UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
Ingeniera e Ingeniero de Sistemas

TEMA:
DESARROLLO DE UN ALGORITMO QUE PERMITA PARALELIZAR IMÁGENES 3D SOBRE TECNOLOGÍA NVIDIA Y OPENCL

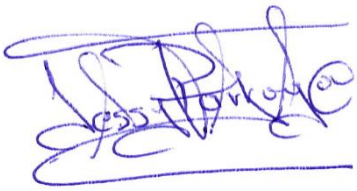
AUTORA Y AUTOR:
JÉSSICA VIRGINIA PÁRRAGA PÁRRAGA
LUIS FERNANDO CASA LÓPEZ

TUTOR:
WASHINGTON ARSENIO RAMÍREZ MONTALVÁN

Quito, marzo del 2017

CESIÓN DE DERECHOS DE AUTOR

Nosotros: Jéssica Virginia Párraga Párraga, con documento de identificación Nº 1311902561, y Luis Fernando Casa López, con documento de identificación N° 1712034501, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación con el tema: "DESARROLLO DE UN ALGORITMO QUE PERMITA PARALELIZAR IMÁGENES 3D SOBRE TECNOLOGÍA NVIDIA Y OPENCL.", mismo que ha sido desarrollado para optar por el título de INGENIERA E INGENIERO DE SISTEMAS, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que hacemos entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Politécnica Salesiana.



JÉSSICA VIRGINIA
PÁRRAGA PÁRRAGA
C.I.: 1311902561

LUIS FERNANDO
CASA LÓPEZ
C.I.: 1712034501

Quito, marzo del 2017

DECLARATORIA DE COAUTORÍA DEL TUTOR

Yo declaro que bajo mi dirección y asesoría fue desarrollado el trabajo de titulación, con el tema: DESARROLLO DE UN ALGORITMO QUE PERMITA PARALELIZAR IMÁGENES 3D SOBRE TECNOLOGÍA NVIDIA Y OPENCL. Realizado por Jéssica Virginia Párraga Párraga y Luis Fernando Casa López, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, marzo del 2017

……

WASHINGTON ARSENIO RAMÍREZ MONTALVÁN

C.I.: 1710804681

Dedicatoria

En primer lugar, dedico el presente trabajo a Dios por darme la fortaleza suficiente para culminar esta pequeña e importante etapa de mi vida, un profundo agradecimiento a mi madre y hermanos que siempre han sido esa luz al final del túnel a lo largo de mi carrera, a cada uno de mis amigos y familiares que tuvieron las palabras justas en el momento preciso para brindarme un consejo.

Jéssica Virginia Párraga Párraga

El conjunto fundamental para llegar al punto de realizar el presente documento es Dios y mis padres, a mis padres les agradezco infinitamente por confiar en mis capacidades, y extenderme la mano para salir adelante, que a pesar de haberme saltado uno o varios pasos en el proceso que me cambió, de hijo a padre de familia siempre han estado junto a mí. Sin dejar de lado a mi esposa e hijo, que forman parte de éste agradecimiento ya que son la voz de aliento, y el motor que me da fuerza para seguir y no rendirme ante nada. Este pequeño triunfo está dedicado para Guadalupe, Luis, Angélica y Alan.

Luis Fernando Casa López

Agradecimiento

Queremos extender un sincero agradecimiento a la Universidad Politécnica Salesiana que ha contribuido a nuestra formación profesional durante estos años de estudio y un agradecimiento especial a cada una de las personas que brindaron su aporte en el desarrollo de este proyecto.

Luis Fernando Casa López

Jéssica Virginia Párraga Párraga

ÍNDICE DE CONTENIDO

INTRODUCCION 1

Alcance 3

Justificación 4

Objetivos 4

Objetivo General 4

Objetivos Específicos 5

1. REVISIÓN BIBLIOGRÁFICA 6

1.1. Digitalización de imágenes 6

1.2. Imagen digital 6

1.2.1. Tipos de imágenes digitales 7

1.3. Procesamiento de Imágenes 12

1.3.1. Preprocesamiento 13

1.3.2. Segmentación 13

1.4. Computación paralela 19

1.4.1. Procesamiento en paralelo 20

1.4.2. Modelo de arquitectura paralelas 21

1.5. Grupos de investigación relacionados con el área de procesamiento de imágenes 28

1.6. Trabajos relacionados 29

2. METODOLOGÍA 34

2.1. Paso 1: Prerrequisitos 34

2.2. Paso 2: Clúster 35

2.3. Paso 3: DataSet 37

2.4. Paso 4: Algoritmo 38
2.5. Paso 5: Definición de variables 45
3. RESULTADOS 47
3.1. Pruebas imágenes 47
CONCLUSIONES 59
RECOMENDACIONES 60
GLOSARIO DE TÉRMINOS 61
LISTA DE REFERENCIAS 64

ÍNDICE DE TABLAS

Tabla 1. Análisis de algoritmos de procesamiento de imágenes 18

Tabla 2. Clasificación de la taxonomía de Flynn 22

Tabla 3. Estudio de variables 45

Tabla 4. Tiempos de ejecución algoritmo Retinex secuencial y paralelo 49

Tabla 5. Datos proceso Retinex secuencial y paralelo en tomografía computacional 52

Tabla 6. Datos proceso Retinex secuencia y paralelo en Toutatix 56

ÍNDICE DE FIGURAS

Figura 1. Tipo de imágenes digitales 7

Figura 2. Imagen binaria 8

Figura 3. Imagen en escala de gris 9

Figura 4. Imagen indexada 9

Figura 5. Imagen en RGB 10

Figura 6. Representación jerárquica de un archivo DICOM 11

Figura 7 Imagen DICOM 12

Figura 8 Máscara 3 x 3 14

Figura 9. Segmentación usando regiones 16

Figura 10 A. Ejecución secuencial de un algoritmo, B. Ejecución simultánea de instrucciones 20

Figura 11 A. Arquitectura CPU, B. Arquitectura GPU 24

Figura 12 Arquitectura de OpenCL 25

Figura 13. Modelo de plataforma en OpenCL 26

Figura 14 Explicación del NDRange 27

Figura 15. Arquitectura ESXI 36

Figura 16. A. Toutatix, B. Volumerge 37

Figura 17. Convolución gaussiano 40

Figura 18 A. Original con neblina. B. Resultado Retinex neblina 41

Figura 19. Imagen subacuática tortugas con proceso Retinex e histograma (image8-1) 48

Figura 20. Imagen subacuática pez con proceso Retinex e histograma (image16-1) 48

Figura 21. Imagen subacuática tortuga con proceso Retinex e histograma (image2-1) 48

Figura 22. Entorno 3D del DataSet Toutatix 50

Figura 23. Espina dorsal con proceso Retinex e histograma (v_img1)..... 51

Figura 24. Espina dorsal con proceso Retinex e histograma (v_img11)..... 52

Figura 25. Tiempo de ejecución tomografía computacional grupo1 54

Figura 26. Arteria coronaria con proceso Retinex e histograma (t_img1)..... 55

Figura 27. Arteria coronaria con proceso Retinex e histograma (t_img11)..... 55

Figura 28. Tiempo de ejecución Volumerge..... 58

Resumen

El presente trabajo, se desarrolla en torno a la problemática sobre los procesos que se ejecutan de manera serial en la CPU y al excesivo tiempo que toma en mostrar resultados al usuario final; principalmente en el procesamiento de imágenes. La propuesta de solución para mejorar el tiempo de ejecución es la paralelización de procesos y que éstos se ejecuten en la GPU, para lo cual, se utilizó tecnología NVIDIA y programación con OpenCL. Con respeto al procesamiento de imágenes se aplicó el algoritmo de Retinex, del que podemos destacar que se basa en la visión humana para mejorar los colores en escenarios de escasa luminosidad. Para las pruebas realizadas en la experimentación se utilizó DataSet de tomografías computarizadas, las cuales permiten mostrar un entorno 3D con la información de los frames que contiene.

Abstract

The present project is developed around the problem about the processes that are executed serially in the CPU and the excessive time it takes to show results of the user mainly in the image processing. The offer of the solution to improve the execution time is the parallelization of processes and spoils them in the GPU, for which NVIDIA technology is used and OpenCL programming. Regarding to image processing, the Retinex algorithm is applied, from which it can be emphasized that it is based on the human vision to improve the colors and scenarios of low luminosity. DataSet of computerized tomographies was used for the tests performed, which allow to display a 3D environment with the information of the frames that it contains.

INTRODUCCION

En 1844, Charles Wheatstone crea “el estereoscopio”, el cual consiste en obtener dos fotografías casi idénticas pero que se diferencian ligeramente en el punto de toma de la imagen; estas serán observadas por cada ojo de manera separada y el cerebro las mezclará en una sola imagen creando un efecto tridimensional, el cual corresponde a una serie de fórmulas matemáticas que describen una interpretación virtual de una realidad volumétrica. Desde una perspectiva gráfica, el diseño tridimensional refiere a una “representación esquemática visible, a través de un conjunto de objetos, elementos y propiedades que una vez procesados, o renderizados, se convertirán en una imagen o animación tridimensional” (Molina, 2011; Rosa, 2002).

Al pasar de los años, en los computadores secuenciales se ha observado un desarrollo importante, sin embargo a nivel de los supercomputadores el crecimiento ha sido mínimo, ya que en 15 años se ha mejorado únicamente en 4 nanosegundos con relación a los 12 nanosegundos de períodos de reloj de la máquina CRAY1 creada en 1970, en donde se refleja los inicios de la evolución en la capacidad de procesamiento, con la inclusión del concepto de computación paralela en la arquitectura del sistema, ya con este cambio se elimina la dependencia del hardware para enfocarse en mejorar el diseño del software.

Hoy en día la programación en paralelo, son varios procesadores trabajando en conjunto para dar solución a una tarea en común, lo que permite dividir el trabajo y cada procesador realiza una porción del problema al poder intercambiar datos por una red de interconexión o a través de memoria compartida, es por ello que ésta programación fue creada porque se comprobó que trabajar en paralelo es eficiente, debido a que el seccionamiento del hilo principal se fracciona en hilos hijos (más

pequeños), además que permite resolver problemas que no caben en un solo procesador y que no se resuelven en un lapso de tiempo razonable, se pueden ejecutar problemas con mayor complejidad más rápidamente (Museum, 1996; Benoit, 1999; Aguilar, 2005; Gordillo & Itehua, 2012).

En la década de los 60 nace la GPU (Unidad de Procesamiento Gráfico), en las impresoras con el único propósito de plasmar en papel el resultado del monitor, pero a finales de los 80 los sistemas operativos toman un giro, al implementar la interfaz gráfica y con esto empieza la comercialización de aceleradores gráficos 2D para computadores. En 1993 se forma uno de los principales fabricantes de tarjetas gráficas NVIDIA. En 1995 y 1997 se crean las primeras tarjetas hibridas 2D y 3D respectivamente fabricadas por empresas como Matrox[1], Creative [2], ATI [3] y 3dfx[4]. En 1999 NVIDIA fabrica la tarjeta gráfica Geforce 256, lo que daría la pauta a las siguientes generaciones de tarjetas gráficas, que implementarían la primera versión del DirectX8, que hace posible que los programadores desarrollen aplicaciones en la GPU. En Octubre del 2010, se lleva a cabo una revolución en el ámbito tecnológico, la National Supercomputing Center de China construye la Tianhe-1A, un supercomputador que es capaz de realizar 2566 billones de operaciones de punto flotante por segundo, la importancia de ésta creación radica en la utilización de la GPU, donde se aplica el término GPGPU (GPU de Propósito General), concepto que se refiere, a que la GPU no solo se centra en el procesamiento de gráficos sino que

[1] Matrox fundada en 1976 se encuentra en Dorval, Quebec, Canadá.
[2] Creative fundada en 1981 con sede en Singapur
[3] ATI fundada en 1985 y fue absorbida por la empresa AMD en 2006
[4] 3dfx fundada en 1994 con sede en California, USA

también ayuda a las operaciones y procesamientos que realiza la CPU (Unidad Central de Procesos) (Piccoli M. F., 2011; Luis, 2012; Huesca, 2012; NVIDIA, 2015).

OpenCL (Open Computing Lenguage) es un lenguaje abierto multiplataforma que es desarrollado bajo el Grupo Khronos, grupo enfocado en desarrollos para gráficos y medios gráficos. El desarrollo del lenguaje OpenCL posee tres ventajas: i) portabilidad ii) procesamiento vectorial estandarizado iii) programación paralela (Reimúndez, 2009-2010; Vance, 2010; Scarpino, 2012).

Para este estudio se aplica todos los conceptos antes mencionados, enfocados en optimizar un algoritmo que permita paralelizar imágenes 3D sobre tecnología NVIDIA y OpenCL.

Alcance

En el presente trabajo se ha realizado una investigación de algoritmos de procesamiento paralelizado de imágenes 3D, usando fuentes bibliográficas confiables donde se mencione la integración de la paralelización de imágenes 3D aplicadas a la GPU, y posteriormente evaluar el rendimiento de al menos 3 algoritmos investigados para determinar el más eficiente, una vez determinado el algoritmo se evaluará mediante el uso de OpenCL, el cual permitirá aprovechar la potencia de la computación masiva paralela de la GPU NVIDIA que ayuda a crear aplicaciones de procesamiento gráfico en 3D, mejorando así el rendimiento de los procesos.

Usando la tecnología de la GPU de NVIDIA se construirá un clúster para HPC (High Performance Computing) y posteriormente se implementará el algoritmo de paralelización de imágenes 3D desarrollado en OpenCL para el procesamiento de imágenes 3D en el clúster construido, al finalizar el desarrollo del proyecto, y una vez

implementado el algoritmo en el clúster, se realizará pruebas de rendimiento, la cuales permitirán evaluar la mejora en el tiempo de procesamiento de las imágenes 3D.

Justificación

En la actualidad, con la mayor demanda de procesos que debe ejecutar un computador, se ha hecho casi imposible que solo sean ejecutados en la CPU, por lo cual es necesaria la aplicación de procesos multiplataforma; es decir, que se ejecuten tanto en la CPU como en la GPU.

Los procesos relacionados con la manipulación de imágenes 3D, están dentro de los que mayor demanda de recursos requiere, por lo cual para la gestión rápida de ésta gran cantidad de procesos, las tarjetas gráficas han evolucionado y cada vez poseen más procesadores, y es aquí donde se aplica la programación mediante hilos (paralelización).

A lo largo del desarrollo del proyecto, para la programación en paralelo se utilizó OpenCL, que es ejecutado sobre un GPU con tecnología NVIDIA.

El objeto de trabajar con estas herramientas tecnológicas es aplicarlas a un sistema que maneja imágenes 3D acoplado en un clúster, que posee una inadecuada distribución de procesos; con estos antecedentes el aporte será optimizar un algoritmo para aplicarlo en el sistema y mejorar su funcionamiento.

Objetivos

Objetivo General

Desarrollar e implementar un algoritmo optimizado que permita paralelizar procesos de imágenes 3D del cuerpo humano con OpenCL sobre tecnologías NVIDIA y GPU.

Objetivos Específicos

1. Investigar 3 algoritmos de procesamiento de imágenes 3D y evaluar cuál es el más eficiente.
2. Analizar el algoritmo seleccionado y optimizarlo con GPU, paralelizando los métodos en OpenCL y tecnología NVIDIA para el procesamiento de las imágenes 3D.
3. Levantar un Clúster para HPC con tecnología NVIDIA sobre imágenes 3D.
4. Implementar el algoritmo optimizado y en paralelo para imágenes 3D.
5. Realizar métricas que permitan verificar la optimización de tiempos del algoritmo implementado.

Capítulo 1

Revisión bibliográfica

Los temas a tratar en este apartado son referentes a las imágenes digitales y los procesamientos que se efectúan sobre las mismas. Se plantean algunos factores que afectan el proceso de captura de las imágenes y se ejemplifican algoritmos que ayudan a mejorar y restaurar dichas imágenes, centrando la atención en los algoritmos basados en el procesamiento de imágenes. Se hace referencia a la programación paralela que nos ayuda a diseñar algoritmos de manera eficiente. A su vez, se ha definido el estado del arte de las características principales de las tecnologías actuales que permiten dar solución al problema planteado mediante el uso de computación paralela y distribuida. Finalmente se realiza una revisión de trabajos relacionados con respecto al tema "Desarrollo de un algoritmo que permita paralelizar imágenes 3D sobre tecnología NVIDIA y OpenCL".

Digitalización de imágenes

La digitalización se encarga de la transformación de una imagen analógica mediante la utilización de un sistema de adquisición apropiado como un escáner o una cámara fotográfica, para posteriormente ser utilizada por un computador. A ésta nueva imagen se la denomina imagen digital (Salih, 2010).

Imagen digital

Una imagen digital puede considerarse como un arreglo o un conjunto de arreglos bidimensionales (matriz de bits) que se forman a través de un proceso de digitalización de imágenes o señales analógicas. Las imágenes digitales están compuestas por un número finito de elementos, los cuales se conocen con el nombre de píxeles (elementos

de imagen). Cada píxel posee una localización y tienen un valor que representa la intensidad de la luz que llega a él. Por tanto, una imagen puede ser almacenada como una matriz de píxeles $m \times n$, donde m y n representan las dimensiones y cada elemento de esta matriz es un píxel (Gonzalez, 2004).

1.1.1. Tipos de imágenes digitales. Dentro de las imágenes digitales estáticas tenemos dos clasificaciones: i) Imágenes vectoriales y ii) Mapa de bits. Las imágenes vectoriales están formadas por líneas, las cuales definen un gran número de figuras geométricas y la principal característica es que no pierden su forma al aumentar su tamaño. Por otra parte en una imagen de mapa de bits al aumentar el tamaño se puede observar la división de la malla de pixeles por la que está compuesta. En la Figura 1, como se puede apreciar la diferencia entre imágenes vectoriales y mapa de bits, teniendo mayor nitidez las imágenes vectoriales al aumentar el tamaño, a diferencia de las imágenes de mapas de bits que se distorsiona la imagen (Valdivia, 1996; Arribas, 2011).

Figura 1. Tipo de imágenes digitales
Fuente: (Arribas, 2011)

Los tipos más utilizados de imágenes de mapa de bits son: i) Imágenes binarias, ii) Imágenes de intensidad, iii) Imágenes indexadas e iv) Imágenes en RGB.

