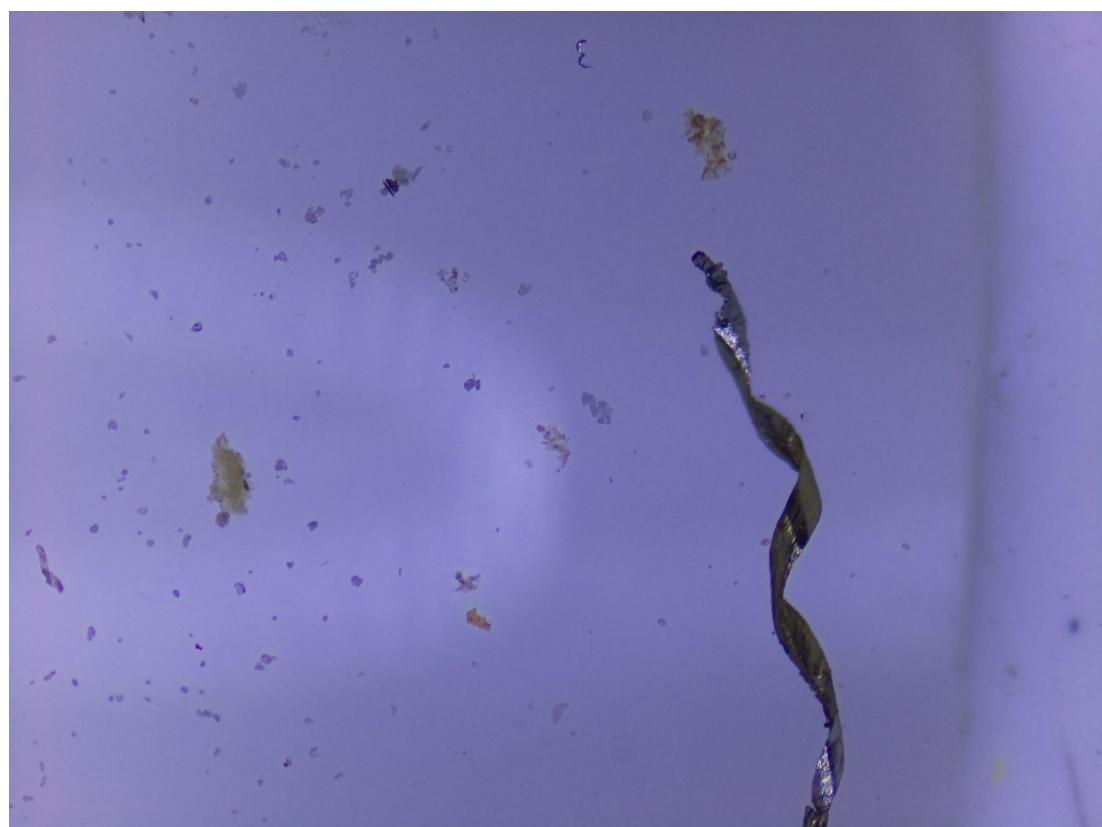


Figura 12.15: Imagen para prueba de tiempos

Los tiempos obtenidos según diferentes tamaños de imagen son los siguientes:

- 640 x 480 píxeles:

-Implementación paralela:

El tiempo de ejecución es: 0.0019271373748779297 segundos

-Implementación secuencial:

El tiempo de ejecución es: 1.0493483543395996 segundos

- 1280 x 960 píxeles:

-Implementación paralela:

El tiempo de ejecución es: 0.0029926300048828125 segundos

-Implementación secuencial:

El tiempo de ejecución es: 4.760015249252319 segundos

- 2028 x 1520 píxeles:

-Implementación paralela:

El tiempo de ejecución es: 0.003989219665527344 segundos

-Implementación secuencial:

El tiempo de ejecución es: 12.90911054611206 segundos

- 4056 x 3040 píxeles:

-Implementación paralela:

El tiempo de ejecución es: 0.00895380973815918 segundos

-Implementación secuencial:

El tiempo de ejecución es: 50.2353732585907 segundos

- 8112 x 6080 píxeles:
 -Implementación paralela:

El tiempo de ejecución es: 0.03023505210876465 segundos

- Implementación secuencial:

El tiempo de ejecución es: 202.64594292640686 segundos

Si se hace una comparación de los tiempos obtenidos es sencillo de observar la gran diferencia que se encuentra presente entre la programación paralela y la secuencial, y por tanto de esta forma se puede ver la gran eficiencia que puede conseguirse en un programa con esta manera de programar.

El siguiente gráfico muestra la tendencia que siguen los tiempos de ambas implementaciones, mostrando los segundos que ha tardado el algoritmo en ejecutarse en el eje Y y la resolución de las imágenes utilizadas para la prueba en el eje X:

