UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

PROPUESTA DE ALGORITMO DE ROTACIÓN DE IMÁGENES EN TIEMPO REAL BASADO EN GEOMETRÍA VECTORIAL Y PROGRAMACIÓN MULTIHEBRAS

PRESENTADA POR:

ELMER ABRAHAM MONTESINOS VALLEJO

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA

PUNO, PERÚ

2014

UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

PROGRAMA DE MAESTRÍA

MAESTRÍA EN INFORMÁTICA

TESIS

PROPUESTA DE ALGORITMO DE ROTACIÓN DE IMÁGENES EN TIEMPO REAL BASADO EN GEOMETRÍA VECTORIAL Y PROGRAMACIÓN MULTIHEBRAS

PRESENTADA POR:

ELMER ABRAHAM MONTESINOS VALLEJO

PARA OPTAR EL GRADO ACADÉMICO DE:

MAGISTER SCIENTIAE EN INFORMÁTICA

APROBADA POR EL SIGUIENTE JURADO:

PRESIDENTE

.....

M.Sc. Ernesto Nayer Tumi Figueroa

PRIMER MIEMBRO

.....

M.Sc. Reynaldo Sucari León

SEGUNDO MIEMBRO

.....

M.Sc. Paco Wilson Marconi Quispe

ASESOR DE TESIS

.....

M.Sc. Leonid Alemán Gonzales

Puno, 27 de marzo del 2014

ÁREA: Inteligencia artificial
TEMA: Procesamiento digital de maquina

DEDICATORIA

Dedico este trabajo a mi querida familia mis padres Abraham y Norma, mi esposa Nelida y mis queridos hijos Katherine y Diego, quienes han estado a mi lado este tiempo y me han animado y apoyado.

AGRADECIMIENTOS

- A Dios, por todo lo que me ha dado, a mis amados padres, por su apoyo, a mi esposa por su amor y comprensión, y a mis hijos que son mi motor.
- A la Universidad Nacional del Altiplano, a la Maestría en Informática por permitirme lograr un nivel más en mi formación profesional, así como a mi asesor Leonid Alemán por compartir sus conocimientos.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

La presente investigación intitulada “PROPUESTA DE ALGORITMO DE ROTACIÓN DE IMÁGENES EN TIEMPO REAL BASADO EN GEOMETRÍA VECTORIAL Y PROGRAMACIÓN MULTIHEBRAS” presenta un estudio de las principales características que debe seguir un algoritmos que optimice el tiempo de procesamiento para la rotación de una imagen teniendo en cuenta que las imágenes pueden variar en su dimensión matricial, teniendo en el mejor caso una imagen VGA de 640x480 y en el peor caso una de 6000x5000 que presentamos en este investigación. Se presenta los tiempos calculados una vez implementada la interfaz desarrolla en el lenguaje C/C++ con interfaz visual para poder ver la rotación de las imágenes mencionadas, para el desarrollo del programa demostrativo se ha empleado la metodología de desarrollo de la programación extrema, se ha omitido la validación del software porque se quiere validar el algoritmo mas no el programa desarrollado como parte demostrativa de esta investigación. Se concluye que los resultados mostrados han mejorado en un 50% el procesamiento del proceso de rotación de imágenes, al implementar una interfaz con un algoritmo multihebra para el procesamiento de dichos cálculos.

Palabras clave: Imagen, procesamiento, multihebra, rotación, matrices.

ABSTRACT

The present research entitled "**PROPUESTA DE ALGORITMO DE ROTACION DE IMAGENES EN TIEMPO REAL BASADO EN GEOMETRIA VECTORIAL Y PROGRAMACION MULTIHEBRAS**" (PROPOSED ALGORITHM OF ROTATION OF REAL-TIME IMAGES BASED) presents a study of the main characteristics that must be followed by algorithms that optimize the processing time for the rotation of an image taking into account That the images can vary in their matrix dimension, having in the best case a VGA image of 640x480 and in the worst case one of 6000x5000 that we present in this investigation. It presents the calculated times once the interface has been developed in the C / C ++ language with visual interface to be able to see the rotation of the images mentioned, for the development of the demo program has been used the methodology of development of extreme programming, Omitted the validation of the software because we want to validate the algorithm but not the program developed as a demonstrative part of this research. It is concluded that the results shown have improved the image rotation process by 50% by implementing an interface with a multi-threaded algorithm for the processing of such calculations.