- **Imágenes binarias:** Son únicamente imágenes en blanco y negro. Representan una matriz de datos o píxeles que toman valores lógicos tales como verdadero-falso ó 0-1, donde el 0 representa el negro y el 1 el blanco. Como se puede apreciar en la Figura 2, se toma una parte específica de la imagen en blanco y negro y se le aumenta de tamaño, notando así la representación binaria de 1 (blanco) y 0 (negro) (Gonzalez, 2004).

Representación de imagen Binaria

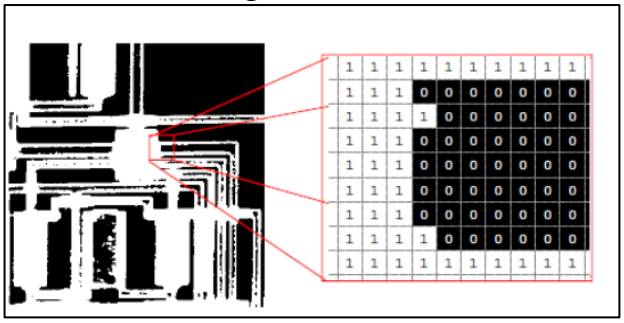


Figura 2. Imagen binaria
Fuente: (Gonzalez, 2004)

- **Imágenes de intensidad:** Son imágenes en escala de gris o escala monocromática. Se representa como una sola matriz de $m \times n$ píxeles, cuyos valores para una imagen de 8 bits van desde 0 hasta 255 donde: el 0 representa el negro, el 255 el blanco y los valores intermedios son los distintos tonos de gris. Las fotografías en blanco y negro son un ejemplo de este tipo de imagen (Gonzalez, 2004). Como se puede observar en la Figura 3 tiene colores negro, blanco y varios tonos de gris, los cuales van desde el 0 al 255.

Representación de imágenes de intensidad

Figura 3. Imagen en escala de gris
Fuente: (Gonzalez, 2004)

- **Imágenes indexadas:** Este tipo de imágenes consiste en un arreglo o matriz, que se denota como x y y, una matriz que se llama mapa de color o paleta de colores. El arreglo x tiene el mismo tamaño de la imagen original y guarda un valor por cada píxel. En la Figura 4 se muestra una imagen indexada, donde se puede observar la asignación de colores mediante los arreglos en cada columna de la matriz, es posible notar que cada número de la matriz está asociado con 3 números decimales de la paleta de colores y así tornar el tono requerido en la imagen (Gonzalez, 2004).

Representación de imágenes indexadas

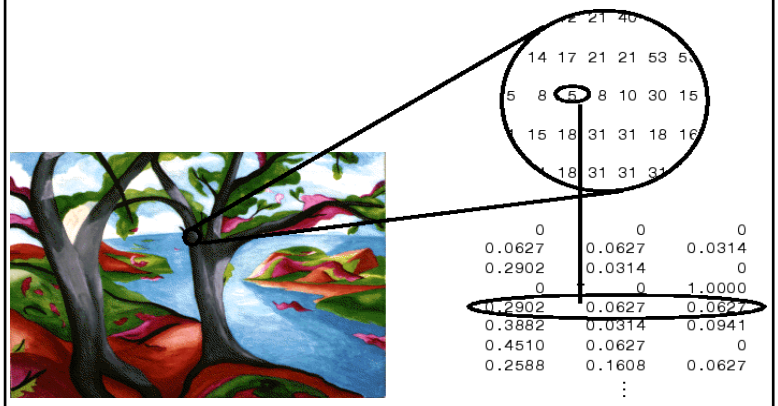


Figura 4. Imagen indexada
Fuente: (Gonzalez, 2004)

- **Imágenes en RGB:** Son imágenes a color que corresponden al modelo o espacio de color RGB. Son similares a las imágenes en escala de gris, con la diferencia de que tiene tres canales o componentes, que corresponden a los colores: rojo (R-red), verde (G-green) y azul (B-blue). Este tipo de imagen no usa un mapa de color, por lo que el valor final de cada píxel está determinado por la combinación de las intensidades de rojo, verde y azul guardadas en su correspondiente componente. En el caso de que R=G=B la imagen está representada en escala de gris. La Figura 5 nos muestra los tres colores representativos en las imágenes RGB, los cuales al unir forman el color de la imagen, así mismo se puede ver que los valores de los tres colores combinados van del 0 al 1 según la intensidad que se necesite para formar el color (Gonzalez, 2004).

Representación de imágenes RGB

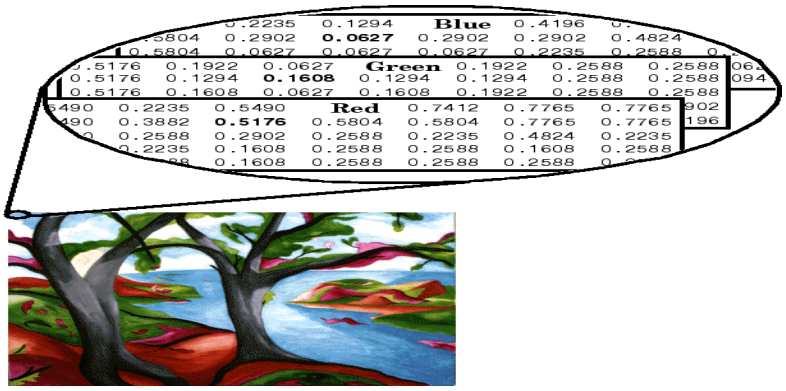


Figura 5. Imagen en RGB
Fuente: (Gonzalez, 2004)

- **Imágenes DICOM:** DICOM (Digital Imaging and Communications in Medicine) es el estándar reconocido mundialmente para el manejo, almacenamiento, impresión y transmisión de imágenes médicas. La extensión de un archivo DICOM es *.dcm (Grupo PAS, 2004).

Formato de un archivo DICOM

El formato genérico del archivo DICOM consiste en dos partes: Header y un DataSet de DICOM. El DataSet contiene la imagen o las imágenes especificadas. El Header contiene sintaxis de transferencia UID (identificador único) que especifica la codificación y la compresión del Data Set (Grupo PAS, 2004).

Header: El Header consta de 128 bytes de archivos de preámbulo y 4 bytes de prefijos "DICM". El preámbulo puede estar en blanco o contener información sobre la aplicación principal con la que debe ser ejecutado.

DataSet: este componente es reconocido mundialmente como un conjunto de imágenes que representan un archivo DICOM, el Dataset se construye de DataElement que es una etiqueta que contiene información relevante del mismo (Grupo PAS, 2004).

La Figura 6, representa un ejemplo de una imagen DICOM que describe los medios de formato e intercambio de imágenes médicas y la información relacionada para facilitar la conectividad de dispositivos y sistemas médicos DICOM.

Representación de elementos de imágenes DICOM

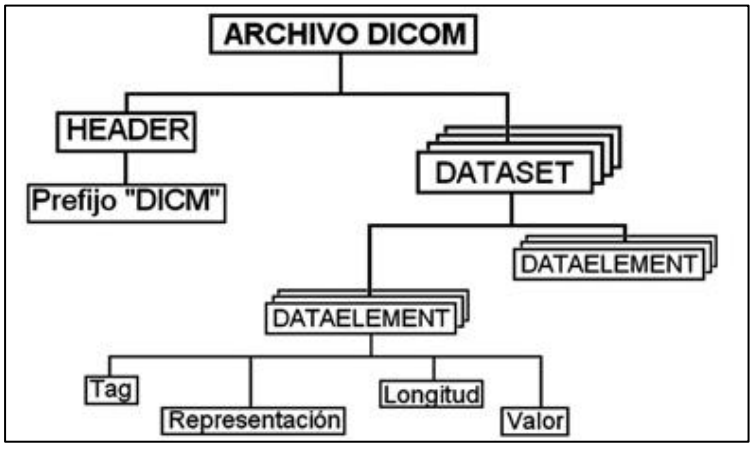


Figura 6. Representación jerárquica de un archivo DICOM
Fuente: (Grupo PAS, 2004)

La Figura 7, donde se puede apreciar claramente los dos componentes de la imagen DICOM como son el Header que viene a representar la información de la imagen y el DataSet, el cual contiene la representación de la imagen.

Representación de imagen DICOM real

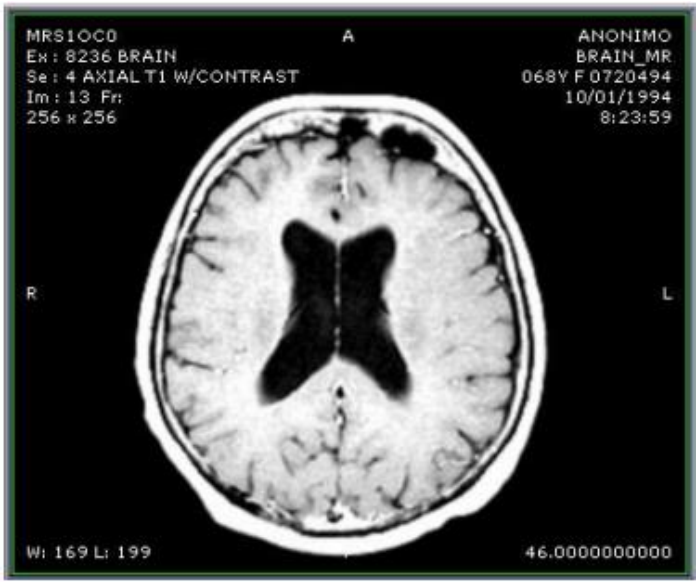


Figura 7 Imagen DICOM
Fuente: (Grupo PAS, 2004)

Procesamiento de imágenes

En el procesamiento de imágenes se pueden distinguir una secuencia de pasos (áreas o tareas) que permiten extraer las características y datos de interés de la escena observada. Entre las áreas principales se encuentran (Jähne, 2005).

1.1.2. Preprocesamiento. Conjunto de técnicas y operaciones diferentes que se realizan a fin de mejorar la apariencia visual, recuperar o restaurar las imágenes que se encuentran degradadas. Como ejemplo de estas operaciones podemos citar: la reducción del ruido, la mejora del contraste, brillo y saturación, el perfilado de la imagen, entre otras.

1.1.3. Segmentación. Se ocupa de la subdivisión de las imágenes en regiones o áreas significativas. Estas divisiones facilitan la posterior extracción de características o el proceso de clasificación.

Técnica de detección de discontinuidades

La respuesta a la máscara de cualquier píxel de la imagen viene dada por la ecuación [1.1]:

$$\mathrm{R} = w_1 z_1 + w_2 z_2 + \cdots + w_3 z_3 = \sum_{i=1}^{9} w_i z_i \qquad [1.1]$$

Donde z_i es el nivel de gris asociado al pixel de la imagen con coeficiente de la máscara w_i. La respuesta de la máscara viene referida a su posición central. Cuando la máscara esté centrada en un píxel de borde de la imagen, la respuesta se determina empleando el vecindario parcial apropiado (Martín, 2002). En la Figura 8, se puede ver un caso general de mascara de 3 x 3.

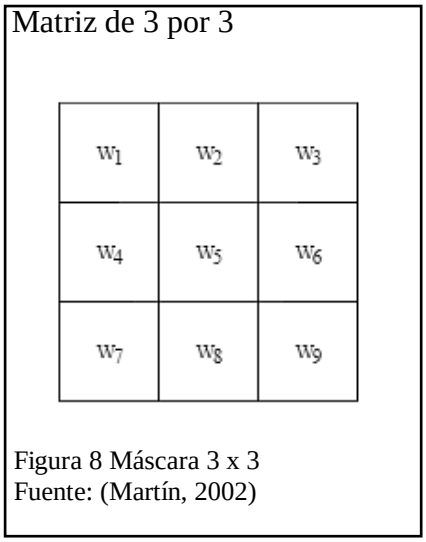


Figura 8 Máscara 3 x 3
Fuente: (Martín, 2002)

Existen varios tipos de discontinuidades, descritas a continuación:

Detección de puntos

Se basa en medir la diferencia entre el pixel central y sus vecinos, puesto que un pixel será un punto aislado siempre que sea suficientemente distinto de sus vecinos. Solamente se considerarán puntos aislados a aquellos cuya diferencia con respecto a sus vecinos sea significativa. Es decir, un punto se detecta en la posición en la que está centrada la máscara si:

$$|R| \geq T \qquad [1.2]$$

Donde:

T es un umbral no negativo

R es la suma de productos de los coeficientes con los niveles de gris contenidos en la región en la que se superpone la máscara, como lo indica Martínez (2002).

Detección de líneas

Se basa en la unión de los puntos que cumplen con un patrón de orientación, cortando una línea por cada punto. Si en un punto de la imagen se cumple, $|Ri| > |RJ|\ j \neq i$. El punto i es el que tendrá mayor probabilidad de estar asociado con una línea a continuación de la máscara (Martín, 2002).

Detección de bordes

Es el procedimiento más habitual para la detección de discontinuidades. Un borde se define como la frontera entre dos regiones con un tono relativamente diferente (Martín, 2002).

Técnica orientada a regiones:

Un tipo de segmentación es la que se enfoca en los pixeles, donde se detectan regiones según el conjunto de pixeles vecinos que poseen características similares (Palomino, 2009).

Sea R la región correspondiente a la imagen a segmentar. Se analiza el proceso de segmentación como un proceso en el cual se divide la región R en n subregiones $R_1, R_2 \dots R_n$ tal que:

$$\bigcup_{i=1}^{n} R_i = R$$

R_i Es una región conectada, $i = 1, 2, \dots, n$

$R_i \cap R_j = \emptyset$ Para todo i y j con $j \neq i$

$P(R_i) = CIERTO$ Para $i = 1, 2, \dots, n$

$P(R_i \cup R_j) = FALSO$ Para todo i y j adyacentes con $j \neq i$

La Figura 9 es un claro ejemplo, donde se muestra que una imagen contiene varias regiones y cada región es distinta (segmentación por regiones), tanto en intensidad de su color como su tamaño (Martín, 2002).

Representación de segmentación por regiones

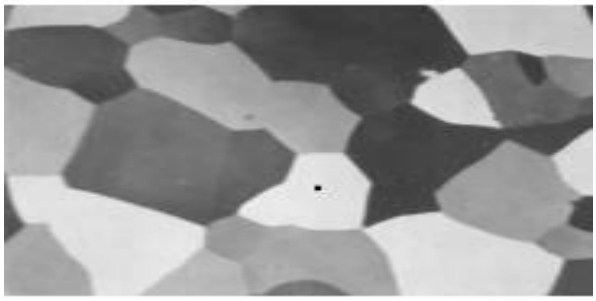


Figura 9. Segmentación usando regiones
Fuente: (Martín, 2002)

En la siguiente sección se abordará la revisión de varios algoritmos que fueron analizados y estudiados para la implementación del presente proyecto.

1.1.3.1. Algoritmos. Algoritmo Hounsfield: En 1970 Godfrey Hounsfield y Alan Cormack, presentaron la primera demostración de la tomografía axial computada, fue la primera técnica de representación tridimensional utilizada. Es una condición "todo o nada" (se ve o no se ve), y lo que se ve se clasifica como el mismo tipo de tejido, la capacidad de poder ver en mejor forma, con más precisión y menor invasión el interior del cuerpo humano.

Modelo matemático: La nueva escala tomó como referencia el agua. Por ello la nueva unidad habría que aplicar la ecuacion.

$$\mathrm{UH} = \frac{(\mathrm{uobjeto} - \mathrm{uagua})}{(\mathrm{uagua} * 1000)} \qquad [1.3]$$

Algoritmo crecimiento de regiones: El crecimiento de regiones es un procedimiento donde agrupan pixeles o subregiones en regiones mayores. La región a segmentar se la hace a partir de una semilla (*de la imagen completa*) esto se denomina agregación de pixeles de forma que a partir de una semilla crecen regiones añadiendo pixeles a dicha semilla de entre aquellos pixeles vecinos que tienen propiedades similares.

Modelo matemático: El algoritmo de segmentación por crecimiento es O (w x h), donde w es el ancho y h el alto de la imagen respectivamente (Pereira, 2008).

Algoritmo Retinex: El algoritmo Retinex mejora la representación visual de una imagen capturada en condiciones de baja iluminación. El método es conocido en la literatura científica como algoritmo MSRCR (MultiScale Retinex with Color Restoration - Retinex multiescala con restauración de color) y constituye la base del filtro Retinex que está inspirado en el mecanismo biológico del ojo humano el cual permite su adaptación relativamente rápida a condiciones de baja iluminación a partir de la contracción de retina y el córtex (Kreiner, 2008).

Modelo matemático: El modelo Retinex fue propuesto por el Dr. Edwin H. Land, que demostró como el HVS (*Sistema de Visión Humano*) es capaz de detectar la cantidad de energía que impacta a un objeto desde una fuente luminosa desconocida, lo que motivó el desarrollo de la teoría de la "Constancia del Color" (FAUNE, 2008; Mario Dehesa, 2015).

El modelo Retinex basado en el HVS recibe de cada pixel de la imagen los datos en tres canales: rojo, verde y azul, para que en el proceso del algoritmo defina la reflexión de luz en cada pixel y luego obtener el dato estimado de la fuente de iluminación como se detalla en la ecuación [1.4]:

$$f_i(x,y) = G(x,y)R_i(x,y)I_i \qquad [1.4]$$

Donde:

$I_i \rightarrow Fuente\ de\ Iluminación\ estimada$

$f_i(x,y) \rightarrow Intensidad\ de\ cada\ pixel\ de\ una\ imagen\ en\ la\ posición\ (x,y)$

$G(x,y) \rightarrow Factor\ que\ depende\ de\ las\ características\ de\ la\ escena$

$R_i(x,y) \rightarrow Reflectancia\ de\ un\ punto\ del\ objeto\ en\ la\ misma\ posición$

i →Corresponde al canal de color en la imagen (rojo, verde, azul)

1.1.3.2. Análisis rendimiento algoritmos. La tabla 1 presentada a continuación muestra una breve descripción de las características de los algoritmos seleccionados, con el fin de obtener un análisis comparativo entre ellos.