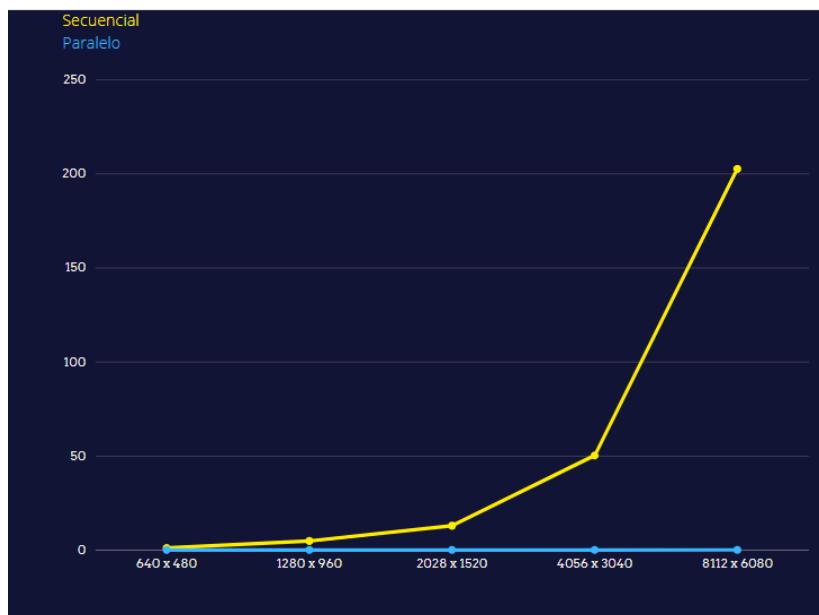


Figura 12.16: Resultados tiempo paralelo vs secuencial

Ambas distribuciones tienen una tendencia exponencial conforme sube el tamaño de la imagen pero puede apreciarse una enorme diferencia entre una distribución de otra, ya que con la implementación paralela pueden procesarse imágenes de tamaños considerablemente grandes sin tener un cambio notable en el tiempo de ejecución desde el punto de vista del usuario, mientras que la secuencial muestra tiempos medianamente aceptables para imágenes de poco tamaño pero conforme sube la resolución el tiempo tiende a ser más largo de lo conveniente en un programa de estas características.

Gracias a los resultados de estas pruebas se puede observar lo ventajoso que puede ser trabajar con programación paralela para procesar elementos que precisen mucha computación como pueden ser las imágenes, haciendo que se pueda sacar provecho del hardware del que se dispone de una manera sencilla y eficiente y permitiendo que el programa pueda ser ejecutado y lo haga en un tiempo aceptable sin importar la potencia de los recursos del ordenador en el que se ejecuta.

Capítulo 13

Conclusiones

Para finalizar se realizarán una serie de conclusiones sobre todo lo visto a lo largo del proyecto, repasando si se han cumplido los objetivos definidos, sintetizando el conocimiento aprendido y proponiendo futuras propuestas que ayuden a la expansión y mejora del mismo.

13.1. Conclusiones sobre los objetivos propuestos.

En este apartado, se hace un repaso de los objetivos propuestos al inicio del proyecto, analizando si han sido cumplidos o no. Para ello se muestra una tabla que indica la consecución del objetivo y algunas observaciones sobre el mismo.

Objetivo	Consecución	Observaciones
OB-01	Sí	Se han estudiado y aprendido los distintos formatos de imagen existentes para su procesamiento en un sistema computador, ventajas e inconvenientes de cada uno de ellos. Se han analizado

		cómo influyen los distintos filtros que se pueden aplicar y software de tratamiento de imágenes existente tanto comercial como utilizado en investigaciones actuales.
OB-02	Sí	<p>Se han estudiado conceptos de procesamiento paralelo y aprendido a implementar programas paralelos y computación heterogénea.</p> <p>En concreto, se ha aprendido en profundidad el conocimiento de unidades de procesamiento gráfico (GPU) y la programación paralela OpenCl.</p>
OB-03	Sí	<p>Se ha estudiado y analizado la caracterización de microplásticos y microfibras naturales presentes en una imagen. Identificando bordes y la posible difuminación de un objeto en la imagen para discriminar si se trata de un tipo de material u otro.</p>
OB-04	Sí	<p>Se ha desarrollado un algoritmo en OpenCl para el procesamiento de imágenes basado en el análisis de cada píxel con sus adyacentes que pretende identificar tipos de bordes distintos en los objetos de la imagen para poder discriminar el tipo de material presente en la imagen.</p>
OB-05	Sí	<p>Se ha optimizado el nuevo algoritmo mediante el uso de la programación paralela con OpenCL para hacer un uso eficiente de la memoria y el sistema hardware de GPU's.</p>

OB-06	Sí	Se ha probado la calidad del nuevo algoritmo comparando su rendimiento con el del algoritmo de Canny y teniendo en cuenta el número y tipo de partículas detectadas en la imagen por cada algoritmo.
OB-07	Sí	Se ha configurado y documentado el entorno proporcionando información útil para la implementación de herramientas de programación en GPU con OpenCl en proyectos con necesidades similares.

13.2. Conceptos aprendidos.

Durante este apartado se nombrarán los conceptos y tecnologías utilizadas que eran desconocidas al inicio y que han servido tanto para el desarrollo del proyecto como para el personal, lo que aporta experiencia y conocimiento de cara a la realización de proyectos futuros que consten de unas características similares a este.

- **Procesamiento de imagen:** este campo ha sido fundamental a la hora de formular el proyecto ya que para poder realizar tareas relacionadas con imágenes es necesario conocer una serie de conceptos que han ayudado en gran medida a la hora de pensar el algoritmo y que no se sabían previamente. También se ha utilizado por primera vez la herramienta de OpenCV la cuál ha facilitado el uso de diversas técnicas relacionadas con el procesamiento de imagen permitiendo adaptar la imagen a las necesidades requeridas por el algoritmo.
- **Programación paralela:** si bien la herramienta OpenCl ya era conocida, este proyecto ha ayudado en gran medida a aprender conceptos nuevos que abren fronteras ante los numerosos usos que se le pueden dar a la misma.
- **Lenguajes de programación:** se ha utilizado por primera vez el lenguaje Python, este fue escogido debido a su gran sencillez y legibilidad, lo que permite realizar proyectos como este de una manera mucho más eficaz que con otros lenguajes.