Keywords: Image, processing, multithreading, rotation, matrices.

INTRODUCCIÓN

Los Algoritmos son una posible solución planteada, está siempre se pone a discusión, en este caso particular es para solucionar el tiempo de procesamiento en la rotación de imágenes, teniendo en cuenta que no habrá restricciones con respecto a la dimensión de ella y sobre todo el espacio total que ocupe en memoria, se evaluara el mejor caso de procesamiento con una imagen de 50x50 pixels (anchura y altura) que darán 50x50x4 = 10000 bytes que procesar en memoria y uno de los peores casos de una imagen tomada de una cámara digital de 16Mpx que será una imagen de 6000 x 5000 pixeles que darán en cálculo de 6000 x 5000 x 4 = 120'000,000 de bytes en memoria, equivalente a 120MB para procesar en menos de un segundo, obviamente computacionalmente costoso.

Para mejorar el tiempo de procesamiento del peor caso de procesamiento se mejorará el algoritmo con la propuesta de generación de Threads (Hilos, Hebras) que permitirán aprovechar el computador con la generación de una cantidad no finita de hebras de procesamiento, simulando la programación paralela. El procesamiento digital de imágenes es una joven línea de investigación en la computación, comparado con el estudio de compiladores y sistemas operativos que iniciaron en la década de los años 1950, lo que hace que los conceptos y teoremas para la resolución de nuevos problemas aun estén en desarrollo, debido a que antes de pensar en ello, había que desarrollar el hardware y los sistemas operativos gráficos que permitieran hacerlo.

Por otro lado, los algoritmos y las técnicas de optimización que han tenido que desarrollarse para el procesamiento digital de imágenes tienen que ser sofisticados. Una imagen contiene elementos numéricos matriciales definidos

por el ancho y la altura los cuales permiten que nuestros sentidos visuales puedan interpretarlo como una representación pictórica de la realidad. Este elemento matricial es el color, el cual está representado generalmente por el modelo RGB (componentes Rojo, Verde y Azul) que tiene una representación tridimensional en el espacio del color, y como tal es posible aplicar los conceptos de geometría del espacio y vectorial.

CAPÍTULO I

PROBLEMÁTICA DE INVESTIGACIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

El procesamiento digital de imágenes es una posterior línea de investigación en la computación, comparado con el estudio de compiladores y sistemas operativos que iniciaron en la década de los años 1950, lo que hace que los conceptos y teoremas para la resolución de nuevos problemas aun estén en desarrollo, debido a que antes de pensar en ello, había que desarrollar el hardware y los sistemas operativos gráficos que permitieran hacerlo. Por otro lado, los algoritmos y las técnicas de optimización que han tenido que desarrollarse para el procesamiento digital de imágenes tienen que ser sofisticados.

Una imagen contiene elementos numéricos matriciales definidos por el ancho y la altura los cuales permiten que nuestros sentidos visuales puedan interpretarlo como una representación pictórica de la realidad. Este elemento matricial es el color, el cual está representado generalmente por el modelo RGB (componentes Rojo, Verde y Azul) que tiene una representación tridimensional en el espacio del color, y como tal es posible aplicar los conceptos de geómetra

del espacio y en nuestro caso métodos estadísticos de regresión, medias y series de tiempo para optimizar la transición del color y generar la mejor representación gráfica de ellos, se puede generar un histograma y suavizarlo mediante técnicas estadísticas.

Esto nos conduce a la pregunta de investigación: ¿El Desarrollo e Implementación de un Algoritmo de rotación de imágenes basado en geometría vectorial y programación multihebras mejorará el proceso de rotación de imágenes en tiempo real?

1.2 OBJETIVOS

1.2.1 Objetivo General

Analizar e implementar un Algoritmo de rotación de imágenes en tiempo real basado en Geometría Vectorial y programación multihebra.

1.2.2 Objetivos Específicos

- Analizar el costo computacional del proceso de rotación de imágenes.
- Codificar la rotación geométrica en un algoritmo computacional y analizar resultados.
- Implementar un esquema multihebra para la solución de grandes cantidades de datos, imágenes de alta resolución e integrarlo al algoritmo de rotación.