Tabla 1. Análisis de algoritmos de procesamiento de imágenes

Características	**Hounsfield**	**Crecimiento Regiones**	**Retinex**
Imágenes en escala de grises	Si	Si	Si
Imágenes RGB	No	No	Si
Agrupación de píxeles	No	Si	Si
Gestión de componentes de cromaticidad en objetos	No	No	Si
Uso del sistema de visión humana (HVS)	No	No	Si
Implementado en condiciones de baja iluminación	No	No	Si
Procesamiento en serie (CPU)	No	Si	Si
Procesamiento en paralelo (GPU)	Si	No	No
Reduce las variaciones de luz	No	No	Si
Optimiza el contraste de imágenes	No	No	Si
Uso de constancia de color	No	No	Si

Nota: Esta tabla contiene algunas de las características más destacadas de los algoritmos

Como se puede observar, en la Tabla 1 se detallan algunas de las características más importantes de los algoritmos seleccionados, las cuales sirven para determinar cuál es el más óptimo, siendo Retinex el que se analizará en el presente trabajo ya que tiene características que posee también los otros dos algoritmos, Hounsfield y Crecimiento de Regiones y el enfoque será realizar el proceso del algoritmo Retinex en paralelo el cual no se encuentra desarrollado actualmente.

Computación paralela

La computación paralela se ha convertido en una poderosa herramienta empleada para obtener soluciones que requieren computación intensiva, permitiendo reducir el tiempo de cálculo en problemas que necesitan alto costo computacional. El objetivo fundamental es ejecutar muchas instrucciones simultáneamente mediante el uso concurrente de varios procesadores los cuales resuelven problemas más rápido que con un solo procesador. Existen varios niveles de paralelismo que se diferencian unos de otros por la forma de ejecución simultánea (Rodriguez, 2011).

- Paralelismo de bit: realiza un aumento del tamaño de la palabra del procesador. Con este aumento del tamaño de la palabra se reduce el número de procesos que se deben ejecutar para realizar una operación determinada.
- Paralelismo de instrucciones: es la ejecución simultánea de varias instrucciones donde se debe cumplir la condición de que no exista dependencia entre los datos.
- Paralelismo de datos: en este nivel se divide el conjunto de datos de entrada del algoritmo o problema a resolver, de forma que a cada procesador le corresponda un subgrupo como resultado de la división. Cada procesador realiza las mismas operaciones sobre el subgrupo que le fue asignado.
- Paralelismo de tareas: asigna tareas diferentes a cada uno de los procesadores del sistema, por lo que cada procesador realiza operaciones diferentes entre sí.

La Figura 10 muestra cómo un algoritmo se encuentra dividido en pasos o instrucciones y estos se ejecutan de forma secuencial.

El desarrollo tecnológico ha permitido a los programadores encontrar una forma para utilizar lo mejor posible las capacidades disponibles, apareciendo la Computación Paralela y Distribuida (CPD), que consiste en el empleo simultáneo de varios

elementos con capacidad de multiprocesamiento con la finalidad de resolver una problemática.

Los sistemas de cómputos paralelos son aquellos sistemas capaces de ejecutar muchas instrucciones simultáneamente para resolver un problema como se observa en la Figura 10**.**

Comparativo de Memoria compartida y distribuida

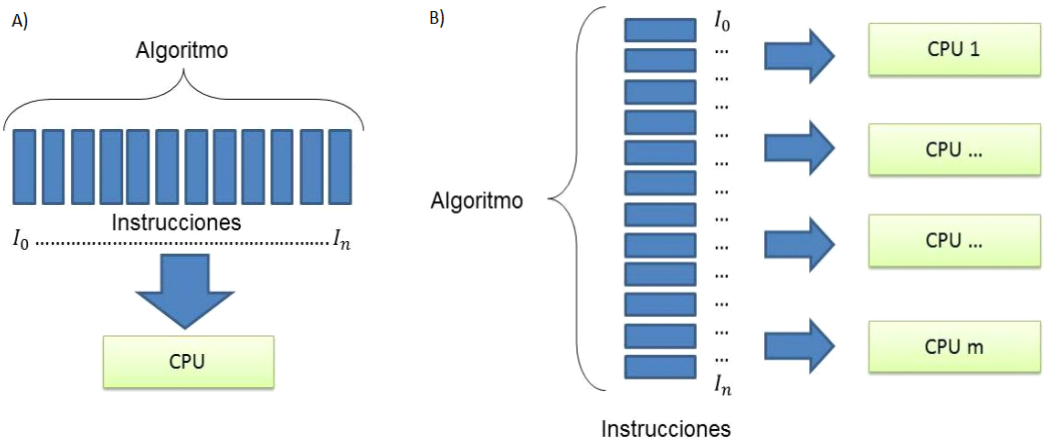


Figura 10 A. Ejecución secuencial de un algoritmo, B. Ejecución simultánea de instrucciones
Fuente: (Rodriguez, 2011)

1.1.4. Procesamiento en paralelo

Memoria distribuida: Es la arquitectura de sistema que posee una colección de computadores, estaciones de trabajos o servidores de forma que se comporte como un único sistema de cómputo. En este tipo de sistema distribuido los usuarios pueden acceder a recursos remotos, ejecutar procesos y luego recibir los resultados procedentes de estos recursos. La Computación Distribuida permite que recursos separados cooperen para resolver tareas que en un único computador no se podrían resolver debido a las limitaciones de recursos de cómputos (Scarpino, 2012).

Memoria compartida o multinúcleos: Es la arquitectura en la cual un conjunto de procesadores comparte la misma memoria, cada procesador puede ejecutar una instrucción diferente. Un procesador multinúcleo es un procesador que incluye múltiples unidades de ejecución (núcleos) en un mismo chip, un núcleo contiene varios hilos de ejecución, los hilos se ejecutan en un único espacio de direcciones, lo que significa que el hardware de direccionamiento de la computadora está configurado para permitir que los hilos lean y escriban en las mismas posiciones de memoria.

1.1.5. Modelo de arquitectura paralelas

1.1.5.1. Taxonomía de Flynn. La taxonomía propuesta por Flynn (2005), es uno de los modelos más destacados y utilizados para clasificar la arquitectura de los computadores. Esta clasificación se basa en la relación que se establece entre los distintos flujos de información en un computador: las instrucciones y los datos. Un flujo de instrucciones es una secuencia de instrucciones ejecutadas en el computador por un único procesador.

Como se puede observar en la Tabla 2. Clasificación de la taxonomía de Flynn se observa un flujo de datos es una secuencia de datos sobre los cuales trabajan las instrucciones y las posibles clasificaciones en que se puede encontrar en marcado un computador.

Tabla 2. Clasificación de la taxonomía de Flynn

	Datos simples	**Datos múltiples**
Instrucciones simples	SISD	SIMD
Instrucciones múltiples	MISD	MIMD

Nota: Esta tabla contiene las instrucciones vs los datos
Fuente: (Briceño, 2014)

- **SISD (Single Instruction, Single Data):** Un flujo de instrucciones único (simple) trabaja sobre un flujo de datos únicos. Solo se realiza una operación a la vez en un solo elemento de datos, por ello no hay paralelismo. Corresponde a la arquitectura clásica (modelo Von Neumann): computadores secuenciales (Briceño, 2014).
- **SIMD (Single Instruction, Multiple Data):** Un flujo de instrucciones único trabaja sobre un flujo de datos múltiples. Corresponde a máquinas que soportan procesamiento vectorial, asignando cada elemento del vector a una unidad funcional diferente para procesarlos concurrentemente. En las máquinas SIMD hay diferentes unidades de procesamiento supervisadas por una única unidad de control. La misma instrucción se ejecuta en varios procesadores sobre datos distintos simultáneamente (Briceño, 2014).
- **MISD (Multiple Instruction, Single Data):** Un flujo de instrucciones múltiple trabaja sobre un flujo de datos único. En esta arquitectura se plantean dos enfoques: i) Las unidades de procesamiento reciben distintas instrucciones para operar sobre el mismo flujo de datos, ii) Un mismo flujo de datos fluye a través de varias unidades de procesamiento.

  El primer enfoque se considera imposible por varios arquitectos de computadores. El segundo enfoque es más viable y como ejemplo, las arquitecturas altamente segmentadas (procesadores vectoriales) suelen considerarse de este tipo. En ambos

casos, los resultados (salida) de un procesador se convierten en los operados (entrada) del siguiente (Briceño, 2014).

- **MIMD (Multiple Instruction, Multiple Data):** Un flujo de instrucciones múltiple trabaja sobre un flujo de datos múltiple. La mayoría de los sistemas multiprocesadores (arquitecturas con memoria compartida) y multicomputadores (arquitecturas con memoria distribuida) se encuentran ubicados dentro de este grupo. Los computadores MIMD tienen varias unidades de procesamiento que pueden ejecutar múltiples instrucciones sobre diferentes datos simultáneamente. Cada unidad de procesamiento tiene su propia unidad de control. Según la distribución de la memoria se pueden categorizar en: memoria compartida o memoria distribuida. Este tipo de máquinas son complejas, pero ofrecen la posibilidad de lograr un alto grado de paralelismo (Briceño, 2014).

1.1.5.2. GPU (Graphics Processing Unit). Es un procesador dedicado al procesamiento gráfico y/u operaciones de coma flotante para liberar la carga del procesador cuando se requiere una gran cantidad de procesamiento gráfico como por ejemplo en aplicaciones o videojuegos que hacen uso del renderizado 3D. Con el aumento de la capacidad de procesamiento de las GPU comparada con las CPU y la reducción del costo, se abre la posibilidad de aprovechar la gran capacidad de procesamiento que tienen las GPU, esto unido a que el procesamiento en GPU utiliza paralelismo (Vance, 2010).

La arquitectura de una GPU está basada en many-cores (muchos núcleos) lo cual permite ejecutar varios procesos en paralelo optimizando el desempeño de aplicaciones que requieren un alto rendimiento. Una GPU utiliza y libera varios hilos de ejecución de forma dinámica, donde cada proceso puede ser ejecutado médiate un

hilo, permitiendo usar los que se encuentren libres si se requiere invocar más procesos, y liberando los demás cuando terminen su ejecución. Como se puede observar en la Figura 11, a diferencia de un CPU, en la arquitectura de una GPU, por cada proceso en paralelo se usa una memoria dedicada (cache) para almacenar información temporal, una unidad de control para manejar e interpretar todas las rutinas que interactúan en cada procesamiento y una unidad aritmética lógica para realizar todas la operaciones aritméticas (suma, resta, multiplicación y división) requeridas para cada proceso ejecutado. En cambio en un CPU, los procesos se establecen sobre un mismo esquema, donde cada uno de ellos es encolado para su respectiva ejecución. (Piccoli M. 2011).

Comparativo CPU vs GPU

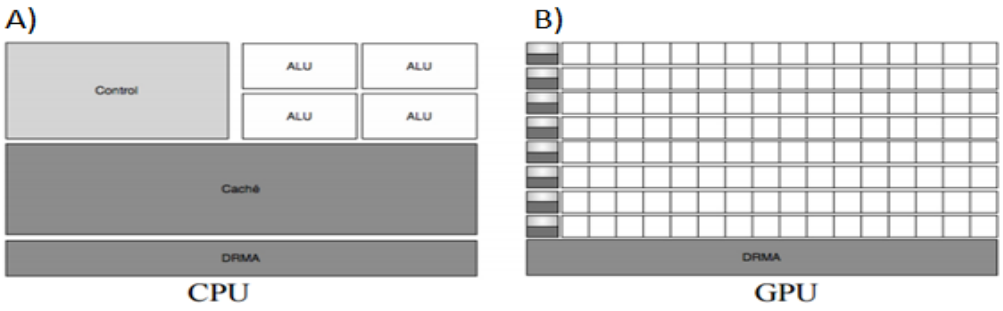


Figura 11 A. Arquitectura CPU, B. Arquitectura GPU
Fuente: (Piccoli, 2011)

1.1.5.3. OpenCL (Open Computing Lenguaje). Es un estándar abierto para la programación de propósito general multiplataforma, de una colección heterogénea de CPU, GPU y otros dispositivos informáticos organizado en una única plataforma. El objetivo de OpenCL es permitir escribir código portable y eficiente. Por lo tanto OpenCL proporciona una abstracción de hardware de bajo nivel, además de un framework de apoyo a la programación.

Como se puede apreciar en la Figura No. 12, la arquitectura de OpenCL está compuesta por un dispositivo que está conformado por unidades de cómputo, las cuales están formadas por varias memorias privadas y una local y una memoria de cómputo que se distribuye en memoria global y constante, donde por cada unidad de cómputo se le asigna parte de las mismas, conformando la memoria cache para todas las unidades de cómputo. Cabe mencionar que los elementos que conforman la arquitectura de OpenCL se explican en los siguientes apartados (Munshi, 2012).

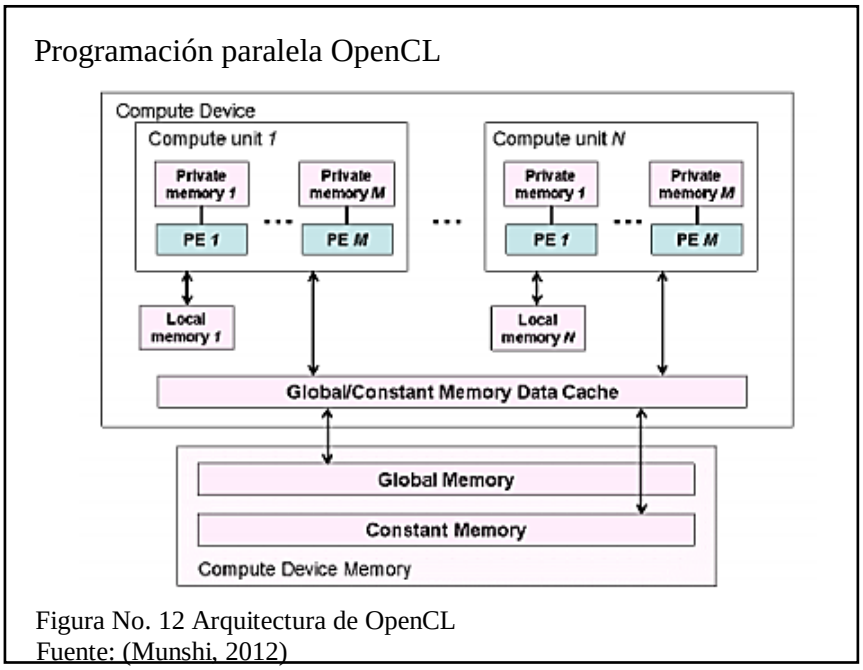


Figura No. 12 Arquitectura de OpenCL
Fuente: (Munshi, 2012)

OpenCL utiliza una jerarquía de modelos:

Modelo de plataforma: El modelo consiste en un host conectado a uno o más dispositivos de computación de OpenCL, cada uno de estos dispositivos están divididos en una o más unidades de computación (UC), que se dividen en uno o varios elementos de procesamiento (PE) (Munshi, 2012).

La Figura 13 muestra el proceso de una aplicación OpenCL donde se envía comandos desde el host (procesador principal del sistema que ejecuta el programa principal) para

ejecutar cálculos en los elementos de procesamiento dentro de un dispositivo. Los elementos de procesamiento dentro de una unidad de procesamiento ejecutan un único flujo de instrucciones como unidades SIMD (ejecutar a la par de una única secuencia de instrucciones) o como unidades SPMD (cada elemento de procesamiento mantiene su propio contador de programa).

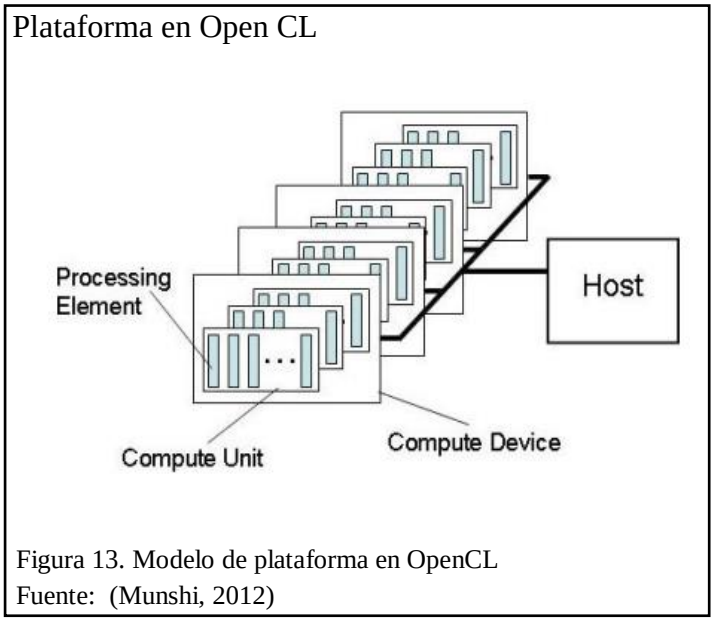


Figura 13. Modelo de plataforma en OpenCL
Fuente: (Munshi, 2012)

Modelo de ejecución: En la Figura 14 muestra el proceso de OpenCL que se ejecuta por medio de un programa principal y por los kernels propios de dispositivos. Es importante recalcar que el programa principal es el encargado de gestionar las ejecuciones de los kernels y los espacios de índices denominados NDRange. En el espacio de índices se ejecutan instancias de kernel las cuales son llamadas wotk-item. Los work-items son recorridos por medio del espacio de nombres como una matriz a partir de un identificador global y otro local. Además los work-items ejecutan un mismo código, con la diferencia de que recorren caminos de ejecución específicos. Existe otro término denominado work-group el cual consiste en definir un conjunto de varios work-items.