13.3. Conclusión general.

La intención inicial del proyecto era profundizar en un tema interesante como es el de la programación paralela y sus ventajas en ciertos tipos de programas. Esto daba lugar a una gran cantidad de posibles implementaciones, pero la pretensión que se tenía era la de mostrar la eficacia que obtenemos con este tipo de programación a la vez que se desarrollaba un programa útil que se pudiera utilizar en la vida real para desarrollar algún tipo de función. Esto hizo que se introdujera dos conceptos igual de interesantes y que son de gran importancia en nuestros días como son el procesamiento de imagen y el cuidado del medio ambiente.

Junto con los cambios en formato digital de contenido multimedia surgió la necesidad de desarrollar nuevas herramientas que pudieran cambiar tanto el aspecto como las propiedades de fotografías, vídeos y otro tipo de contenido, esto dió pie a nuevas formas de creación que permiten la realización de proyectos visuales como los que se pueden ver hoy en día. Actualmente es un tema sumamente estudiado y sigue siendo tendencia en el desarrollo de programas, por lo que su estudio ha sido de gran interés y sin duda servirá para futuros proyectos.

Otro de los temas actuales en los que más se está focalizando es el cuidado del medio ambiente y la búsqueda de nuevas tecnologías que permitan crear un entorno sostenible y compatible con el modo de vida que llevamos los seres humanos hoy en día. Por ello, la idea de realizar este programa era sumamente interesante y se espera que sirva para obtener más información sobre la cantidad de fibras que se encuentran presentes en una zona determinada de agua y ayude a la limpieza de las mismas.

Al final no solo se han cumplido los objetivos que se pretendían al inicio, lo que aporta una gran satisfacción, sino que además ha servido para aprender numerosos conceptos que amplían el conocimiento personal y que podrán ser utilizados en futuros proyectos.

Además, aunque no era objetivo de este trabajo, se ha realizado la compilación del software desarrollado como aplicación de escritorio con un interfaz sumamente sencillo que únicamente facilita cargar la imagen a procesar, elegir el tipo de partícula a detectar y mostrar los resultados. En el anexo III y IV se aporta la información correspondiente a esta pequeña aplicación.

13.4. Futuras mejoras.

Como futura mejora se propone incrementar la precisión del algoritmo teniendo en cuenta otra serie de parámetros como puede ser la forma de las partículas, observando la dirección a la que tienden los píxeles mediante el análisis de las adyacencias entre ellos. Esta característica puede ser distintiva para ciertos tipos de partículas, y combinada con la diferenciación de bordes presente en el algoritmo actual puede dar lugar a un análisis más acertado que el que se consigue actualmente.

Por otra parte también se plantea una mejora de la interfaz gráfica del programa añadiendo nuevas opciones y transformando el aspecto y diseño de la misma para una mejor experiencia de usuario.

Por último se pretende mejorar el algoritmo de tal forma que sea capaz de distinguir partículas que se encuentren superpuestas dentro de la imagen y de esta forma mejorar aún más la precisión del mismo.

Bibliografía

[1] Introduction to Parallel Computing. Ed.: Springer. Autores: Roman Trobec, Boštjan Slivnik, Patricio Bulić, Borut Robič.

[2] Multicore and GPU Programming An Integrated Approach. Ed.: Morgan Kaufmann. 2nd Edition – Feb., 2022. Autor: G. Barlas. ISSN: 9780128141212

[3] PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Aut: Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, Ahmed Fasih, 2012. Parallel Computing, Volume 38, Issue 3, Pages 157-174, ISSN: 0167-8191, doi: 10.1016/j.parco.2011.09.001.

[4] Documentación PyOpenCl. <https://documentacion.de/pyopencl/> . Consultado el 27/06/2022

[5] Profesional Review. Qué es OpenCl y su importancia en GPUs. <https://www.profesionalreview.com/2021/01/02/opencl-que-es/>. Consultado el 10/06/2022.

[6] Modelo de memoria OpenCl.
<https://www.mql5.com/es/articles/407#:~:text=La%20memoria%20constante%20en%20el,es%20%C3%BAnico%20para%20cada%20dispositivo>. Consultado el 05/06/2022.

[7] About OpenCV. <https://opencv.org/about/> .Consultado el 11/06/2022.

[8] Arquitectura de un microprocesador.

<https://www.aboutespanol.com/que-es-la-arquitectura-de-un-procesador-841131> . Consultado el 21/06/2022.

[9] Sobre los núcleos en CPU.

<https://impactotic.co/chipset-nucleos-y-por-que-es-importante-su-cantidad/> . Consultado el 19/06/2022.

[10] Profesional review. Niveles de la memoria caché.

<https://www.profesionalreview.com/2019/05/02/memoria-cache-l1-l2-y-l3/#:~:text=hasta%201024%20KB.-,Memoria%20cach%C3%A9%20L3,s%20y%2011%20ns%20de%20latencia> . Consultado el 21/05/2022.

[11] Technical city. Especificaciones técnicas Nvidia GTX 1050.

<https://technical.city/es/video/GeForce-GTX-1050> . Consultado el 18/06/2022.

[12] Ozeros. Análisis de la arquitectura Pascal.