CAPÍTULO II

MARCO TEÓRICO

2.1 ANTECEDENTES

Ventura (2009) conocido el problema, se formuló como objetivo reconocer rubricas digitalizadas con el Sistema Inteligente que Valide las Rubricas Digitalizadas en Trámite Documentario en la Región Puno, usando técnicas clásicas de Procesamiento Digital de Imágenes.

La técnica utilizada para el reconocimiento de firmas digitalizadas, es la reducción de características, convirtiendo la imagen digitalizada a escala de grises, luego aplicar el operador de Sobel, que calcula la intensidad de brillo de cada pixel y por ultimo llevarlo a una matriz de CONVOLUCIÓN para extraer las características de las rubricas a validar.

Carranza (2008) se analiza la efectividad de las Wavelets de Gabor como extractor de características de imágenes médicas, asesorados por el Dr. Cesar Beltrán Castañón, quien tiene experiencias en el área de Bioinformática y el procesamiento digital de imágenes, los conceptos vertidos en esta investigación nos permiten conjeturar en mejores técnicas para el

procesamiento de imágenes lo que ha conllevado a impulsar nuestra investigación de la aplicación de métodos estadísticos en la mejora significativa de los métodos actuales.

2.2 BASE TEÓRICA

2.2.1 Imagen

Es la colección de información pictórica, usualmente representa a una superficie bidimensional, puede tomarse como tal a un vector en el espacio 2D y hasta 3D en una malla de polígonos que trazan la superficie de un rostro, en el caso de una fotografía se debe tomar como una imagen de mapa de bits, es decir una colección matricial de puntos, el cual recibe el nombre de Pixel. Podemos tener una descripción visual y grafico para ello obsérvese la Figura 1.

El termino imagen se refiere a una función bidimensional de la luz y la intensidad, a la que indicamos por f (x, y) da la intensidad (iluminación) de la imagen en este punto puesto que la luz es una forma de energía, f (x, y) debe ser estrictamente mayor que cero y finita, es decir, cumplirá siempre la condición: $0 < f(x, y) < 256$, soportando un total de 256 tonalidades de color por elemento (García, 2008, p.32)

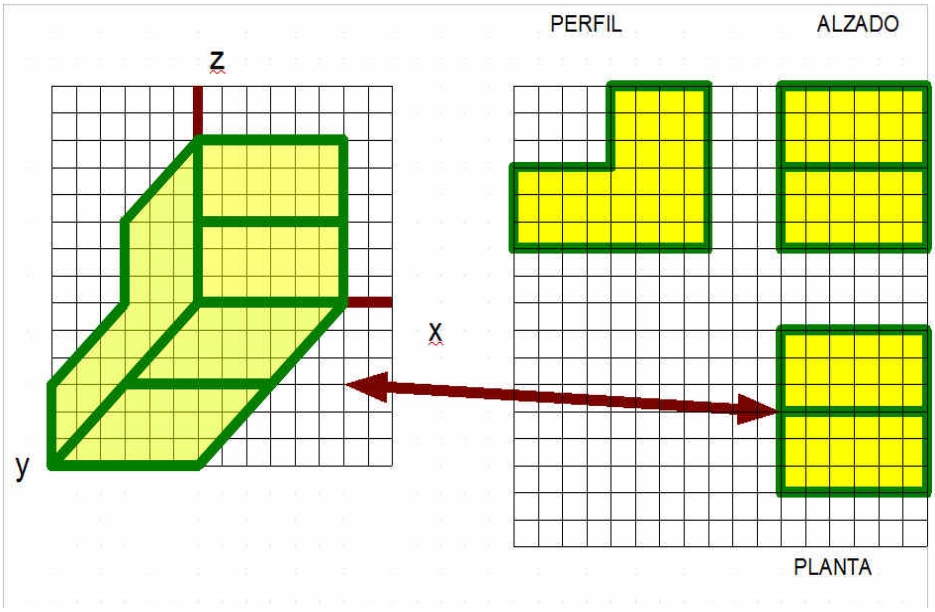

Figura 1. Comparación grafica de una imagen y su ampliación para ver el mapa de bits y pixeles que lo contienen.
Fuente: (García, 2008)

2.2.2 Pixel

Es la unidad básica de una imagen, su nombre proviene de la unión de las palabras Picture Element, debe tenerse en cuenta que el peso de un pixel variara de acuerdo a la profundidad de color de la imagen, esto debido a que los primeros sistemas gráficos como CGA y VGA únicamente podían desplegar imágenes de 16 colores lo que permitía una profundidad de 4 bits por pixel, posteriormente los de 256 colores (8 bits de profundidad) y los modernos Súper VGA de 32 bits (profundidad de 32 bits, 24 bits reales) (García, 2008).