Kernel modelo ejecución

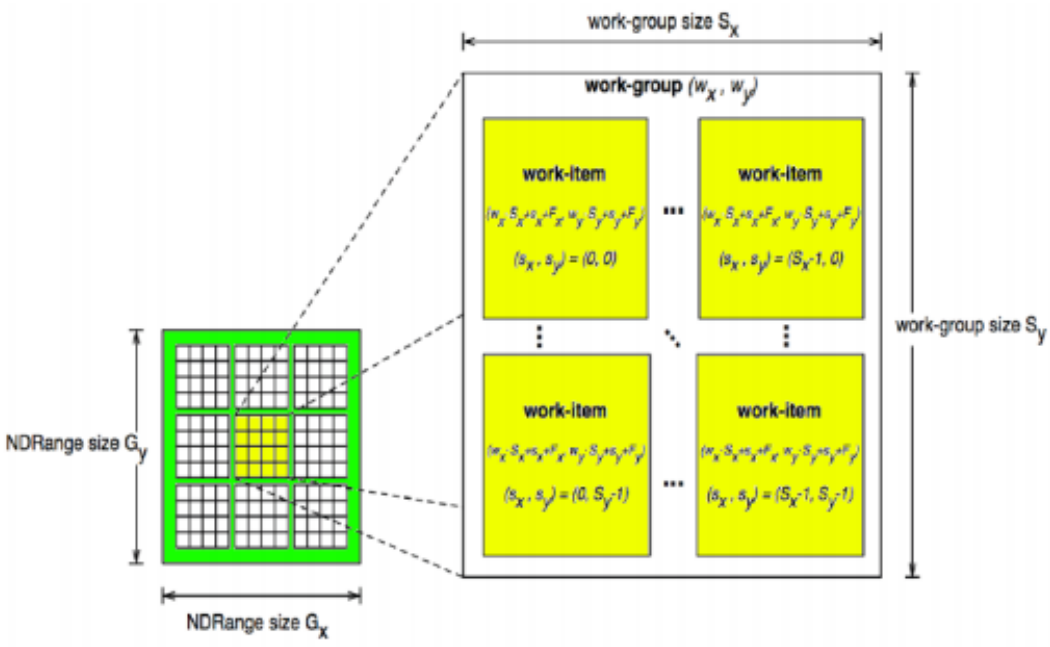


Figura 14 Explicación del NDRange
Fuente: (Munshi, 2012)

Modelo de memoria: A continuación se muestran las cuatro regiones de memoria distintas a los cuales tiene acceso un kernel:

- **Memoria Global**: En caché se puede almacenar las lecturas y escrituras de la memoria global en función de las capacidades del dispositivo.
- **Memoria constante**: Durante la ejecución de un kernel una región de memoria global se mantiene constante. Los objetos de la memoria colocada son asignados e inicializados por el host.
- **Memoria local**: Asigna variables que son compartidas por todos los work-item del work-group que corresponden. Los sectores de memoria global pueden contener regiones de memoria local.
- **Memoria Privada**: Las variables definidas en memoria privada de un work-item no son visibles a otros work- items (Munshi, 2012).

Modelo de programación: Dentro de OpenCL existen 2 modelos de programación los cuales son explicados a continuación.

- **Programación paralela en datos:** Se define como la secuencia de instrucciones ejecutadas a elementos propios de un objeto que se encuentra en memoria. Un kernel puede ser ejecutado en paralelo enlazando uno a uno a los work-items y a los elementos de un objeto en memoria (Munshi, 2012).
- **Programación paralela en tareas:** Esta sección define como se ejecuta una instancia de un kernel sin importar el espacio de índices. En otras palabras, es similar a un proceso de un núcleo que se encuentra en una unidad del CPU con un workgroup que posea un work-item (Munshi, 2012).

Grupos de investigación relacionados con el área de procesamiento de imágenes

El procesamiento digital de imágenes, en sus inicios estaba limitado a unas pocas esferas de investigación: ciencias de la computación, matemáticas y astronomía. Pero, con el avance de las tecnologías y el paso del tiempo, el margen de estudio de esta técnica ha crecido considerablemente.

A continuación se evidencian algunas aplicaciones (Medvis, 2016) (Jähne, 2005):

- Austria - Grupo de informática gráfica universidad técnica de viena. El grupo lleva a cabo una amplia investigación básica y aplicada en gráficos por ordenador (Purgathofer, 2016).
- Alemania – Grupo de análisis y visualización de imágenes médicas, Universidad de Bonn. El grupo sigue un enfoque interdisciplinario que integra los fundamentos teóricos y aplicaciones prácticas de visualización y análisis de imágenes médicas. Trabajos recientes han abordado diversas aplicaciones dentro de la biología, la química y la ingeniería (Schultz , 2013).

- Reino Unido – Grupo de visualización de gráficas médicas (VMG). Es el grupo de investigación más grande dentro de la Facultad de Ciencias de la Computación de la Universidad de Bangor, teniendo las siguientes áreas de investigación: Entornos virtuales, visualización de información, visual analytics, el uso de las interfaces apticas, segmentación, vida artificial, alto artificial, alto rango dinámico de imagen, realidad aumentada (Bangor, 2016).

Trabajos relacionados

1. La Integración de las curvas DER (Densidad Electrónica Relativa) al proceso de verificaciones dosimétricas de un planificador de tratamiento 3D conformadas para radioterapia. Partiendo de la verificación de rangos de desviación entre la dosis planificada en radioterapia 3D Conformacional considerando la incerteza causada por la determinación de Hounsfield (HU). Es posible llevarlo a cabo utilizando el Control de Calidad Dosimétrico de Sistemas de Planificación de Tratamientos (RTPS). Sería necesaria una conversión exacta de Hounsfield (HU) a DER de acuerdo al escáner, para minimizar las incertezas de la dosis a causa de las Hounsfield (HU) especialmente para densidades altas (Muñoz, 2009).
2. El Análisis digital de imágenes tomográficas sin contraste para la búsqueda de tumores cerebrales**.** El diagnóstico de tumores cerebrales con base en imágenes tomográficas sin el uso de medios de contraste, es una tarea difícil de realizar debido a la atención que debe de prestar el especialista a las imágenes y a la necesidad de detenerse con detalle a analizar regiones que a simple vista parecen normales. Con la transformación de imágenes mediante la escala de Hounsfield y la distribución de imágenes médicas (DICOM), es posible realizar la búsqueda de los tumores. Teniendo en cuenta que el tratar con los archivos DICOM permite

poder operar con cualquier tipo de imagen médica sin importar los dispositivos que las generan, pero crea la necesidad de construir un decodificador y visualizador de las imágenes médicas que opere bajo la norma de las imágenes DICOM (Franco, 2011).

3. Algoritmo de segmentación 3D basado en crecimiento de regiones por tolerancia adaptativa y optimización de contraste. El tratamiento de imágenes médicas es crítico cuando se trata de segmentación debido a que se deben obtener patologías y criterios de análisis de manera eficiente. Para resolver este tema se pueden implementar varios métodos de segmentación tridimensional donde intervienen algoritmos de crecimiento de regiones y Hounsfield los cuales se basan en multitolerancia y procesamiento de imágenes TC (Serrano, 2008).
4. Algoritmo de crecimiento de regiones con características de texturas: se trata de una aplicación en biopsias de médula ósea**.** Se debe desarrollar una herramienta que a través de técnicas de procesamiento digital de imágenes, contribuya a la cuantificación de las estructuras presentes para realizar cálculos posteriores. Utilizando para ello el crecimiento de regiones por comparación de texturas para el desarrollo de la aplicación. Se presentó un algoritmo que permite mejorar considerablemente la eficiencia en el reconocimiento de trabéculas que presenta una imagen microscópica de una biopsia, respecto a la segmentación lograda mediante la técnica de crecimiento de regiones por similitud de niveles de gris (Meschino, 2002).
5. La segmentación de imágenes digitales 3D basado en regiones y contornos activos para la generación de mallas de superficie. Uno de los problemas críticos dentro del campo del procesamiento de imágenes y la visualización computacional es la segmentación de imágenes tridimensionales (3D) y la generación de

representaciones geométricas asociadas a las componentes detectadas, como en el caso de estructuras anatómicas en imágenes médicas. Para solventar el problema se utiliza la segmentación por crecimiento de regiones las cuales constituyen un enfoque flexible y poderoso para la segmentación de imágenes, orientado a la búsqueda de regiones homogéneas. El resultado de la segmentación es utilizado para inicializar un contorno deformable, que será adaptado a la estructura buscada dentro de la imagen durante una segunda etapa de segmentación, usando un modelo basado en T-snakes para 2D o T-surfaces en el caso de 3D (Cardona, 2006).

6. Desarrollo de un método de fusión de regiones para segmentación de imágenes. La segmentación de imágenes es uno de los aspectos más importantes de la percepción visual humana y posiblemente la más compleja debido a que las personas usan el sentido visual para dividir el entorno en diferentes objetos y así reconocerlos. Desafortunadamente, no es fácil crear algoritmos artificiales cuya actuación sea comparable a la del sistema visual humano. Sin embargo, el algoritmo de post-procesamiento trata de asemejarse al sentido de la vista. Es por ello, que la implementación de algoritmos de post-procesamiento servirá como desarrollo para futuros trabajos, puede ser la mejora de los algoritmos realizados para el procesamiento de imágenes 2D o 3D, y con la continuación dentro del proceso de tratamiento digital de imágenes (Fajardo, 2009).
7. Implementación del modelo Retinex aplicado al procesamiento de imágenes subacuáticas para mejorar su contenido cromático. Un problema que se observa en los medios electrónicos para la captura de imágenes es que son muy sensibles a las diferentes fuentes de luz y los fenómenos naturales, las imágenes capturadas con cámaras de video en presencia de estos fenómenos muestran dificultades y

deficiencias en el reconocimiento de regiones y áreas de interés para determinadas aplicaciones. Utilizando el modelo Retinex es posible minimizar este problema, ya que Retinex puede ser aplicado a imágenes subacuáticas como una herramienta que realiza la corrección del color. Debido a que el contenido de longitudes de onda que iluminan la escena depende de la profundidad a la que se captura la imagen ocasionando que los colores no correspondan a los colores que se obtienen bajo una iluminación tipo D65 (Dehesa, 2015).

8. Implementación y aplicación de algoritmos Retinex al preprocesamiento de imágenes de retinografía a color. La percepción humana presenta dos capacidades que aún los sistemas de imágenes más sofisticados no han podido imitar: amplitud de rango dinámico (DR) y constancia de color (CC). De aquí, la diferencia que existe entre la calidad que un observador ve en persona de una escena y su correspondiente imagen capturada por un sensor (una cámara fotográfica, por ejemplo) y visualizada en papel o un monitor TRC o LCD. Para minimizar las diferencia se implementa los algoritmos de la teoría Retinex, los cuales tienen como fundamento de desarrollo aspectos relacionados con el sistema visual humano (SVH), el cual percibe de una escena el producto de su reflectancia y la distribución espectral de la iluminación (Durango, 2009).
9. Implementación del modelo Retinex con corrección de color para mejorar la iluminación y el color de imágenes capturadas en condiciones de contaminación ambiental. Para poder capturar imágenes nítidas los diferentes sistemas electrónicos como los encargados del procesamiento de imágenes en donde se realiza el reconocimiento de placas de automóviles, reconocimiento de rostros y otros, es importante tener un grado de visibilidad aceptable, en donde se observen los diferentes elementos de los que está compuesta la imagen. Para ello se usa el

modelo Retinex. Los resultados obtenidos muestran como el algoritmo MSRCR mejora la cromaticidad y la iluminación de imágenes capturadas en ambientes desfavorables como la neblina o la lluvia que filtran la luz y no permite capturar el color de los diferentes objetos que se encuentran presentes en la escena (Rosales & Silva, 2015).

10. Sistema experto para la selección de algoritmos de constancia de color. El sistema visual humano es limitado, ya que depende de las condiciones de luz para poder distinguir claramente los colores. Afortunadamente, la mayoría de la veces, los colores de una superficie en particular, parecen ser los mismos, en interiores o exteriores, en la mañana o en la tarde, a este fenómeno se le conoce como constancia de color. La teoría Retinex nos ayuda a disminuir la diferencia de colores ante la vista humana. Se ha diseñado el sistema experto utilizando conjuntos difusos de tipo triangular y de tipo Gaussiano. Siendo estos últimos los que proporcionan mucho mayor desempeño al sistema (Cepeda, 2012).

CAPÍTULO II

Metodología

En el capítulo 2 se detallan todos los elementos necesarios para la ejecución del algoritmo, como son el hardware, software, data set utilizado y las variables con las que se realizará el análisis de los resultados.

2.1. Paso 1: Prerrequisitos

Hardware

Para este trabajo se utilizó memoria RAM DIMM Kingston de 16GB DDR3 1333 Mhz, un CPU con procesador Intel Core I7-3517u de 2 núcleos con 4 MB de cache y velocidad en bus de 5 GT/s para Asus S46c de sexta generación, y una Tarjeta NVIDIA QUADRO 4000 con 256 núcleos CUDA con memoria de 2 GB de tipo GDDR5.

Software

Centos 7: Se ha instalado el sistema operativo Centos 7 junto con las actualizaciones propias del hardware y posteriormente se ha configurado los componentes previos para el desarrollo. Los drivers utilizados para la configuración en Centos 7 fueron NVIDIA Kernel Module 367.27, CUDA 7.5 y gcc 4.8.5. Cabe mencionar que toda la información requerida para la instalación y configuración en Linux, puede ser apreciada en los anexos del presente trabajo (Ver anexo 2).

Windows: De igual forma que en el apartado anterior, se ha instalado el sistema operativo Windows 8 junto con las actualizaciones propias del hardware y posteriormente se ha configurado los componentes previos para el desarrollo.

Los drivers utilizados para la configuración en Windows 8 fueron OpenCV 12 y Cuda ToolKit 8. Cabe mencionar que toda la información requerida para la instalación y configuración en Windows, puede ser apreciada en los anexos del presente trabajo (Ver anexo 3).

2.2. Paso 2: Clúster

Herramientas para infraestructura

ESXI es un hypervisor de VMWare, conocido también como vSphere en el ámbito comercial. Esta herramienta establece una capa robusta de virtualización permitiendo montar varias máquinas virtuales en un host físico, sin la necesidad de que el mismo cuente con un sistema operativo. Cabe mencionar que ESXI brinda una arquitectura que facilita el control y mantenimiento sobre infraestructuras virtuales, tomando en cuenta la simplificación de actualizaciones, donde la gestión de los procesos se los maneja mediante líneas de comando de forma remota (Añazco, 2016).

Como se muestra en la Figura 15, la arquitectura de ESXI establece el uso inmediato de los componentes del hardware de un servidor físico, como es el caso del CPU (*Unidad Central de Procesos*), la memoria RAM, tarjetas de red, discos duros, entre otros. Los drivers de instalación de ESXI son montados directamente en el hardware, donde se aprovecha el uso y se controla de mejor manera los recursos, ya que no es necesario la instalación de sistemas operativos. Al momento de virtualizar, cada máquina acoplada sobre ESXI contará con los drivers respectivos de forma independiente, donde los componentes del hardware se distribuyen conforme se aumente o disminuya la cantidad de máquinas virtuales instaladas.

Gráfica ESXI

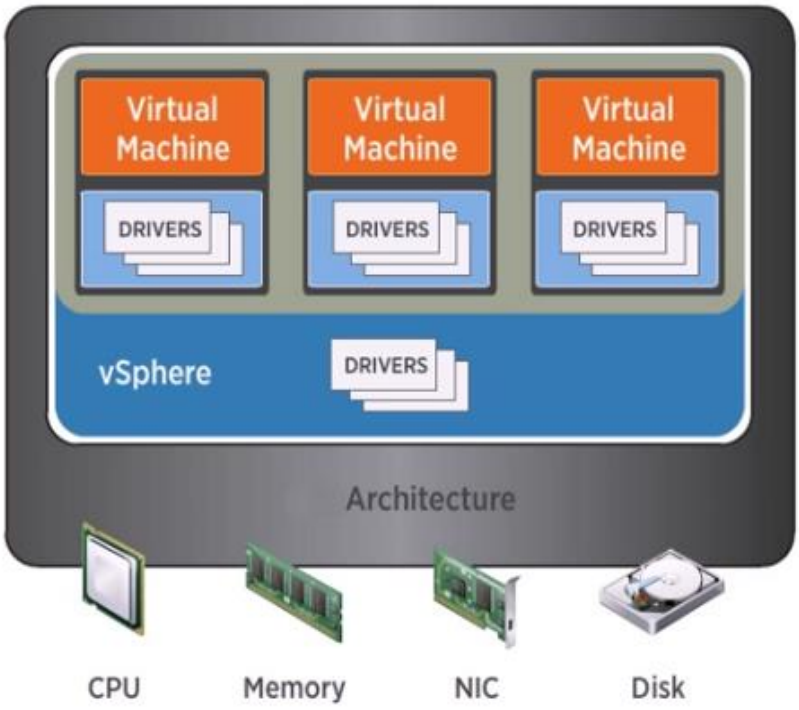


Figura 15. Arquitectura ESXI
Fuente: (VMWARE, 2016)

Para solucionar esta fase del trabajo se ha instalado la plataforma de virtualización VMWare ESXI 6.0 donde se realizó la configuración de un clúster virtual con varios clientes, los cuales tendrán instalado Centos 7. Así con dicha virtualización los clientes configurados podrán consumir los recursos de hardware del propio servidor. El servidor tiene una tarjeta NVIDIA QUADRO 4000 y es necesario instalar el driver propio de la tarjeta, pero dicho driver no existe para la versión de ESXI 6.0 tomando en cuenta que el proceso fue debidamente probado como se puede visualizar en el anexo (Ver Anexo 1), pero sin embargo no fue posible realizar las configuraciones con éxito por la incompatibilidad existente entre el ESXI 6.0 y el driver de la tarjeta NVIDIA QUADRO 4000 es por ello que se deja como referencia un documento que sustenta la incompatibilidad (VMWare, 2016).

2.3. Paso 3: DataSet

Para realizar las pruebas del algoritmo Retinex se toma dos DataSet que poseen los siguientes datos: i) nombre: Toutatix, modalidad: CT, tamaño del archivo: 195 MB, descripción: Anomalía de la arteria coronaria, ii) nombre: Volumerge, modalidad: MR, tamaño del archivo: 21 MB, descripción: Espina dorsal. En la Figura 16 se pude identificar la previsualización de los DataSet (Osirix, 2017).