<https://www.ozeros.com/2016/05/analisis-a-fondo-arquitectura-gpu-nvidia-pascal-disenada-para-la-velocidad/#:~:text=Arquitectura%20Pascal,-La%20arquitectura%20Pascal&text=Consiste%20de%207.2%20mil%20millones,en%20frecuencia%20y%20eficiencia%20energ%C3%A9tica> . Consultado el 22/06/2022.

[13] Modelo de memoria de la GPU.

<https://blitzman.gitbooks.io/cuda/content/modelo-de-memoria.html> . Consultado el 22/06/2022.

[14] Niveles de optimización de software.

<https://www.go4it.solutions/es/blog/los-6-niveles-de-optimizacion-de-software-mejora-tu-codigo> . Consultado el 14/06/2022.

[15] Paradigmas de la programación paralela. http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html . Consultado el 15/06/2022.

[16] Simplilearn. About image processing. <https://www.simplilearn.com/image-processing-article> . Consultado el 11/06/2022.

[17] Cómo afecta y beneficia la tecnología al medio ambiente. <https://www.ecologaverde.com/como-afecta-la-tecnologia-al-medio-ambiente-1205.html#:~:text=La%20tecnolog%C3%ADa%20permite%20mayores%20conocimientos,energ%C3%ADa%20solar%20o%20la%20e%C3%B3tica> . Consultado el 13/06/2022.

[18] Fundamentos básicos del procesamiento de imágenes. <https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap2.html> . Consultado el 23/06/2022.

[19] Introducción a las imágenes digitales. <http://alojamientos.us.es/gtocoma/pid/tema1-1.pdf> . Consultado el 24/06/2022.

[20] Xataka. GPU como pasado, presente y futuro de la computación. <https://www.xataka.com/componentes/las-gpu-como-pasado-presente-y-futuro-de-la-computacion> . Consultado el 14/06/2022.

[21] Profesional Review. Arquitectura GPU, todo lo que necesitas saber. <https://www.profesionalreview.com/2022/05/22/arquitectura-gpu/> . Consultado el 15/06/2022.

[22] Wikipedia. Algoritmos de detección de objetos. https://es.wikipedia.org/wiki/Detecci%C3%B3n_de_objetos . Consultado el 16/06/2022.

[23] Wikipedia. Algoritmo de Canny. https://es.wikipedia.org/wiki/Algoritmo_de_Canny#:~:text=El%20algoritmo%20de%20detecci%C3%B3n%20de,respecto%20a%20la%20versi%C3%B3n%20original . Consultado el 16/06/2022.

[24] LearnOpenCV. Detección de bordes mediante OpenCV. <https://learnopencv.com/edge-detection-using-opencv/> . Consultado el 17/06/2022.

[25] Introducción al lenguaje Python. <https://www.cursoaula21.com/que-es-python/> . Consultado el 10/11/2021.

[26] Instalación de Python en Windows. <https://www.python.org/downloads/windows/> . Consultado el 09/11/2021.

[27] Instalación de PyopenCl. <https://pypi.org/project/pyopencl/> . Consultado el 09/11/2021.

[28] Instalación de OpenCV en Python. <https://omes-va.com/installacion-de-python-y-opencv-en-windows/> . Consultado el 09/11/2021

[29] Instalación de tkinter en Windows. <https://es.acervolima.com/como-instalar-tkinter-en-windows/> . Consultado el 13/05/2022.

[30] GitHub. Repositorio de código del programa.
<https://github.com/i82sayuj/Trabajo-de-Fin-de-Grado>.

[31] Mega. Repositorio con el ejecutable de la aplicación.
<https://mega.nz/folder/wIIGzI7R#iyQ8EPZyoElPt9wQTbEjUA>.

Parte IV

Anexos

Anexo I

Implementación del algoritmo
Manual de código (host y kernel)

Código principal (host)

En este código se encuentran la lectura de imagen, la aplicación de filtros a la misma, la parte de host de la función OpenCl en el que se preparan los parámetros necesarios para el lanzamiento de la misma, como el conteo y análisis de las partículas encontradas como resultado de la función OpenCl.

```
import pyopencl as cl
import cv2 as cv
import numpy as np
import sys
import time
import re

def SetContoursVector(mat):
    ret,thresh = cv.threshold(mat,127,255,0)
    contours,hierarchy =
    cv.findContours(thresh,cv.RETR_EXTERNAL,cv.CHAIN_APPROX_NONE)
    return contours

def splitAxis(contourList,i):
    contours2 = np.empty((len(contourList[i]),2),int)
    for j in range(len(contourList[i])):
        contours = str(contourList[i][j]).split(sep = ' ')
        k = 0
        while( k < len(contours)):
            if(contours[k] == ""):
                contours = np.delete(contours,k)
                k = k-1
            k = k+1

    if(len(contours) == 3):
```

```

        contours[0] = contours[1]
        contours[1] = contours[2]
    contours2[j][0] = re.sub('[',",contours[0])
    contours2[j][1] = re.sub(']',",contours[1])
return contours2

def middlePoints(contours):
    pixels = np.zeros((len(contours)),dtype = np.float64)
    for i in range(len(contours)):
        contourAxis = splitAxis(contours,i)
        for j in range(len(contourAxis)):
            current = contourAxis[j][0]
            for k in range(len(contourAxis)):
                if(current == contourAxis[k][0] and j!=k):
                    pixels[i] = pixels[i] + (abs(contourAxis[j][1]-contourAxis[k][1])/2)
    return pixels

def ShowContours(contours,tam):
    f = open('output\\contours.txt', 'w')
    totalSuperficieParticula = 0
    for i in range(len(contours)):
        tamanoParticula = contours[i]*8000/tam
        tamanoParticula = round(tamanoParticula,3)
        totalSuperficieParticula = totalSuperficieParticula +tamanoParticula
        particle = 'CONTORNO '+str(i)+': con '+str(tamanoParticula)+' milímetros
cuadrados'
        f.write(particle+ '\n')

    f.write('Número de partículas: '+str(len(contours)) + '\n')
    totalSuperficieParticula = round(totalSuperficieParticula,3)
    f.write('Total superficie partícula: ' + str(totalSuperficieParticula) +
'milímetros cuadrados \n')
    f.write('Porcentaje partículas en imagen: ' +
str(round(totalSuperficieParticula*100/8000,3)) + '%\n')

def DeleteSmallObjects(contours):