En la actualidad basamos el procesamiento digital de imágenes con imágenes de profundidad de 24 bits por pixel (24bpp) bits per pixel el siguiente grafico muestra la composición general de un pixel y la comparación del vector que genera para los componentes RGB (Red,

Green and Blue) equivalente a las tonalidades de rojo verde y azul. (García, 2008)

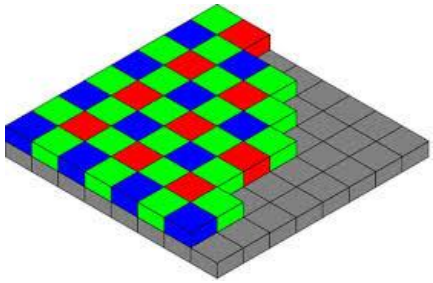

Figura 2. Vista de un conjunto de Pixeles y las componentes RGB posibles.
Fuente: (García, 2008)

Debe tenerse en cuenta que una imagen de 24bpp es un cubo RGB con los 3 componentes por cada pixel en la matriz, es decir subjetivamente tendremos la representación de un cubo en el espacio, cuando se obtiene la escala de grises de un mapa de bits este cubo se reduce a una matriz con una profundidad de 8 bits haciendo ligero el proceso de cálculo para el tratamiento de estas imágenes. (García, 2008)

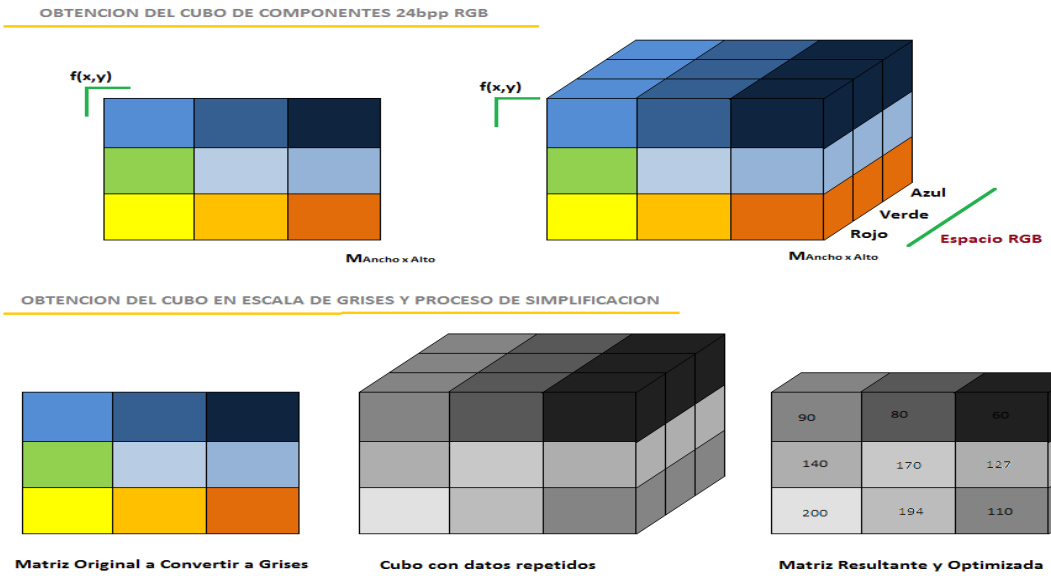

Figura 3: Vista Tridimensional del cubo de componentes RGB de una imagen y representación del proceso de obtención de imágenes en escala de grises.
Fuente: (García, 2008).