Imágenes DataSet

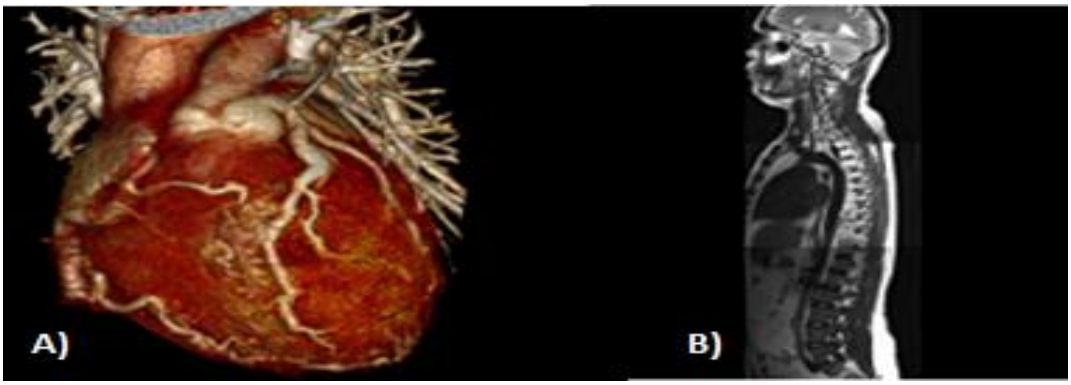


Figura 16. A. Toutatix, B. Volumerge
Fuente: (Osirix, 2017)

De Toutatix y Volumerge se toma la secuencia de imágenes desde IM-0001-0001 hasta IM-0001-00011 de las carpetas: *CorCTA 0.75 B25f Diastole 65%* y *MobiView – 305* respectivamente, dichas imágenes se renombran para nuestra experimentación con la nomenclatura: primera letra del nombre del DataSet, seguido por el texto "_img" y por último el número que le corresponde a la secuencia ejemplo: t_img1 para el caso de Toutatix y v_img1 para Volumerge.

Adicional se selecciona otro grupo de imágenes subacuáticas con poca luminosidad que son las mismas imágenes del estudio realizado en el artículo titulado *Implementación del modelo Retinex aplicado al procesamiento de imágenes subacuáticas para mejorar su contenido cromático.* Datos de las imágenes: i) nombre:

image8-1, dimensiones: 500x416, ii) nombre: image16-1, dimensiones: 500x454, iii) nombre: image2-1, dimensiones: 500x418 (Dehesa, 2015).

2.4. Paso 4: Algoritmo

A continuación se describe paso a paso la implementación del algoritmo Retinex, sobre la plataforma OpenCL y los dispositivos GPU instalados en el sistema, siendo el principal la tarjeta NVIDIA QUADRO 4000 para el procesamiento de imágenes.

Lo descrito a continuación es la fórmula tomado del trabajo (Petro, 2014) el cual lo ha llamado MSRCR (Retinex multiescala con restauración de color). La forma básica de Retinex está dado por:

$$R_i(x_i, x_2) = \sum_{k=1}^{k} W_k(\log I_i(x_i, x_2) - \log(F_k(x_i, x_2) \otimes I_i(x_i, x_2))) \quad [2.1]$$

La ecuación [2.1] nos representa un algoritmo que es capaz de reducir los cambios bruscos de brillo, color y saturación. En donde i nos representa el canal actual, $\otimes$ expresa la convolución, log es la función logaritmo natural, (x_i, x_2) es la coordenada espacial del pixel, l_i es la imagen actual, R_i es la salida del proceso W_k, es el peso asociado con F_k, K es el número de función envolvente, en donde la función envolvente F_k está definida por la ecuación (2) (Rahman, Jobson, & Woodell, 2004).

$$F_k(x_i, x_2) = K\, exp[-(x_1^2 + x_2^2)/\sigma_k^2] \quad [2.2]$$

Donde, σ_k es la desviación estándar típica de las envolventes Gaussianos, y su magnitud controla la extensión de la envolvente, y toda la función es normalizada según K. Para medir la eficiencia de mejoramiento del color para el método Retinex propuesto, se utilizará un criterio de evaluación de cromaticidad que nos permita medir la magnitud del vector de cromaticidad en el espacio de color CIELab.

Retinex se encuentra basado en los conceptos de convolución gaussiana para lo cual se describe a continuación su proceso. La convolución gaussiana es una operación matemática formada de tres funciones.

La primera función Gaussiana tiene la forma:

$$f(\mathrm{x}) = a * e - \frac{(x-b)^2}{2c^2} + d \qquad [2.3]$$

Donde:

$a \rightarrow$ Altura de la curva.

$b \rightarrow$ Posición del centro.

$c \rightarrow$ Desviación estándar, controla el ancho de la curva.

La segunda función se define por:

$$g(\mathrm{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{x^2}{2\sigma^2}} \qquad [2.4]$$

Donde:

$x \rightarrow$ Posición en el eje X.

$\sigma \rightarrow$ Desviación estándar.

La tercera ecuación es la combinación de la primera y la segunda función

$$F(x) = f(x) \otimes g(x) \qquad [2.5]$$

Donde, en el procesamiento de imágenes:

$f \rightarrow$ Función que se puede tomar como una imagen de entrada.

$g \rightarrow$ Matriz de convolución.

$\otimes \rightarrow$ Operador de convolución.

Si la segunda función (el filtro gaussiano) se aprecia como Error, su definición es:

$$\mathrm{g(x)} = \frac{1}{\sqrt{2\pi\sigma^2}} * \mathrm{e}^{-(\frac{x^2-y^2}{2\sigma^2})} \qquad [2.6]$$

Dónde: *y* es la posición en el eje Y.

En la Figura 17 se detalla una matriz 5x5 con los pixeles de una imagen y un matriz 3x3 llamado máscara para tener una matriz resultante de 5x5, el contorno verde en este caso es la máscara escogida para realizar el proceso y obtener como resultado un valor que será reemplazado por el pixel del centro también llamado pixel inicial, el cálculo se realiza multiplicando cada pixel por el valor correspondiente de la máscara y se suman los resultados para reemplazar el pixel inicial.

Ejemplo: (40*0) + (42*1) + (46*0) + (46*0) + (50*0) + (55*0) + (52*0) + (56*0) + (58*0) = 42

Cálculo del valor de cada pixel.

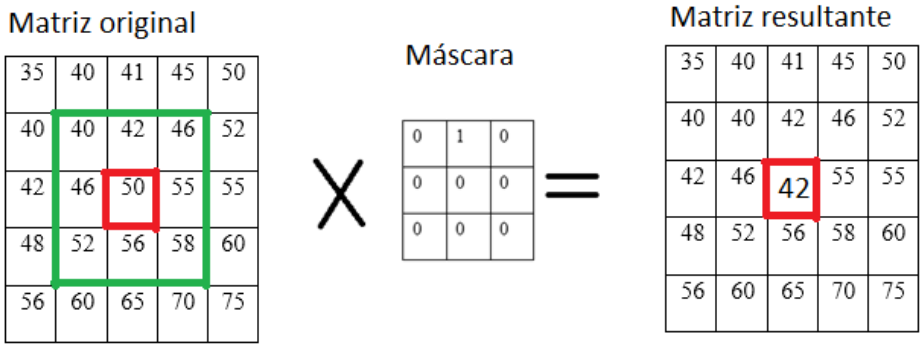


Figura 17. Convolución gaussiano
Elaborado por: Jessica Parraga y Luis Casa

Finalmente se ejecuta el algoritmo de Retinex para lo cual se procede a colocar el pseudocodigo que se muestra a continuación. Los resultados de la ejecución de Retinex se pueden observar en la Figura 18. Donde se puede apreciar que las condiciones de la neblina no permiten observar la imagen claramente, y al ser procesada por el algoritmo Retinex se observa la mejora en la Figura 18. Para evidenciar las mejoras de la imagen aplicando el algoritmo se utiliza histogramas en escala de grises.

Datos: I, Imagen de color de entrada; σ1, σ2, σ3 las escalas; s1, s2 el porcentaje de pixeles de recorte en cada lado.

Resultado: Salida de imagen a color

Empezar

para cada c ∈ {R, G, B} **hacer** //Para cada canal de color

para cada σi **hacer** //Para cada escala

$Diff_{i,c} = \log(I_c) - \log(I_c * G_{\sigma i})$ //Retinex escala única

fin

$MSR_c = \sum_i \frac{1}{3} Diff_{i,c}$ //Retinex Multi escala

$MSRCR_c = MSR_c * (\log(125 I_c) - \log(I_R + I_G + I_B))$

$Out_c = \text{SimplestColorBalance}(MSRCR_c, s_1, s_2$ //Restauración de color

fin

fin

Ejemplo del proceso del algoritmo Retinex

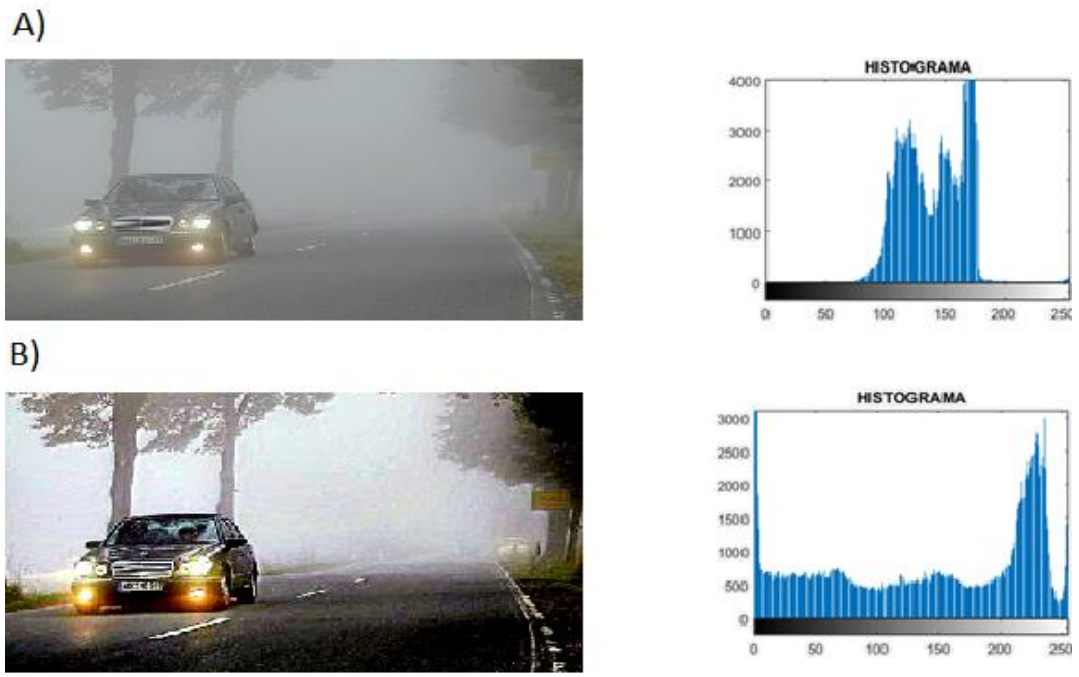


Figura 18 A. Original con neblina. B. Resultado Retinex neblina
Elaborado por: Jessica Párraga y Luis Casa

Configuración paralela

La línea $MSRCR_c = MSR_c * (\log(125I_c) - \log(I_R + I_G + I_B))$ del algoritmo es donde se realiza el filtrado en varias escalas y se ha mantenido la idea original del funcionamiento del MSRCR, pero antes de llamar a la función se hace el uso del contexto OpenCL para crear el kernel de tareas paralelas y después de llamar a la función MSRCR se liberan los kernels que han sido creados, como se evidencia en fragmentos de código que se presentan a continuación y así ver los procesos que se han paralelizado.

1) Dentro del metodo main se escribe el codigo para inicializar las estructuras del datos OpencCL.

```
int devType = CL_DEVICE_TYPE_GPU;
```

2) A continuacion se presenta la estructura de datos del Programa o declaracion de variables OpenCL.

```
cl_platform_id cpPlatform; // Plataforma OpenCL
cl_device_id device_id;    // identificador de dispositivo de cómputo
cl_context context;        // contexto de cómputo
cl_command_queue  command_queue = NULL; // cola de comandos de cómputo
//cl_program
// Usar la variable status para chequear la salida para cada llamada del API
cl_int status;
cl_int err;               // código de retorno desde las llamadas a las apis OpenCL

//Reserva Kernels y ubica datos
cl_kernel kernel[5]        = { NULL, NULL, NULL, NULL };
cl_int i = 0, j = 0;
```

3) Luego buscar e inicializar todos los dispositivos y drivers disponibles

```
//Acceder a los drivers instalados y conectar con dispositivos de cómputo
err = clGetPlatformIDs(1, &cpPlatform, NULL);
if (err != CL_SUCCESS) {
         cerr << "Error: Fallo buscando plataformas!" << endl;
         return EXIT_FAILURE;
}
//Obtener un dispositivo de cómputo del tipo apropiado
err = clGetDeviceIDs(cpPlatform, devType, 1, &device_id, NULL);
if (err != CL_SUCCESS) {
         cerr << "Error: Fallo creando el grupo de dispositivos!" << endl;
         return EXIT_FAILURE;
}
cout << "---------Datos del dispositivo GPU y OpenCL---------" << endl;
//Imprimir los datos del dispositivo
printDeviceInfo(device_id);
cout << endl;
```

4) Despues se crea el contexto OpenCL asociado al dispositivo

```
//Crear el contexto OpenCL
context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);
if (!context) {
  cerr << "Error: Fallo creando el contexto para el dispositivo!" << endl;
  return EXIT_FAILURE;
}
```

5) Crear cola de comandos que será enviados al dispositivo

```
// Creación de una cola de comandos
command_queue = clCreateCommandQueue(context, device_id, 0, &err);
if (!command_queue) {
         cerr << "Error: Fallo creando la cola de comandos!" << endl;
         return EXIT_FAILURE;
}
```

6) Crear el programa y compilar el ejecutable

```
//Estructura de datos Program/kernel
//Lectura del fichero del programa  mscrc_kernel.cl que contiene los kernel en el
buffer
cl_program program;
//Leer el fichero del programa  y ubicarlo dentro del buffer
// Crear el kernel a partir de los fuentes

program = clCreateProgramWithSource(context, 1, (const char **)&KernelSource, NULL,
&err);

if (!program) {
         cerr << "Error: Fallo al crear el programa!" << endl;
         return EXIT_FAILURE;
}
// Construir el programa ejecutable
err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
if (err != CL_SUCCESS) {
         size_t len;
         char buffer[2048];
         cerr << "Error: Fallo al construir el programa ejecutable!" << endl;
         clGetProgramBuildInfo(program,                                    device_id,
CL_PROGRAM_BUILD_LOG,sizeof(buffer), buffer, &len);
         cerr << buffer << endl;
         exit(1);
}
```

7) Crear Kernel OpenCL

```
kernel[0] = clCreateKernel(program, "operationsmath", &err);
if (!kernel || err != CL_SUCCESS) {
       cerr << "Error: Fallo creando el kernel!" << endl;
       exit(1);
}
kernel[1] = clCreateKernel(program, "imageoperationsinit", &err);
if (!kernel || err != CL_SUCCESS) {
       cerr << "Error: Fallo creando el kernel!" << endl;
       exit(1);
}
kernel[2] = clCreateKernel(program, "imageoperationsfinal", &err);
if (!kernel || err != CL_SUCCESS) {
       cerr << "Error: Fallo creando el kernel!" << endl;
       exit(1);
```

```
}
kernel[3] = clCreateKernel(program, "compute_mean_var", &err);
if (!kernel || err != CL_SUCCESS) {
       cerr << "Error: Fallo creando el kernel!" << endl;
       exit(1);
}
kernel[4] = clCreateKernel(program, "gausssmooth", &err);
if (!kernel || err != CL_SUCCESS) {
       cerr << "Error: Fallo creando el kernel!" << endl;
       exit(1);
}
```

8) Inicializar los argumentos de los kernel

```
// Crear datos para la ejecución del primer kernel
// Llenar el vector con los valores que se emplearán en el cálculo
// equivalente a la dimensión original de la imagen original
// Ejecución del kernel dentro del dispositivo y lectura
err = clSetKernelArg(kernel[1], 0, sizeof(cl_mem), &ksrc);
err |= clSetKernelArg(kernel[1], 1, sizeof(cl_mem), &kdst);
err |= clSetKernelArg(kernel[1], 2, sizeof(cl_int), &knChannels);
err |= clSetKernelArg(kernel[1], 3, sizeof(cl_int), &kstep);
err |= clSetKernelArg(kernel[1], 4, sizeof(cl_int), &knHeight);
err |= clSetKernelArg(kernel[1], 5, sizeof(cl_int), &knWidth);
if (err != CL_SUCCESS) {
cerr << "Error: Fallo en inicializando los argumentos del kernel! " << err << endl;
       exit(1);
}
```

9) Ejecutar el proceso Retinex La línea $MSRCR_c = MSR_c * (\log(125I_c) - \log(I_R + I_G + I_B))$

```
status = retinex_process(sFilename,dFilename);
if (status == 0) cout << "\nproceso retinex exitoso\n";
// Mostrar la imagen procesada empleando el método retinex
cvShowImage("Imagen procesada (Retinex)", dst);
if (rFilename != NULL)
       cvSaveImage(rFilename, dst);
```

10) Finalmente liberar todos los recurso OpenCL reservados

```
// Liberar las reservaciones del entorno OpenCL
// Esperar por el completamiento de todos los comandos
clFinish(command_queue);
clReleaseProgram(program);
for (i = 0; i <= 4; i++) {
       clReleaseKernel(kernel[i]);
    }
clReleaseCommandQueue(command_queue);
clReleaseContext(context);

time(&end_timer);
seconds = difftime(end_timer, init_timer);

cout << endl << "La imagen original es de:" << endl;
cout << endl <<("El proceso ha durado %.f segundo(s)", seconds) << endl;
cout << "Presionar cualquier tecla para finalizar" << endl;

cvWaitKey(0);
return 0;
```

Cabe mencionar que el aporte realizado fue el uso de tecnología OpenCL y tarjeta NVIDIA QUADRO 4000 para mejorar el tiempo de respuesta del procesamiento de imágenes médicas.