```

```

delete = []
for i in range(len(contours)):
    if(len(contours[i]) < 55):
        delete.append(i)

contours2 = np.delete(contours,delete,0)
return contours2

def settingArguments(nombre):
    f = open('files\\particles.txt',"r")
    found = 0
    a = 0
    for i in f:
        line = i.split(sep=',')
        if(nombre == line[0]):
            a = np.array([line[1],line[2],line[3]])
            break

    return a

def findParticle(number):
    f = open('files\\particles.txt', 'r')
    n = 0
    for i in f:
        line = i.split(sep=',')
        if(np.intc(number)== n):
            return line[0]
        n = n+1

    return 0

```

```

#-----main program-----#
#read image and set standard size
image = cv.imread(sys.argv[2])
if(sys.argv[2] == ""):
    print("Error, debe seleccionar una ruta o la ruta seleccionada no existe.")
    exit()

nombre = findParticle(sys.argv[1])
if(nombre == 0):
    print("Error, la partícula especificada no existe o no se ha especificado")
    exit()

#Standard size: 4056*3040
tam = 4056*3040
size = (4056,3040)

pixels = image.shape[0]*image.shape[1]

if(pixels != tam):
    print("Tamaño de imagen distinto al adecuado, redimensionando...")
    image = cv.resize(image,size)

#gray scale transforming

gray = cv.cvtColor(image,cv.COLOR_BGR2GRAY)

#-----OpenCL-----#
#available platforms
print(cl.get_platforms())

```

```

#choose platform and device

platform = cl.get_platforms()[0]
gpu = cl.get_platforms()[0].get_devices()
print(gpu)

#creating context

context = cl.Context(
    dev_type = cl.device_type.ALL,
    properties= [(cl.context_properties.PLATFORM, platform)])
```

#creating command queue

```
queue = cl.CommandQueue(context)
```

#Creating buffers

```
mf = cl.mem_flags
```

```
imageBuffer = gray.astype(np.uint8)
```

#image buffer and other arguments

```
a_g = cl.Buffer(context,mf.READ_ONLY | mf.COPY_HOST_PTR,
hostbuf=imageBuffer)
```

```
a = settingArguments(nombre)
```

```
minBtm = np.intc(a[0])
maxBtm = np.intc(a[1])
margin = np.intc(a[2])
```

```
rows = np.intc(imageBuffer.shape[0])
cols = np.intc(imageBuffer.shape[1])
```

```

#result vector buffer
b_g = cl.Buffer(context,mf.WRITE_ONLY , imageBuffer.nbytes)

#creating program
src = ".join(open('build\\scripts-3.9\\kernel.cl').readlines())

program = cl.Program(context,src).build()

#setting arguments and launching program
start = time.time()
program.btm(queue,imageBuffer.shape,None, a_g,
b_g,rows,cols,maxBtm,margin,minBtm)
end = time.time()

#getting results

print('El tiempo de ejecución es: ', end-start, 'segundos')
result = np.empty_like(imageBuffer)
cl.enqueue_copy(queue,result,b_g)

```

#-----OpenCV-----#

```

#set contours vector
contours = SetContoursVector(result)
contours = np.array(contours,dtype = object)
contours = DeleteSmallObjects(contours)
pixels = middlePoints(contours)
ShowContours(pixels,tam)
cv.drawContours(image,contours,-1,(0,255,73),3)
print('Número de',nombre,':',contours.shape[0])

#saving images
cv.imwrite('output\\gray.jpg',gray)
cv.imwrite('output\\result.jpg',result)

```

```
cv.imwrite('output\\contours.jpg',image)
```

```
#resize images  
result = cv.resize(result,(640,480))  
image = cv.resize(image,(640,480))  
gray = cv.resize(gray,(640,480))
```

```
#show images  
cv.imshow('micro',result)  
cv.imshow('Contours',image)  
cv.imshow('gray',gray)
```

Código kernel

En este código podemos encontrar el algoritmo principal del programa, que identifica partículas a partir de su tipo de borde.

```
__kernel void btm(__global const uchar *image, __global unsigned char
*result, int rows, int cols,int maxBtm,int margin,int minBtm){

    int i = get_global_id(0);
    int j = get_global_id(1);
    int btm;
    int auxk,auxl;
    int br = 0;

    if((i > 0 && i < rows-1) && (j > 0 && j < cols-1))
    {

        for(int k= i-1; k<= i+1; k++)
        {
            for(int l=j-1;l<= j+1; l++)
            {

                btm=0;
                auxk = k;
                auxl = l;

                while(btm <= maxBtm)
                {
                    if(image[i*cols+j]-image[k * cols+ l] <= -margin ||
image[i*cols+j]-image[k * cols + l] >= margin)
                    {
                        btm++;
                        k = k+(k-i);
                    }
                }
            }
        }
    }
}
```

```

    l = l+(l-j);
    if((k <=0 || k>=rows-1 || l<=0 || l>= cols-1)){
        break;
    }

}
else
{
    break;
}
}

k = auxk;
l = auxl;

if(btm > minBtm && btm <= maxBtm)
{
    result[i*cols+j] = 255;
    br = 1;
    break;
}

if( br == 1)
{
    break;
}

}
}
}
}