2.2.3 Color

La información del Color es el contenido visual más extensamente usado para la recuperación de imágenes. Su valor multidimensional hace la discriminación potencialmente superior frente al hecho de únicamente tomar en cuenta una dimensión simple como puede ser los niveles de gris (grayscale), por ello antes de seleccionar un descriptor apropiado de color es necesario determinar el espacio de color. (Carranza, 2008)

Existe diferentes espacios de color que incluyen el tradicional RGB (red, green, blue), el modelo más simple que mapea directamente las características del dispositivo de salida como HSI (hue, saturation, intensity) que reflejas más precisamente el modelo de colores de percepción humana. El color de un objeto depende no depende del material de la superficie sino varía considerablemente con el cambio de iluminación, la orientación de la superficie y las vistas geométricas de la cámara. Esta variabilidad puede representarse como se aprecia en los gráficos siguientes; sin embargo la invariancia a estos factores no es considerada por ningún método anterior. (Carranza, 2008).

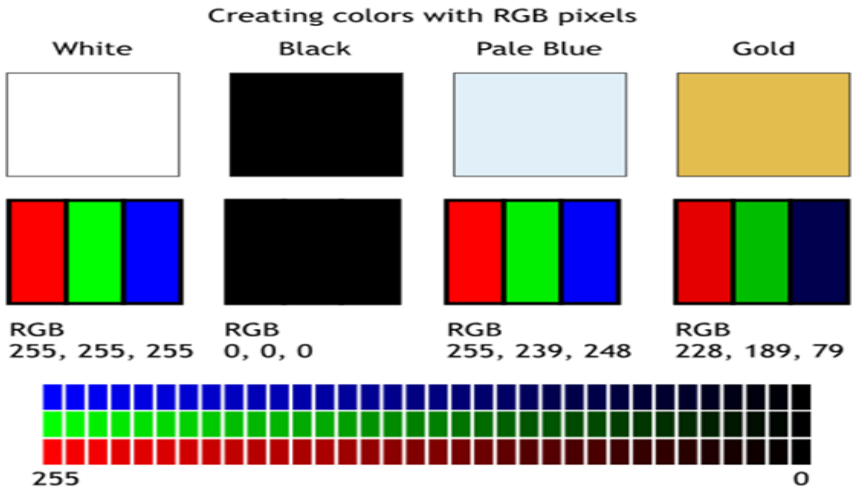

Figura 4. Imágenes presentando distribuciones de distintos tonos de gris
Fuente: (Carranza, 2008).

2.2.4 Modelos De Color

Entre los más importantes podemos citar:

- RGBA
- CMYK
- HSB, HSI

Se refiere a la distribución de un elemento en un espacio definido, la tonalidad de color debe ajustarse a reglas físicas como oscuridad y brillo. (Carranza, 2008)

2.2.5 Textura

Las características de textura determinan que regiones de la imagen poseen un patrón uniforme; relacionadas a la granularidad y la repetitividad de ellos en las superficies dentro de una imagen por ejemplo, pasto, ladrillos, osos de peluche o incluso la textura de la piel humana difieren en textura por la suavidad tanto como sus patrones pueden ser resultado de propiedades físicas de la superficie del objeto o ser resultado de las diferencias de reflexión en la superficie. A pesar de ser fácil para las personas, el reconocimiento de la textura; no acontece de modo similar con procedimientos automáticos; para esta tarea, a veces es necesario técnicas computacionales complejas. (Carranza, 2008)

Básicamente, los métodos de representación por textura pueden ser clasificados en dos categorías: estructurales y estadísticos. El enfoque estructural está dentro de la clasificación y segmentación de imágenes que poseen comportamiento bien definidos; es decir determinístico. En este caso, las imágenes están compuestas por un

arreglo repetitivo de elementos estructurales denominadas textels. (Carranza, 2008)

2.2.6 Rotación geométrica

Una rotación alrededor de cualquier eje en el espacio es algo complicada, pero cuando es alrededor de alguno de los ejes coordenados es mucho más fácil, por ejemplo, para rotar alrededor del eje Z, lo único que hay que hacer es olvidar la coordenada z del punto, y rotar como si la rotación ocurriera en 2 dimensiones. Como se rota alrededor del eje Z, la coordenada Z no cambia. Las ecuaciones de la nueva coordenada del punto rotado se hallan con la ayuda de la figura 5. (García, 2008)

$$P' = \begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Figura 5. Formula matricial para la rotación de puntos geométricos.
Fuente: (García, 2008)