2.5. Paso 5: Definición de variables

Tabla 3. Estudio de variables

Etiqueta	Nombre variable	Tipo de variable	Referencia/tema tesis o artículos
TEPM	Tiempo de Ejecución con relación a los puntos de mallado	Tiempo	Análisis del rendimiento de algoritmos paralelos de propósito general en GPGPU. Aplicación a un problema de mallado de elementos finitos (Gaudiani, 2012).
TETD	Tiempo de ejecución y transferencias de datos	Tiempo	Análisis del rendimiento de algoritmos paralelos de propósito general en GPGPU. Aplicación a un problema de mallado de elementos finitos (Gaudiani, 2012).
ETP	Ejecución total del proceso	Tiempo	Análisis del rendimiento de algoritmos paralelos de propósito general en GPGPU. Aplicación a un problema de mallado de elementos finitos (Gaudiani, 2012).
IM	Iteraciones del método que contenga el algoritmo seleccionado	Numero de	Análisis del rendimiento de algoritmos paralelos de propósito general en GPGPU. Aplicación a un problema de mallado de elementos finitos (Gaudiani, 2012).
NHB	Hilos por bloque durante el proceso	Procesador	Estudio de rendimiento en GPU (Juega, 2010).

NBA	Número de Bloques activos durante el proceso	Procesador	Estudio de rendimiento en GPU (Juega, 2010).
NHDP	Número de hilos utilizados durante el proceso	GPU	Implementación, paralelización y evaluación de una aplicación de tomografía 3D basada en C++ e Intel TBB (Calvo, 2012).

Nota: Esta tabla contiene las variables que se usaran para las métricas

CAPITULO III

Resultados

En el capítulo 3 se documentan los datos obtenidos en las ejecuciones de las imágenes de pruebas seleccionadas, para posteriormente realizar el respectivo análisis del rendimiento del algoritmo.

3.1 Pruebas imágenes

En la ejecución de las siguientes pruebas se toma los archivos detallados en la sección DataSet donde se realizó los siguientes pasos: i) Procesamiento de la imagen original con el algoritmo Retinex, ii) Con la ayuda de MatLab R2015B, se genera la imagen en escala de gris de la imagen original y la procesada por el algoritmo Retinex, iii) Por último se crea los histogramas de las imágenes obtenidas en el paso ii. Como experimento inicial procedemos con el grupo de imágenes del trabajo *Implementación del modelo Retinex aplicado al procesamiento de imágenes subacuáticas para mejorar su contenido cromático* que se evidencia en las figuras: Procesamiento de imagen de estudio previo con algoritmo Retinex 1

Figura 19, Figura 20 y Figura 21.

Procesamiento de imagen de estudio previo con algoritmo Retinex 1

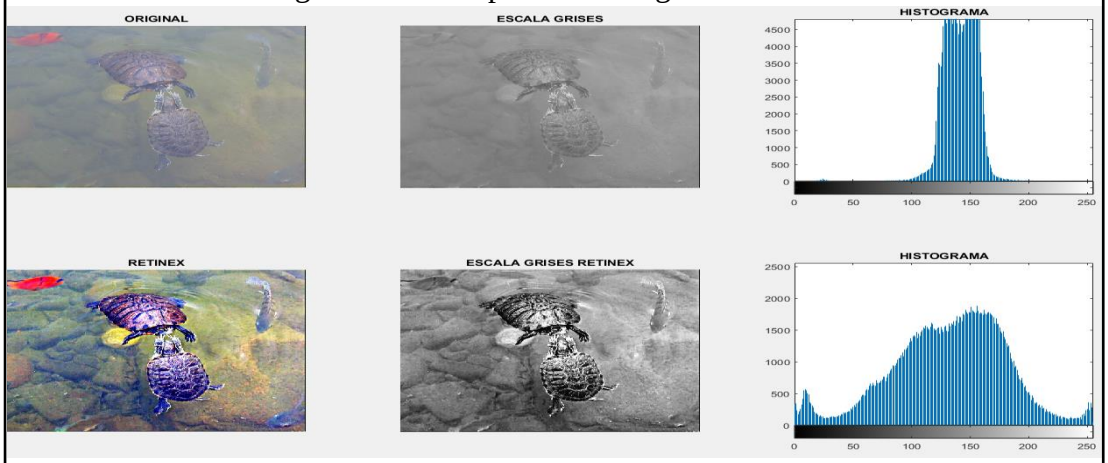


Figura 19. Imagen subacuática tortugas con proceso Retinex e histograma (image8-1)
Fuente: (Dehesa, 2015)

Procesamiento de imagen de estudio previo con algoritmo Retinex 2

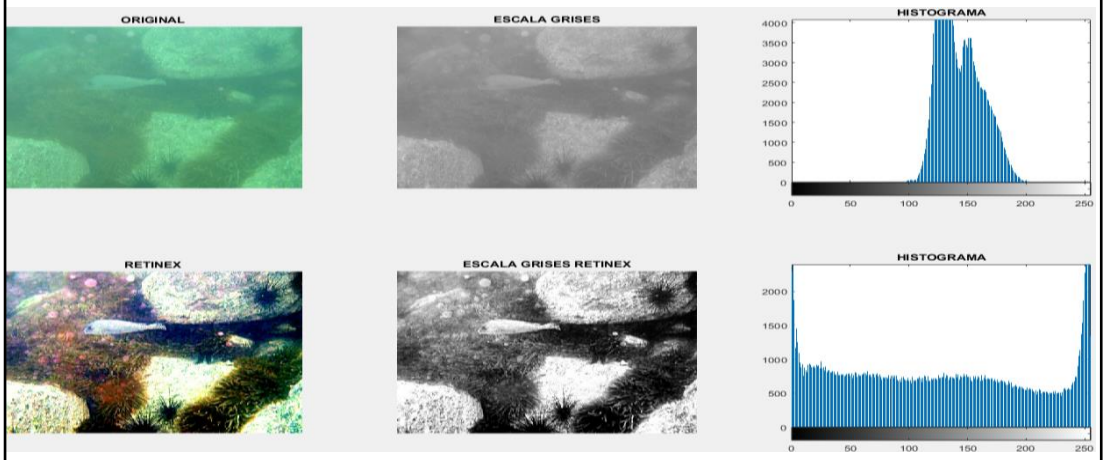


Figura 20. Imagen subacuática pez con proceso Retinex e histograma (image16-1)
Fuente: (Dehesa, 2015)

Procesamiento de imagen de estudio previo con algoritmo Retinex 3

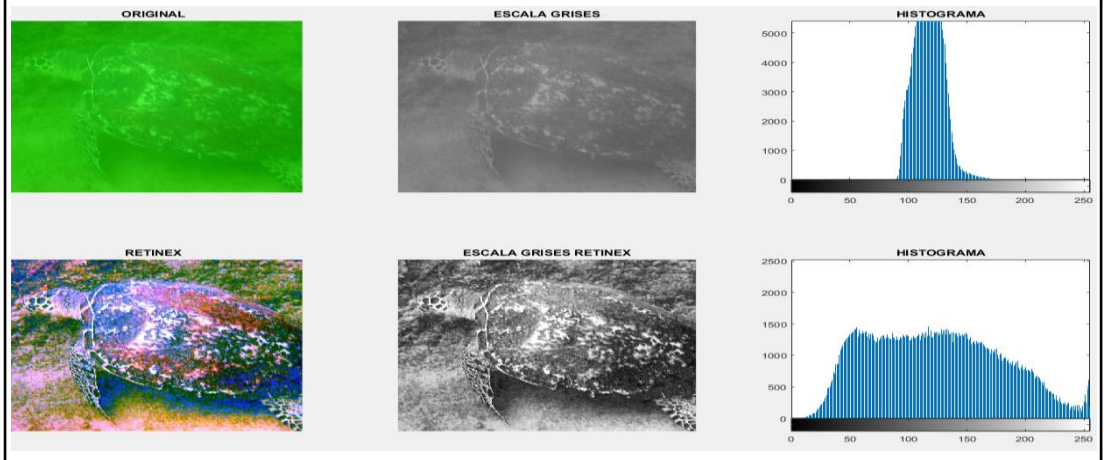


Figura 21. Imagen subacuática tortuga con proceso Retinex e histograma (image2-1)
Fuente: (Dehesa, 2015)

En una primera fase se obtiene los datos del experimento realizado por Dehesa et al. en el año 2015 con el tema: Implementación del modelo Retinex aplicado al procesamiento de imágenes subacuáticas para mejorar su contenido cromático, que son el resultado de la ejecución de Retinex en secuencial en CPU. Con el fin de realizar una comparación se utilizaron las imágenes: image8-1, image16-1 y image2-1 del experimento realizado en la primera fase. En una segunda fase se procedió a reproducir el experimento de Retinex en secuencial con CPUs utilizando la tecnología descrita previamente en la sección de prerrequisitos. Finalmente se procedió a ejecutar el experimento de Retinex en paralelo y con GPUs. Los datos descritos anteriormente se expresan en la Tabla 4, en la cual se puede observar el tiempo de ejecución de cada uno de los experimentos.

Tabla 4. Tiempos de ejecución algoritmo Retinex secuencial y paralelo

Nombre imagen	**Tamaño pixel**	**Tiempo ejecución Retinex secuencial paper segundos**	**Tiempo ejecución Retinex secuencial experimento segundos**	**Tiempo ejecución Retinex paralelo experimento segundos**
image8-1	500 x 416	6,4	2,8	1,0
image16-1	500 x 454	6,2	3,1	1,0
image2-1	500 x 418	6,5	2,7	1,0

Nota: La tabla muestra los tiempos de ejeución de las pruebas realizadas

Una vez realizadas las pruebas comparativas del algoritmo Retinex en paralelo, se procede a la experimentación utilizando tomografías computacionales (TAC), representadas por los DataSet denominados Volumerge y Toutatix, la razón por la cual trabajamos con estos archivos es debido a su estructura, formada por un conjunto de frames e información suficiente para visualizar un entorno 3D.

Para visualizar el entorno 3D formado por el DataSet utilizamos la aplicación RadiAnt DICOM Viewer v3.4.2.13370 en la que podemos visualizar el ejemplo de Toutatix en 3D en la Figura 22.

Entorno 3D con RadiAnt DICOM Viewer

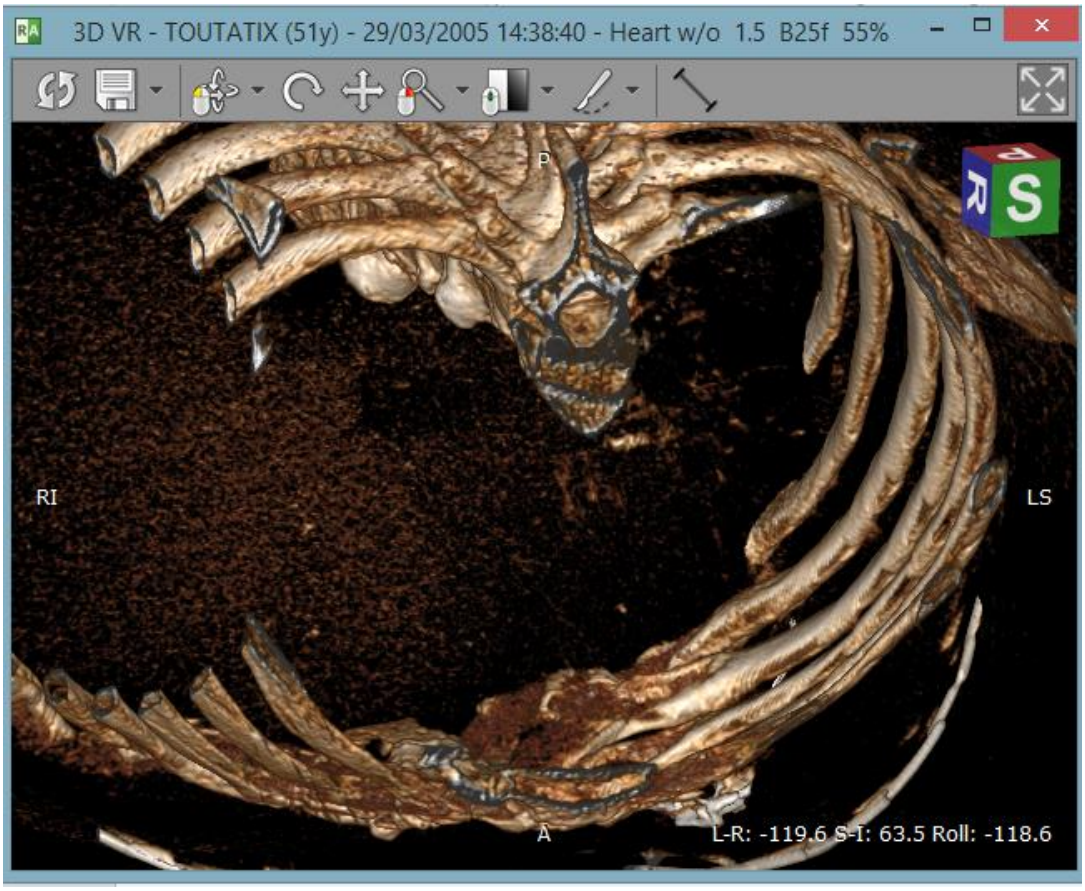


Figura 22. Entorno 3D del DataSet Toutatix
Fuente: Elaborado por: Luis Casa y Jéssica Párraga

En la Figura 23 y Figura 23 se genera la imagen en escala de gris y el histograma tanto para el frame original como también para el frame procesado por Retinex, los frames antes mencionadas corresponden al primero y último de la secuencia del DataSet de la tomografía computacional denominada Volumerge, la sección completa de once frames procesados por el algoritmo Retinex se encuentran en el presente trabajo (Ver anexo 4).

Por otra parte utilizando la secuencia de la DataSet de Volumerge se realizó la experimentación para ingresar los datos en la

Tabla 5 donde tenemos el tamaño en pixeles de cada frame y la sumatorio del tamaño en KB, además de los correspondientes tiempos de ejecución del algoritmo Retinex en secuencial CPUs y paralelo GPUs, se generó como resultado la Figura 25.

Pruebas DataSet Volumerge:

Procesamiento de imagen con algoritmo Retinex espina dorsal 1

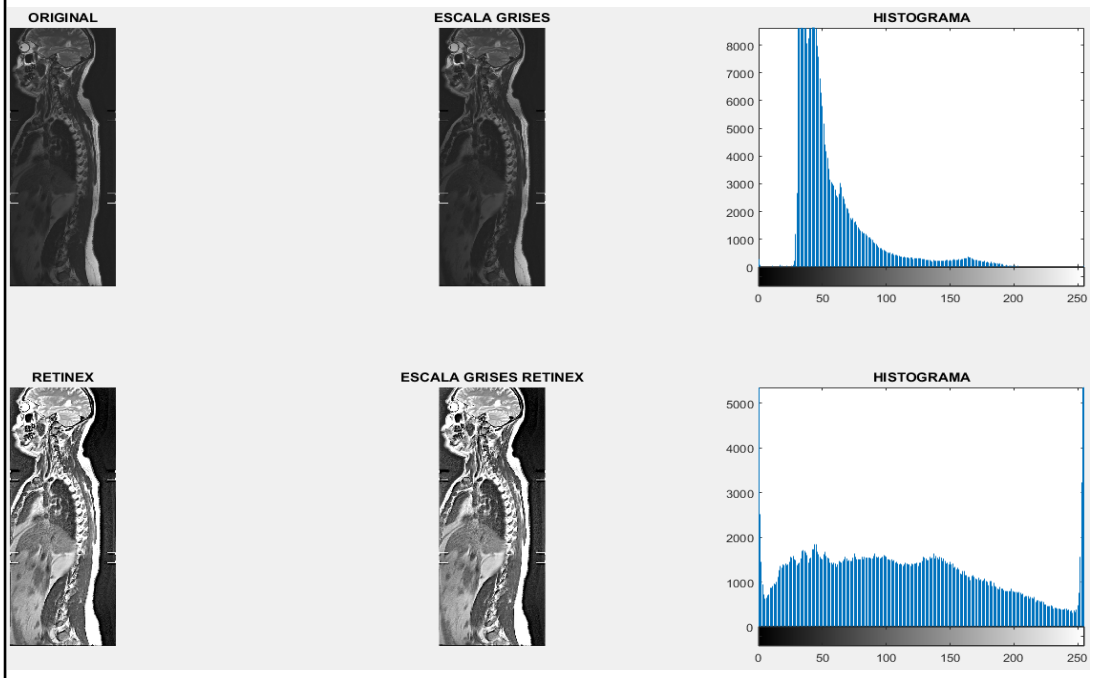


Figura 23. Espina dorsal con proceso Retinex e histograma (v_img1)
Fuente: Elaborado por: Luis Casa y Jéssica Párraga

Procesamiento de imagen con algoritmo Retinex espina dorsal 2

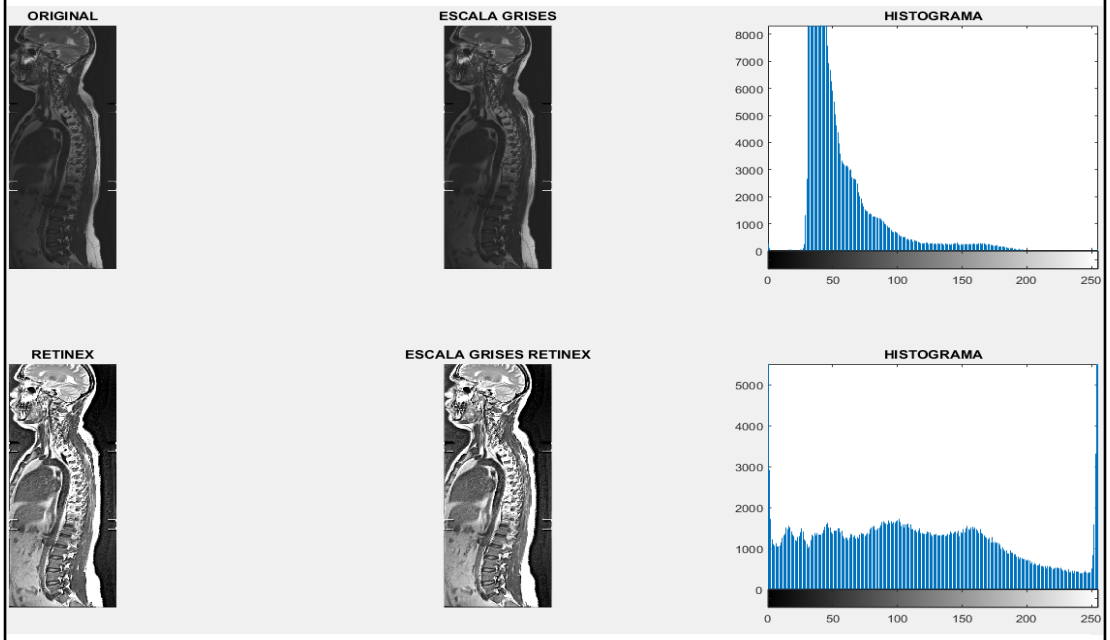


Figura 24. Espina dorsal con proceso Retinex e histograma (v_img11)
Fuente: Elaborado por: Luis Casa y Jéssica Párraga