```

Anexo II

Implementación de Canny

Código de implementación de Canny

En el siguiente código se encuentran todas las funciones pertenecientes a la implementación de Canny, código que fue utilizado en versiones tempranas del programa pero se sustituyó por el algoritmo explicado durante el proyecto.

```
import pyopencl as cl
import cv2 as cv

def SetContoursVector(mat,image):
    ret,thresh = cv.threshold(mat,127,255,0)
    contours,hierarchy =
    cv.findContours(thresh,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
    cv.drawContours(image,contours,-1,(0,255,73),3)
    return contours

def ShowContours(contours):
    for i in range(len(contours)):
        print('CONTORNO',i,':',len(contours[i]),'puntos')

    print('Número de partículas: '+str(len(contours)))

#-----main program-----# 

#read image
image = cv.imread('imagenes_definitivas\\foto7-500-1.jpg');
size = (4056,3040)
image = cv.resize(image,size)
```

```
#-----Canny detection-----#
```

```
#apply some filters
```

```
Size = (3,3)
```

```
cv.blur(image,Size)
```

```
gray = cv.cvtColor(image,cv.COLOR_BGR2GRAY)
```

```
#apply canny filter
```

```
canny = cv.Canny(gray,0,50)
```

```
canny = cv.dilate(canny,None,iterations=1)
```

```
canny = cv.erode(canny,None, iterations=1)
```

```
# set contours vector
```

```
contours = SetContoursVector(canny,image) #this vector has the coordinates of  
contours in the picture
```

```
ShowContours(contours)
```

```
#-----#
```

```
cv.imwrite('source.jpg',image)
```

```
cv.imwrite('canny.jpg',canny)
```

```
cv.waitKey(0)
```

Anexo III

Aplicación de escritorio:
Manual de Usuario

Durante este capítulo se procede a detallar el proceso de instalación y uso de la aplicación desarrollada, así como algunos detalles para la ejecución del programa. Recordemos que esta aplicación es simplemente un interfaz para acceder a ejecutar el algoritmo desarrollado y que incorpora facilidades para cargar la imagen a procesar, seleccionar el tipo de partícula a detectar (en caso de que se hayan dado de alta varias) y mostrar los resultados de la ejecución.

Instalación

Para la instalación del programa deben realizarse dos pasos:

- Para la descarga del programa se ha dispuesto mediante el uso de la plataforma Mega [31] un archivo comprimido llamado BTM_program que contiene los archivos necesarios para el correcto funcionamiento del programa.
- Una vez descargados los archivos, debe posicionarse el archivo comprimido en el lugar donde se desee instalar el programa, una vez realizado este paso se deberá seleccionar la opción “extraer aquí” para comenzar con la extracción del programa, por último, una vez se han extraído los archivos el programa estará listo para su ejecución.

Dentro del archivo extraído se encontrarán una serie de carpetas y ficheros, el archivo ejecutable del programa se llama “btm_program”, también se dispone de una carpeta llamada “images” en la que se encuentran las imágenes utilizadas como muestra para pasar al programa, y por último, una vez se ha ejecutado el programa, se pueden visualizar los resultados en la carpeta output.

Uso

Al ejecutar el programa se presentará un menú con tres opciones, “Identificar un tipo de partícula”, “Añadir un tipo de material” y “Modificar un tipo de material”, el funcionamiento de las mismas se expone más adelante. La siguiente figura muestra el interfaz de la aplicación.