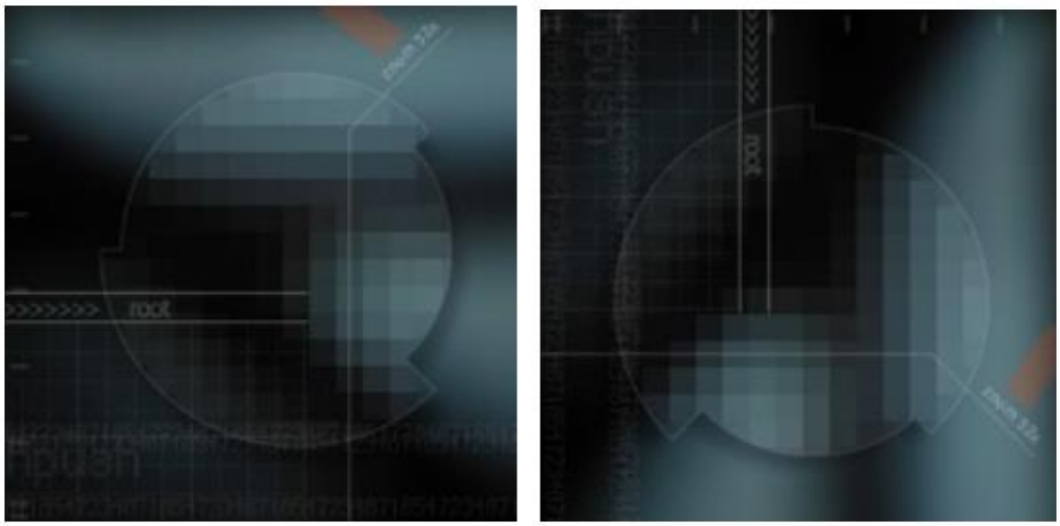

Figura 6. Ejemplo de imagen base y girada 90 grados a la derecha bajo el algoritmo adaptado en C/C++

Fuente: (Carranza, 2008)

2.2.7 Hilos De Ejecución (Threads)

En sistemas operativos, un hilo de ejecución, hebra o subproceso es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

Un hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo con otra tarea. (Stallings, 2011)

Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. El hecho de que los hilos de ejecución de un mismo proceso compartan los recursos hace que cualquiera de estos hilos pueda modificar éstos. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente. (Stallings, 2011)

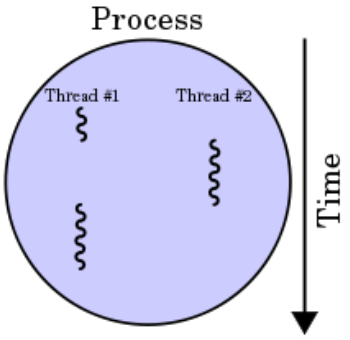

Figura 7. Hilos y procesos de ejecución.
Fuente: (Stallings, 2011)

CAPÍTULO III

METODOLOGÍA

3.1 MÉTODO DE RECOLECCIÓN DE DATOS

Se realiza una investigación de tipo exploratoria - descriptiva, la realización de un estudio de diagnóstico descriptivo que permita indagar sobre el comportamiento de los sistemas existentes y en estudio, lo cual contribuirá a la solución del problema planteado.

3.2 METODOLOGÍA DE DESARROLLO

Son procesos basados en la documentación y el orden para conseguir productos de software de calidad, a lo que los desarrolladores de software que no les gustan del papeleo lo han denominado metodologías burocráticas, por ocupar el mayor tiempo en la documentación y quitando el encanto a la programación. Por los que se eligió la metodología de desarrollo de software Proceso Unificado Ágil.

3.3 MODELO DE REQUERIMIENTOS

Requisitos funcionales.

a. Poner a disposición de los usuarios un programa que rápido de rotación de imágenes.
b. Funciones de visualización de imágenes diversos tamaños
c. Convertir imagen a monocromático.
d. Convertir y grabar imágenes en formato comprimido JPG
e. Diseño de una interfaz de usuario amigable y de fácil modificación

Requisitos no funcionales

Apariencia o Interfaz Externa:

El diseño de la interfaz deberá ser agradable y sobre todo lo más sencillo posible, pues será utilizada tanto por usuarios con una preparación integral como por algunos con conocimientos básicos de computación, por lo que, además, deberá ser sencilla, aprovechando las facilidades del ambiente Web en el que se desarrollará.