Tabla 5. Datos proceso Retinex secuencial y paralelo en tomografía computacional

Tamaño	Nombre imagen	Tamaño pixel	Total datos procesados kb	Tiempo ejecución Retinex secuencial segundos	Tiempo ejecución Retinex paralelo segundos
Grande	v_img1	560 x 1558	2.996	24,84	12
	v_img2	560 x 1558			
	v_img3	560 x 1558			
	v_img4	560 x 1558			
	v_img5	560 x 1558			
	v_img6	560 x 1558			
	v_img7	560 x 1558			
	v_img8	560 x 1558			
	v_img9	560 x 1558			
	v_img10	560 x 1558			

	v_img11	560 x 1558			
Mediano	v_img1_mediano	350 x 974	838	13,7	6
	v_img2_mediano	350 x 974			
	v_img3_mediano	350 x 974			
	v_img4_mediano	350 x 974			
	v_img5_mediano	350 x 974			
	v_img6_mediano	350 x 974			
	v_img7_mediano	350 x 974			
	v_img8_mediano	350 x 974			
	v_img9_mediano	350 x 974			
	v_img10_mediano	350 x 974			
	v_img11_mediano	350 x 974			
Pequeño	v_img1_pequeño	320 x 890	729	9,10	4
	v_img2_pequeño	320 x 890			
	v_img3_pequeño	320 x 890			
	v_img4_pequeño	320 x 890			
	v_img5_pequeño	320 x 890			
	v_img6_pequeño	320 x 890			
	v_img7_pequeño	320 x 890			
	v_img8_pequeño	320 x 890			
	v_img9_pequeño	320 x 890			
	v_img10_pequeño	320 x 890			
	v_img11_pequeño	320 x 890			

Nota: Esta tabla contiene los diferentes tamaños de las imágenes procesadas, así como también los tiempos de ejecución en forma paralela y secuencial de la tomográfia computacional

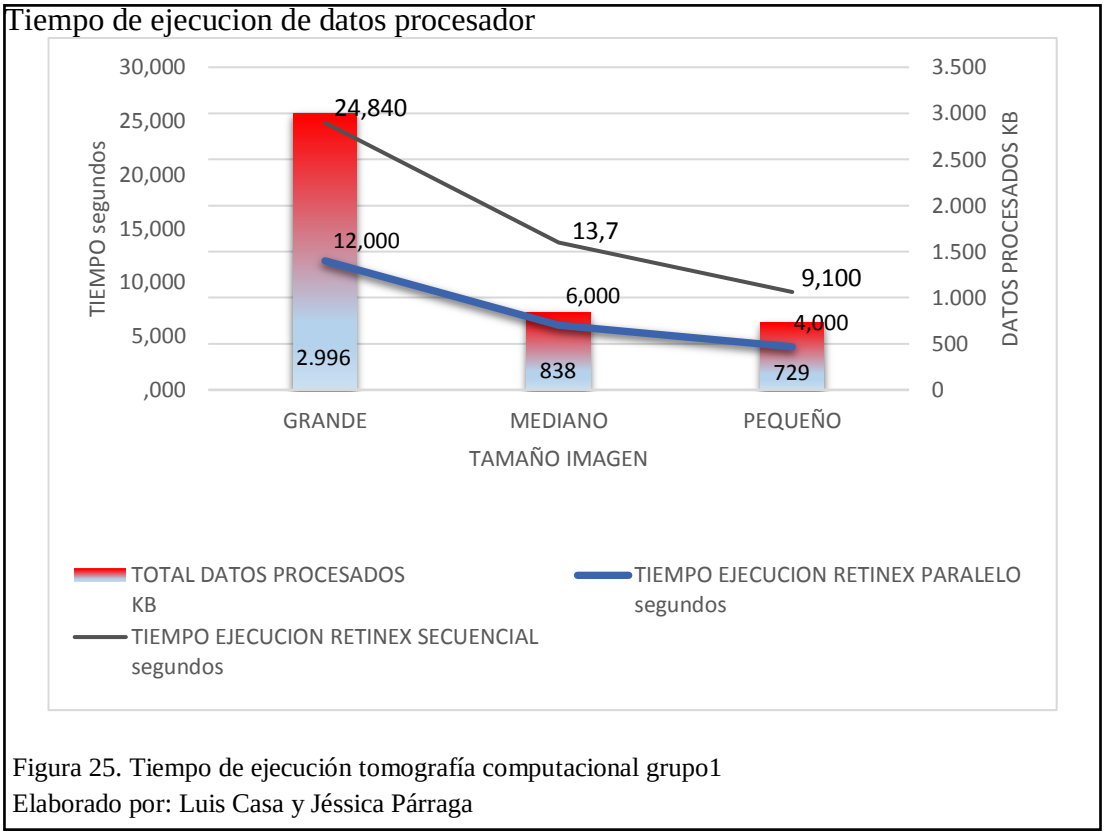


Figura 25. Tiempo de ejecución tomografía computacional grupo1
Elaborado por: Luis Casa y Jéssica Párraga

En la Figura 25, tenemos la gráfica del tiempo de ejecución de nuestro experimento con el algoritmo Retinex en secuencial CPUs y paralelo GPUs con la DataSet de Volumerge y con los diferentes tamaños que se ejecutó para realizar una mayor carga en los GPUs.

Pruebas DataSet Toutatix:

Procesamiento de imagen con algoritmo Retinex arteria coronaria 1

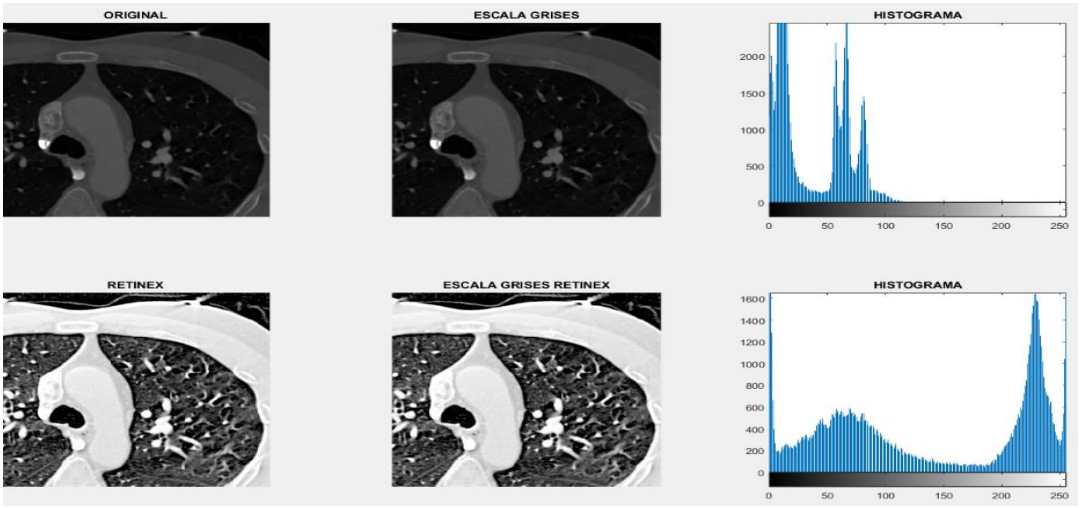


Figura 26. Arteria coronaria con proceso Retinex e histograma (t_img1)
Elaborado por: Luis Casa y Jéssica Párraga

Procesamiento de imagen con algoritmo Retinex arteria coronaria 2

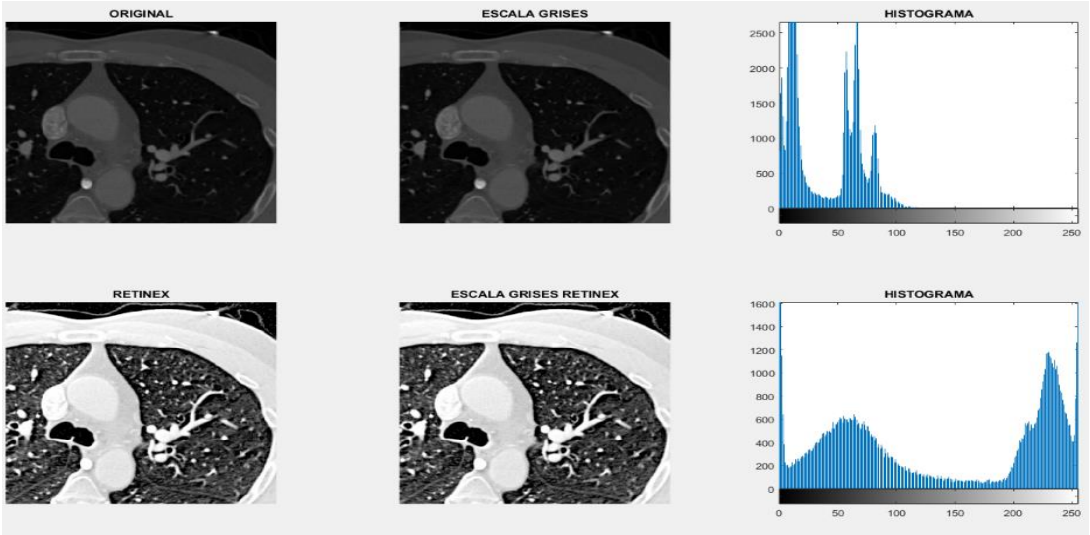


Figura 27. Arteria coronaria con proceso Retinex e histograma (t_img11)
Elaborado por: Luis Casa y Jéssica Párraga

Para la secuencia del DataSet de Toutatix, se realizaron los mismos pasos de pruebas utilizados en Volumerge, y se obtiene la Figura 26 y

Figura 27, de igual manera la sección completa de frames procesados con Retinex se encuentran en el presente trabajo (Ver anexo 5), y por último tenemos la Tabla 6 basada en el mismo formato de la tabla No. 5 cuyos datos sirven para la generación de la Figura 28.

Tabla 6. Datos proceso Retinex secuencia y paralelo en Toutatix

Tamaño	**Nombre imagen**	**Tamaño pixel**	**Total datos procesados kb**	**Tiempo ejecución Retinex secuencial segundos**	**Tiempo ejecución Retinex paralelo segundos**
Grande	t_img1	512 x 512	813	11,53	5
	t_img2	512 x 512			
	t_img3	512 x 512			
	t_img4	512 x 512			
	t_img5	512 x 512			
	t_img6	512 x 512			
	t_img7	512 x 512			
	t_img8	512 x 512			
	t_img9	512 x 512			
	t_img10	512 x 512			
	t_img11	512 x 512			
Mediano	t_img1_mediano	320 x 320	242	7,2	3
	t_img2_mediano	320 x 320			
	t_img3_mediano	320 x 320			
	t_img4_mediano	320 x 320			
	t_img5_mediano	320 x 320			
	t_img6_mediano	320 x 320			

	t_img7_mediano	320 x 320			
	t_img8_mediano	320 x 320			
	t_img9_mediano	320 x 320			
	t_img10_mediano	320 x 320			
	t_img11_mediano	320 x 320			
Pequeño	t_img1_pequeño	180 x 180	110	5,00	2
	t_img2_pequeño	180 x 180			
	t_img3_pequeño	180 x 180			
	t_img4_pequeño	180 x 180			
	t_img5_pequeño	180 x 180			
	t_img6_pequeño	180 x 180			
	t_img7_pequeño	180 x 180			
	t_img8_pequeño	180 x 180			
	t_img9_pequeño	180 x 180			
	t_img10_pequeño	180 x 180			
	t_img11_pequeño	180 x 180			

Nota: Esta tabla contiene los diferentes tamaños de las imágenes procesadas, así como también los tiempos de ejecución en forma paralela y secuencial en toutatix.

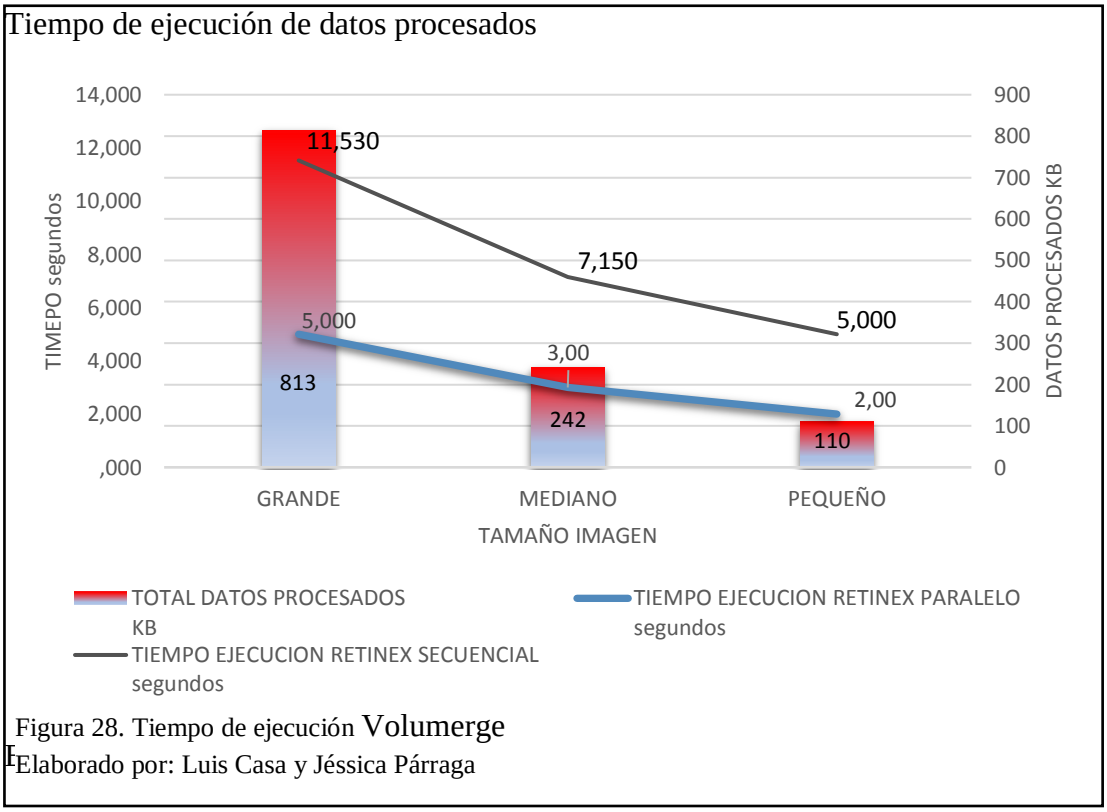


Figura 28. Tiempo de ejecución Volumerge I
Elaborado por: Luis Casa y Jéssica Párraga

Ejecución de nuestro experimento con el algoritmo Retinex en secuencial CPUs y paralelo GPUs con la DataSet de Toutatix y con los diferentes tamaños que se ejecutó para realizar una mayor carga en los GPUs.

CONCLUSIONES

- El algoritmo Retinex utilizado en el presente trabajo fue implementado en conjunto con OpenCL, con el fin de procesar varias imágenes con diferentes ejecuciones en paralelo, las cuales se integran al momento de finalizar cada una de ellas, brindando resultados notorios en cuanto al tiempo de respuesta, como la calidad de la imagen resultante.
- En cuanto al Hardware, el uso de la tarjeta gráfica NVIDIA QUADRO 4000 fue crucial en los resultados obtenidos por el algoritmo Retinex, donde el tiempo de ejecución fue reducido, y también los procesos en paralelo se acoplaron a la GPU de la tarjeta brindando un alto rendimiento en el procesamiento de imágenes.
- Después de realizar las pruebas con el Algoritmo Retinex secuencial, sobre imágenes subacuáticas y tomográficas, el efecto de iluminación mejora considerablemente, lo cual se puede evidenciar mediante los histogramas de los archivos originales y procesados, donde los pixeles se presentan de manera ordenada en los resultados obtenidos.

RECOMENDACIONES

- Cabe mencionar que los recursos hardware deben estar alineados a las aplicaciones informáticas, como es el caso del procesamiento en paralelo, ya que si difiere la ejecución de estos procesos sobre un CPU, por lo cual, se recomienda el uso de un GPU dedicado que se acople al paralelismo que pueden realizan este tipo de programas.
- Es recomendable el uso del algoritmo Retinex en contextos donde se requiera imágenes con gran contraste las cuales brinden información relevante, como es el caso de fotografías tomadas bajo lluvia, neblina o poca luz, las cuales serían ideales para ser procesadas bajo Retinex con el fin de mejorar la calidad de las mismas.
- OpenCL como tal, más allá del ámbito investigativo, puede ser implementado en el mundo laboral donde el manejo de información es indispensable y voluminoso, como es el caso de la banca y el comercio ya que se realizan millones de transacciones en cortos tiempos, y la infraestructura tecnológica se vuelve critica. Es en este punto donde se puede pensar en el procesamiento en paralelo de OpenCL ya que varios procesos se podrían dividir en diferentes hilos de ejecución lo cual mejoraría el tiempo de respuesta y el rendimiento de aplicaciones.

GLOSARIO DE TÉRMINOS

CIELab.- El espacio de color L*a*b*, también referido como CIELAB, es actualmente uno de los espacios de color más populares y uniformes usado para evaluar el color de un objeto (Ideas, 2014).

Kernel.- Proceso ejecutado por múltiples hilos dentro de una GPGPU.

Centos.- Community Enterprise Operating System. Es una bifurcación a nivel binario de la distribución Linux Red Hat Enterprise Linux RHEL.

VMWare.- VM de Virtual Machine.

ESXI.- Es una plataforma de virtualización a nivel de centro de datos producido por VMWare.

NVIDIA.- Tecnología de visualización computacional de NVIDIA, inventora de la GPU. Tarjetas gráficas para juego en el PC.

Clúster.- Un conjunto de computadores interconectados con dispositivos de alta velocidad que actúan en conjunto usando el poder de cómputo de varias CPUs en combinación para resolver ciertos problemas dados.