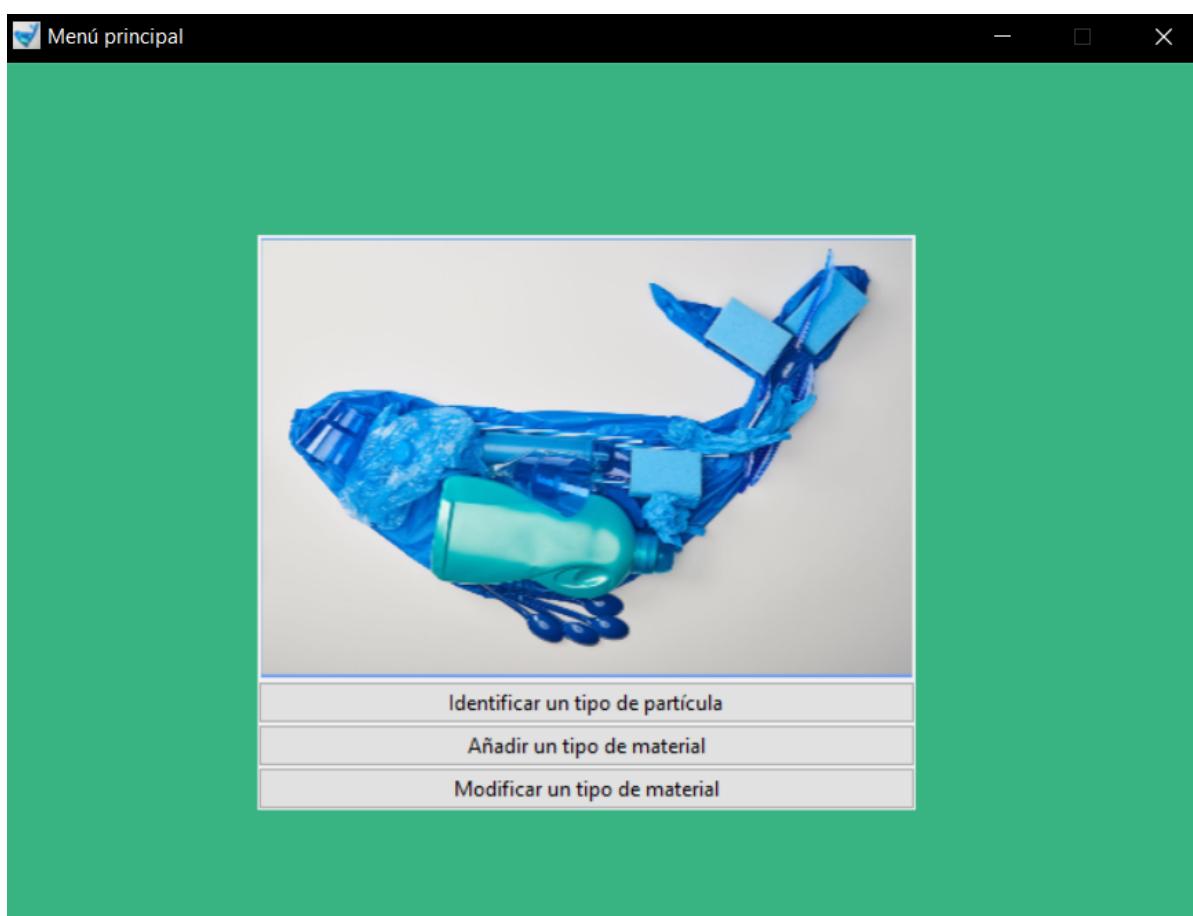


Figura I.1: Interfaz del Menú principal.

Donde:

- Identificar un tipo de material: es la función principal del programa y con ella se ejecuta el algoritmo expuesto a lo largo del proyecto, si se opta por esta opción se presentará un menú con tres opciones, una permite seleccionar el tipo de material que se quiere identificar en la fotografía, la segunda permite seleccionar la fotografía que se quiere identificar, y la tercera es un botón que activará el algoritmo e iniciará la búsqueda del tipo de material seleccionado dentro de la imagen elegida.
- Añadir un tipo de material: esta opción permite registrar un tipo de material concreto para su identificación, para ello se deben introducir cuatro parámetros, el nombre del tipo de material a identificar, su btm o borde de tipo de material mínimo y máximo, y el margen mínimo necesario entre píxel y píxel para ser considerado borde.
- Modificar un tipo de material: esta opción permite modificar una partícula que ya se encuentre presente en el sistema, siendo posible modificar cada uno de los cuatro campos mencionados en el apartado anterior.

Todo el código presente en este manual se puede consultar en la bibliografía ya que se encuentra subido a la plataforma GitHub[30].

Anexo IV

Aplicación de escritorio:
Código de la interfaz

Código de interfaz

En el siguiente código se encuentran todas las funciones pertenecientes a la interfaz gráfica del programa.

```
from tkinter import filedialog
from tkinter import *
from tkinter.ttk import*
from functools import partial
import runpy
import sys
import os

def btm(nombre,c):
    sys.argv = [' ',str(nombre.current()),c.get()]
    runpy.run_path('./build/scripts-3.9/main.py',run_name = '__main__')

def particle():
    root = Tk()
    root.title("Particle identify")
    root.config(width="700", height="500",background = "#38b382")
    root.iconbitmap('images\\icon.ico')
    root.resizable(0,0)

    frame1 = Frame(root)
    frame1.place(x = 250, y = 200)

    texto =Label(frame1, text = "Seleccione la partícula a identificar")
    texto.grid(row=0,column = 0)

    f = open('files\\particles.txt', 'r')
```

```

options = []

for i in f:
    line = i.split(sep = ',')
    options.append(line[0])

w = Combobox(frame1, values = options)
w.grid(row= 1, column = 0)

texto2 =Label(frame1, text = "Seleccione una imagen a procesar")
texto2.grid(row = 3,column = 0)

def OpenFile():
    archivo_abierto = StringVar()
    archivo_abierto.set(filedialog.askopenfilename(initialdir = "/", title =
"Seleccione archivo", filetypes = (("jpeg files", " .jpg"),("all files","*.*"))))
    c.set(archivo_abierto.get())

c = StringVar()
Button(frame1,text = "Abrir archivo",command = OpenFile).grid(row =
4,column = 0)

btm_arg = partial(btm,w,c)
button = Button(frame1,text = "send",command = btm_arg)
button.grid(row = 5, column = 0)

root.mainloop()

def fileWrite(EntryNombre,EntryMinBtm,EntryMaxBtm,EntryMargin,root):

    if(EntryNombre.get() == "" or EntryMinBtm.get()=="" or
EntryMaxBtm.get()=="" or EntryMargin.get() == ""):
        print("Error, debe especificar todos los campos.")
        root.quit()

```

```

file = open('files\\particles.txt','a')
line =
EntryNombre.get()+'+'+EntryMinBtm.get()+'+'+EntryMaxBtm.get()+'+'+EntryMa
rgin.get()+'\n'
file.write(line)

file.close()
print("partícula añadida.")
root.destroy()

def addParticle():

    root = Tk()
    root.title("Add Particle")
    root.config(width="700", height="500",background = "#38b382")
    root.iconbitmap('images\\icon.ico')
    root.resizable(0,0)

    frame1 = Frame(root)
    frame1.place(x = 200, y = 175)

    nombre = Label(frame1,text = "Nombre de la partícula: ")
    nombre.grid(row = 0,column = 0)

    EntryNombre = Entry(frame1)
    EntryNombre.grid(row= 0,column = 1,pady = 5,padx = 5)

    MinBtm = Label(frame1,text = "Btm mínimo de la partícula: ")
    MinBtm.grid(row = 1,column = 0)

    EntryMinBtm = Entry(frame1)
    EntryMinBtm.grid(row= 1,column = 1,pady = 5,padx = 5)

    MaxBtm = Label(frame1,text = "Btm máximo de la partícula: ")
    MaxBtm.grid(row = 2,column = 0)

```

```

EntryMaxBtm = Entry(frame1)
EntryMaxBtm.grid(row= 2,column = 1, pady = 5,padx = 5)

margin = Label(frame1,text = "Margen de borde: ")
margin.grid(row = 3,column = 0)

EntryMargin = Entry(frame1)
EntryMargin.grid(row= 3,column = 1, pady = 5,padx = 5)

v = StringVar()
fileWrite_arg =
partial(fileWrite,EntryNombre,EntryMinBtm,EntryMaxBtm,EntryMargin,root)
button = Button(frame1,text = "send", width = 15,command = fileWrite_arg)
button.grid(row = 4, column = 0)

def
modifyFile(nombre,EntryNombre,EntryMinBtm,EntryMaxBtm,EntryMargin,ro
ot):
    contenido= list()
    with open('files\\particles.txt', 'r') as archivo:
        for linea in archivo:
            columnas = linea.split(',')
            print(nombre.get())
            print(columnas[0])
            if(nombre.get() == columnas[0]):

    contenido.append(EntryNombre.get()+'+'+EntryMinBtm.get()+'+'+EntryMaxBtm
    .get()+'+'+EntryMargin.get()+'\n')
    else:
        contenido.append(linea)

    with open('files\\particles.txt', 'w') as archivo:
        archivo.writelines(contenido)

```

```

print("partícula modificada")
root.destroy()

def modifyParticle2(nombre):

    root = Tk()
    root.title("Modify Particle")
    root.config(width="700", height="500",background = "#38b382")
    root.iconbitmap('images\\icon.ico')
    root.resizable(0,0)

    frame1 = Frame(root)
    frame1.place(x = 200, y = 175)

    labelnombre = Label(frame1,text = "Nombre de la partícula: ")
    labelnombre.grid(row = 0,column = 0)

    EntryNombre = Entry(frame1)
    EntryNombre.grid(row= 0,column = 1,pady = 5,padx = 5)

    MinBtm = Label(frame1,text = "Btm mínimo de la partícula: ")
    MinBtm.grid(row = 1,column = 0)

    EntryMinBtm = Entry(frame1)
    EntryMinBtm.grid(row= 1,column = 1,pady = 5,padx = 5)

    MaxBtm = Label(frame1,text = "Btm máximo de la partícula: ")
    MaxBtm.grid(row = 2,column = 0)

    EntryMaxBtm = Entry(frame1)
    EntryMaxBtm.grid(row= 2,column = 1, pady = 5,padx = 5)

    margin = Label(frame1,text = "Margen de borde: ")
    margin.grid(row = 3,column = 0)

```

```

EntryMargin = Entry(frame1)
EntryMargin.grid(row= 3,column = 1, pady = 5,padx = 5)

modifyFile_arg =
partial(modifyFile,nombre,EntryNombre,EntryMinBtm,EntryMaxBtm,EntryMa
rgin,root)
    button = Button(frame1,text = "send", width = 15,command =
modifyFile_arg)
    button.grid(row = 4, column = 0)

def modifyParticle1():

root = Tk()
root.title("Modify Particle")
root.config(width="700", height="500",background = "#38b382")
root.iconbitmap('images\\icon.ico')
root.resizable(0,0)

frame1 = Frame(root)
frame1.place(x = 200, y = 175)

nombre = Label(frame1,text = "Nombre de la partícula a modificar: ")
nombre.grid(row = 0,column = 0)

EntryNombre = Entry(frame1)
EntryNombre.grid(row= 0,column = 1,pady = 5,padx = 5)

modifyParticle_arg = partial(modifyParticle2,EntryNombre)
button = Button(frame1,text = "send", width = 15,command =
modifyParticle_arg)
button.grid(row = 1, column = 0)

```

```
#-----Setting up interface-----#
```

```
root = Tk()
root.title("Menú principal")
root.config(width="700", height="500", background = "#38b382")
root.iconbitmap('images\\icon.ico')
root.resizable(0,0)
```

```
frame1 = Frame(root)
frame1.place(x = 150, y = 100)
```

```
foto = PhotoImage(file="images\\icono.png")
Label(frame1,image=foto).grid(row = 0, column = 0)
```

```
ParticleButton = Button(frame1, text = 'Identificar un tipo de partícula', width =
62, command = particle)
ParticleButton.grid(row=1,column=0)
```

```
addButton = Button(frame1, text ='Añadir un tipo de material',width =
62, command = addParticle)
addButton.grid(row = 2, column = 0)
```

```
addButton = Button(frame1, text ='Modificar un tipo de material',width =
62, command = modifyParticle1)
addButton.grid(row = 3, column = 0)
```

```
root.mainloop()
```