Estereoscopio.- Consiste en obtener dos fotografías casi idénticas pero que se diferencian ligeramente en el punto de toma de la imagen; estas serán observadas por cada ojo de manera separada y el cerebro las mezclará en una sola imagen creando un efecto tridimensional, el cual corresponde a una serie de fórmulas matemáticas que describen una interpretación virtual de una realidad volumétrica.

Renderización.- Proceso de generar una imagen (imagen en 3D o una animación en 3D) a partir de un modelo, usando una aplicación de computadora.

CRAY1.- Es uno de los supercomputadores más conocidos y exitosos de la historia, y de los más potentes en su época.

Pixel.- Es la unidad básica de una imagen digitalizada en pantalla a base de puntos de color o en escala de grises.

Pixel shader.- Es un programa de sombreado, normalmente ejecutado en la unidad de procesamiento gráfico.

Vertex shader.- Es una herramienta capaz de trabajar con la estructura de vértices de figuras modeladas en 3D, y realizar operaciones matemáticas sobre ella para definir colores, texturas e incidencia de la luz.

Unidades SIMD.- Single Instruction, Multiple Data, en español: "una instrucción, múltiples datos" es una técnica empleada para conseguir paralelismo a nivel de datos.

Estequiometría.- Es la ciencia que mide las proporciones cuantitativas o relaciones de masa de los elementos químicos que están implicados (en una reacción química).

Escáner CT.- Es una máquina compleja que consta de una camilla, en la que se coloca el paciente y un "x", donde está un tubo de rayos X especial, que gira a gran velocidad. Esto permite adquirir información de todo el volumen del paciente.

DER .- Densidad Electrónica Relativa.

T –snakes.- Pueden ser utilizados para segmentar algunas de las estructuras biológicas más complejas en forma de imágenes médicas de una manera eficiente y altamente automatizados.

T-surfaces.- Es un método para representar las superficies isovalor, que se definen en una cuadrícula regular de 3D, por un conjunto de facetas triangulares.

OpenCV.- Es una biblioteca open source para C/C++ para procesamiento de imágenes y visión computarizada, desarrollada inicialmente por Intel. Su primera versión estable fue liberada en 2006.

Retinografía.- La retinografía es una técnica que se utiliza en medicina para obtener fotos en color de la retina.

LISTA DE REFERENCIAS

Aguilar, J., & Leiss, E. (2005). *Introducción a la Computación Paralela.* Texas, USA: Graficas Quinteto. Recuperado el Mayo de 2015, de Introducción a la Computación Paralela: http://www.ing.ula.ve/~aguilar/publicaciones/objetos/libros/ICP.pdf

Añazco, J. (2016). *Implementación de la herramienta web GLPI en un servidor LINUX, virtualizado sobre la plataforma VMware vSphere Hypervisor (ESXi) para el registro de incidentes y control de cambios basado en la norma ISO 20000 para, el Servicio Nacional de Contratación P.*

Arribas, A. (22 de 02 de 2011). *APRENDE TIC*. Recuperado el 23 de 11 de 2016, de https://sites.google.com/site/ticvalcarcel/optimizacion-de-imagenes-para-internet/tipos-de-imagenes-y-formatos

Bangor. (2016). *La visualización y medicina Grupo de Gráficos (VMG)*. Recuperado el 30 de 10 de 2016, de University Bangor: http://vmg.cs.bangor.ac.uk/

Barre, S. (2003). *Imagenes Medicas*. Obtenido de http://www.barre.nom.fr/medical/samples/

Benoit, R. (1999). *Ecole Polytechnique Fédérale, Lausanne, Switzerland.* Obtenido de Computer-Aided Synthesis of Parallel Image Processing Applications: http://infoscience.epfl.ch/record/99849/files/csopipa.pdf

Botella, R. (2002). *Tratamiento de imágenes 3D a partir de la visión estéreo.* Universidad de Alicante Escuela Politécnica Superior.

Briceño, J. (2014). *Arquitecturadel computador.* Obtenido de http://archtectcomp.blogspot.com/2014/10/taxonomia-de-flynn.html

Calvo, D. (2012). *Implementación, paralelización y evaluación de una aplicación de tomografía 3D basada en C++ e Intel TBB.* Universidad Carlos III de Madrid, Colmenarejo. Recuperado el Mayo de 2015, de http://e-archivo.uc3m.es/bitstream/handle/10016/16333/TFG_Diego_Calvo_Picado.pdf?sequence=1

Cardona, A. (2006). *SEGMENTACIÓN DE IMÁGENES DIGITALES 3D BASADO EN REGIONES Y CONTORNOS ACTIVOS PARA LA GENERACIÓN DE MALLAS DE SUPERFICIE.* Obtenido de http://www.cimec.org.ar/ojs/index.php/mc/article/viewFile/526/500

Casa, J. (2011). *Analisis.* Quito.

Castillo, L. (10 de 2008). *Repositorio Académico de la Universidad de Chile.* (U. D. CHILE, Ed.) Obtenido de http://repositorio.uchile.cl/tesis/uchile/2008/castillo_lf/sources/castillo_lf.pdf

Cepeda, J. (2012). *SISTEMA EXPERTO PARA LA SELECCIÓN DE ALGORITMOS DE CONSTANCIA DE COLOR.* Obtenido de http://www.aacademica.org/jcepedanegrete/8.pdf

Creative. (2015). *Creative*. Obtenido de http://www.creative.com/corporate/about/?_ga=1.64749421.311782853.1435057103

Dehesa, M. (2015). *Implementación del modelo Retinex aplicado al procesamiento de imágenes subacuáticas para mejorar su contenido cromático.* Obtenido de http://www.rcs.cic.ipn.mx/2015_91/Implementacion%20del%20modelo%20Retinex%20aplicado%20al%20procesamiento%20de%20imagenes%20subacuaticas.pdf

Durango, N. (2009). *Implementación y aplicación de algoritmos Retinex al preprocesamiento de imágenes de retinografía color.* Obtenido de http://repository.eia.edu.co/bitstream/11190/476/1/RBI00064.pdf

Espinoza, J. (2010). *tesis, Desarrollo de un sistema de información web para mejorar el proceso de evaluación y presentación de perfiles de proyectos de investigación científica y tecnológica a nivel nacional en FINCyT-PCM.* Lima.

Fajardo, L. (2009). *DESARROLLO DE UN MÉTODO DE FUSIÓN DE REGIONES PARA SEGMENTACIÓN DE IMÁGENES.* Obtenido de http://orff.uc3m.es/bitstream/handle/10016/7361/PFC%20-%20LaraFajardo.pdf?sequence=1&isAllowed=y

Franco, A. (2001). *Programación en el lenguaje Java*. Obtenido de http://www.sc.ehu.es/sbweb/fisica/cursoJava/Intro.htm

Franco, E. (2011). *Analisis digital de imagenes tomograficas sin contraste para la b´usqueda de tumores cerebrales.* Obtenido de https://pdfs.semanticscholar.org/d41f/6dbc097d4c5b276c49f80227fe9d99a48a63.pdf

Gallegos, F. (2012). *Mapa Virtual USFQ 3D: Community Aplicación Nativa.* Quito.

Gaudiani, A. A. (Junio de 2012). *SEDICI Repositorio Institucional de la UNPL.* Obtenido de http://sedici.unlp.edu.ar/handle/10915/22691

Gennart, B., & Hersch, R. (1999). *Ecole Polytechnique Fédérale, Lausanne, Switzerland.* Obtenido de Computer-Aided Synthesis of Parallel Image Processing Applications: http://infoscience.epfl.ch/record/99849/files/csopipa.pdf

Gonzalez. (2004). *Digital Image Processing Using Matlab*. Recuperado el 29 de 10 de 2016, de http://users.nik.uni-obuda.hu/vamossy/GepiLatas2007/Segedlet/digital%20image%20processing%20using%20matlab%20(gonzalez).pdf

Gonzalez, R. (2002). *http://folk.uio.no/*. Recuperado el 30 de 10 de 2016, de http://folk.uio.no/ainard/Folder2/Digital%20Image%20Processing%203rd%20ed.%20-%20R.%20Gonzalez,%20R.%20Woods.pdf

Gordillo, J., & Itehua, L. (2012). *Introducción a la Programación Paralela.* DGTIC - UNAM.

Grupo Barre. (2016). *Imagenes Medicas*. Recuperado el 30 de 10 de 2016, de http://www.barre.nom.fr/medical/samples/

Grupo PAS. (2004). *Estándar y Protocolo de Imágenes Medicas DICOM.* Vasco: Universidad de Deusto. Recuperado el 30 de 10 de 2016, de http://www.sicec.unam.mx/app/webroot/files/archivos_portal/archSISEC254505.pdf

Inaquiza, B. (Febrero de 2013). DESARROLLO DE UNA APLICACIÓN PARA LA INTERACCIÓN ENTRE UN GUANTE ELECTRÓNICO E IMÁGENES EN 3D DEL CUERPO HUMANO Y UN CLÚSTER A TRAVÉS DE LA RED AVANZADA PARA EL PROYECTO SISTEMA DE ENTRENAMIENTO VIRTUAL PARA MEDICINA. (K. M. Gavilanes, Ed.) Quito, Pichincha.

Intel. (02 de 04 de 2014). *Developer Zone*. Obtenido de software.intel.com: https://software.intel.com/en-us/forums/opencl/topic/500992

Jähne, B. (2005). *Digital Image Processing*, 6th revised and extended edition. Recuperado el 30 de 10 de 2016, de http://extras.springer.com/2012/978-3-642-04952-1/Content/dip_E.pdf

Juega, C. (2010). *Universidad Complutense Madrid*. Obtenido de http://eprints.ucm.es/11384/

Kreiner, J. (10 de Julio de 2008). *Generación de mallas de elementos finitos en 3D*. Recuperado el 9 de Julio de 2016, de Generación de mallas de elementos finitos en 3D: http://cms.dm.uba.ar/academico/carreras/licenciatura/tesis/kreiner.pdf

López, F. (2009). *diseño e implementación de un laboratorio de Software y redes mediante el usop de un servidor de terminales.* Riobamba.

Manrique & Borja, M. e. (2011). Desarrollo de un sistema móvil/web de georreferenciación para la difusión de ubicaciones de locales comerciales aplicando geosocialización. *Revista de Investiación de Sistemas e Informática*.

Martín, M. (21 de mayo de 2002). *Técnicas Clásicas de Segmentación de Imagen*. Obtenido de http://docplayer.es/9027148-Tecnicas-clasicas-de-segmentacion-de-imagen.html

Matrox. (2015). *Matrox*. Obtenido de http://www.matrox.com/hr/es/company/

Medvis. (2016). *medvis.org*. Recuperado el 30 de 10 de 2016, de http://medvis.org/research-groups/

Meschino, G. (2002). *Algoritmo de Crecimiento de Regiones con característicos de texturas: una aplicación en biopsias de médula ósea.* Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/22925/Documento_completo.pdf?sequence=1

Minguez, D., & Garcia, E. (2008). *Metodologías para el desarrollo de aplicaciones Web.*

Molina, R. (Junio de 2011). Universidad de Sevilla. Recuperado el Mayo de 2015, de Esqueletización de imágenes 3D: http://master.us.es/masterma1/TfM_pdfs(Julio11)/Raul_Reina_Molina.pdf

Monteagudo, P. (1996). *Revista Cubana de Informátia Médica*. Obtenido de http://www.rcim.sld.cu/revista_3/articulos_html/articulo_pedro.htm

Munshi, A. (s4 de noviembre de 2012). *The OpenCL Specification*. Recuperado el 9 de julio de 2016, de The OpenCL Specification: https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf

Muñoz, E. (2009). *Densidad Electrónica Relativa.* Obtenido de http://ricabib.cab.cnea.gov.ar/115/1/1Mu%C3%B1oz_Arango.pdf

Museum, C. H. (1996). *Computer History Museum*. Recuperado el 29 de 10 de 2016, de http://www.computerhistory.org/revolution/supercomputers/10/7

Nevado, C. M. (2010). *Introduccion a Las Bases de Datos Relacionales.*

NVidia. (2015). *nVIDIA*. Obtenido de nVIDIA: http://www.nvidia.es/object/visual-computing-es.html

Osirix. (2017). *OSIRIX*. Obtenido de OSIRIX: http://www.osirix-viewer.com/resources/dicom-image-library/

Páez, P. (28 de Septiembre de 2012). Resolución No. SCPM-DS-2012-001. Quito, Pichincha, Ecaudor: Superintendencia de Control del Poder de Mercado.

Palomino, N. (Julio - Diciembre de 2009). Técnicas de Segmentación en Procesamiento Digital de Imágenes. *Revista de Ingeniería de Sistemas e Informática vol. 6, N.º 2,*, 11-16. Obtenido de http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/risi/2009_n2/v6n2/a02v6n2.pdf

Pereira, O. (2008). *Reconstruccion tridimensional de modelos anatomicos a partir de imagenes medicas digitales.*

Pérez, O. (2011). *Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP - SCRUM.* Bogota: Facultad de Ingeniería Uniminuto.

Petro, A. B. (2014). *Multiscale Retiex.* Image Processing On Line. Obtenido de http://www.ipol.im/pub/art/2014/107/article.pdf

Picado, D. (2012). *Implementación, paralelización y evaluación de una aplicación de tomografía 3D basada en C++ e Intel TBB.* Universidad Carlos III de Madrid, Colmenarejo. Recuperado el Mayo de 2015, de http://e-archivo.uc3m.es/bitstream/handle/10016/16333/TFG_Diego_Calvo_Picado.pdf?sequence=1

Piccoli, M. (2011). *Computación de alto desempreño en GPU.*

Pixmeo. (2016). *Osirix*. Obtenido de http://www.osirix-viewer.com/resources/dicom-image-library/

Purgathofer, W. (2016). *Grupo de Informática Gráfica El Instituto de Computación Gráfica y Algoritmos*. Recuperado el 30 de 10 de 2016, de https://www.cg.tuwien.ac.at/

Quiroz, J. (11 de Enero de 2012). *Proyectos Doctorados 2012.* Obtenido de http://148.206.49.91/ProyectosDoctorado2012/Anteproyecto_JoseQuiroz.pdf

Rahman, Z., Jobson, D. J., & Woodell, G. A. (2004). Retinex Processing for Automatic Image. *Journal of Electronic Imaging*.

Ramírez, A. (s.f.). *Programación multinúcleo.* INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY.

Reimúndez, C. J. (2009-2010). *eprints.ucm.es.* Obtenido de eprints.ucm.es: http://eprints.ucm.es/11384/1/proyecto_master.pdf

Rodet, T., Frederic, N., & Defrise, M. (9 de Marzo de 2004). *http://www.endoexperience.com/*. Recuperado el 9 de Julio de 2016, de http://www.endoexperience.com/: http://www.endoexperience.com/documents/cbcttheory.PDF

Rodriguez, J. (2011). *OpenCL frente a CUDA para análisis de imágenes hiperespectrales en GPU.* Obtenido de http://www.umbc.edu/rssipl/people/aplaza/PFC_JoseManuelRodriguezAlves.pdf

Rosales, & Silva. (2015). *Implementación del modelo Retinex con corrección de color para mejorar la iluminación y el color de imágenes capturadas en condiciones de contaminación ambiental.* Obtenido de http://www.sepi.esimez.ipn.mx/cnies/memorias/IE-8.pdf

Ruiz, J., & Jair, C. (2014). *API de Google Maps para un mapa de conocimiento.* México.

Ruiz, M. (2002). Captación de Tc-99m versus captación de I-131 por el tiroides. *Revista de Física Médica*. Obtenido de http://revistadefisicamedica.sefm.es/index.php/rfm/article/download/213/203#page=16

Salih, R. (Julio de 2010). *http://vereda.ula.ve/*. Recuperado el 20 de 11 de 2016, de http://vereda.ula.ve/curador/assets/docs/ULA_DOCENCIA_DEPART_TrabajoSeminario3_ProfJuanAstorga_RosaSalih_julio2010.pdf

Scarpino, M. (2012). *OpenCL in action.* Shelter Island: Manning Publications Co.

Schultz , T. (2013). *Grupo Análisis y Visualización de Imágenes Médicas, Universidad de Bonn*. Recuperado el 30 de 10 de 2016, de http://cg.cs.uni-bonn.de/en/visualization-and-medical-image-analysis-group-jun-prof-thomas-schultz/

Sensing Americas. (2014). *Konica Minolta.* Obtenido de http://sensing.konicaminolta.com.mx/2014/09/entendiendo-el-espacio-de-color-cie-lab/

Serrano, C. (2008). *Algoritmo de Segmentacion 3D basado en Crecimiento de Regiones por Tolerancia Adaptativa y Optimizacion de Contraste.* Obtenido de https://www.researchgate.net/profile/Ignacio_Garcia_Fenoll/publication/264890116_Algoritmo_de_Segmentacion_3D_basado_en_Crecimiento_de_Regiones_por_Tolerancia_Adaptativa_y_Optimizacion_de_Contraste/links/546f4a340cf2d67fc031090f/Algoritmo-de-Segmentacion-

Strang, P., Persson, O., & G. (2004). A Simple Mesh Generator in Matlab. En S. Review, *A Simple Mesh Generator in Matlab* (págs. 329 - 245).

Tutillo, F., & Fanny. (2013). *Análisis, diseño e implementación de un sistema para la georreferenciación de la comunidad Salesiana en los cantones: Quito y Cayambe utilizando dispositivos móviles y OpenLayers" para la Universidad Politécnica Salesiana.* Quito.

Vance, A. (28 de Octubre de 2010). China Wrests Supercomputer Title From U.S. *The New York Times*, pág. A1.

Vázquez, J. (2009). *Introducción a gvSIG*. Obtenido de http://cgat.webs.upv.es/bigfiles/gvsig/gvsig_112.htm?t6612.html

VMWare. (2016). *Configuring VMDirectPath I/O pass-through devices on a VMware ESX or VMware ESXi host (1010789).* Obtenido de https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1010789

VMWARE, NSX. (2016). *Virtualizacion.* Obtenido de http://www.vmware.com/products/vsphere-hypervisor.html

Wang, W. (Junio de 2012). *TSINGHUA SCIENCE AND TECHNOLOGY.* Obtenido de Parallelization and Performance Optimization on Face Detection Algorithm with OpenCL: A Case Study.