Usabilidad:

El sistema podrá ser usado por cualquier tipo de personas que posean conocimientos básicos en el manejo de la computadora y el ambiente de escritorio en sentido general. Rendimiento: Aunque no se requiere una velocidad de respuesta comparada con los sistemas de tiempo real, se debe garantizar la rapidez de respuesta del sistema ante las solicitudes de los usuarios.

Portabilidad:

El programa podrá implantarse para las distintas versiones de Windows, de forma tal que no haya dificultad en cambiar de una a otra plataforma

sin necesidad de efectuar cambios significativos. Lo anterior se debe a que la aplicación está implementada sobre Lenguaje de programación C++.

Requerimientos de Software:

Para la implantación del sistema se requiere de un entorno de desarrollo de programación para el lenguaje C++ Visual para lo que se ha recurrido al entorno de programación C++ Builder. Los requerimientos en el lado del cliente para la utilización del sistema solo se limitan a tener disponible un navegador Web.

Requerimientos de Hardware:

La computadora dedicada al programa debe tener como mínimo las siguientes características de hardware: Procesador Dual Core, o similar, a 3.8 GHz, 2 Gb de memoria RAM y 160 Gb de capacidad en disco duro. Las computadoras que pueden ser utilizadas por los usuarios para acceder al sistema deben tener instalado el programa.

3.4 LENGUAJE DE MODELAMIENTO UNIFICADO

Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como

expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos. UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

3.5 MÉTRICA DE VALIDACIÓN DE SOFTWARE

ISO 9126 es un estándar internacional para la evaluación de la calidad del software. Está reemplazado por el proyecto SQuaRE, ISO 25000:2005, el cual sigue los mismos conceptos. Este estándar es el más usado actualmente en el desarrollo del software. El estándar está dividido en cuatro partes las cuales dirigen, realidad, métricas externas, métricas internas y calidad en las métricas

de uso y expendido. El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características y sub-características de la siguiente manera:

- **Funcionalidad -** Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas. Idoneidad - Atributos del software relacionados con la presencia y aptitud de un conjunto de funciones para tareas especificadas.
- **Exactitud -** Atributos del software relacionados con la disposición de resultados o efectos correctos o acordados.
- **Interoperabilidad -** Atributos del software que se relacionan con su habilidad para la interacción con sistemas especificados
- **Seguridad -** Atributos del software relacionados con su habilidad para prevenir acceso no autorizado ya sea accidental o deliberado, a programas y datos.
- Cumplimiento de normas.
- **Fiabilidad -** Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período establecido.
- **Madurez -** Atributos del software que se relacionan con la frecuencia de falla por fallas en el software.
- **Recuperabilidad -** Atributos del software que se relacionan con la capacidad para restablecer su nivel de desempeño y recuperar los datos

directamente afectos en caso de falla y en el tiempo y esfuerzo relacionado para ello.

- **Tolerancia a fallos -** Atributos del software que se relacionan con su habilidad para mantener un nivel especificado de desempeño en casos de fallas de software o de una infracción a su interfaz especificada.
- **Usabilidad -** Un conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.
- **Aprendizaje cc-** Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.
- **Comprensión -** Atributos del software que se relacionan al esfuerzo de los usuarios para reconocer el concepto lógico y sus aplicaciones.
- **Operatividad -** Atributos del software que se relacionan con el esfuerzo de los usuarios para la operación y control del software.
- Atractividad.
- **Eficiencia -** Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas.
- **Comportamiento en el tiempo -** Atributos del software que se relacionan con los tiempos de respuesta y procesamiento y en las tasas de rendimientos en desempeñar su función.
- Comportamiento de recursos.
- **Mantenibilidad -** Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.

- **Estabilidad -** Atributos del software relacionados con el riesgo de efectos inesperados por modificaciones.
- **Facilidad de análisis -** Atributos del software relacionados con el esfuerzo necesario para el diagnóstico de deficiencias o causas de fallos, o identificaciones de partes a modificar.
- **Facilidad de cambio -** Atributos del software relacionados con el esfuerzo necesario para la modificación, corrección de falla, o cambio de ambiente.
- **Facilidad de pruebas -** Atributos del software relacionados con el esfuerzo necesario para validar el software modificado.
- **Portabilidad -** Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.
- **Capacidad de instalación -** Atributos del software relacionados con el esfuerzo necesario para instalar el software en un ambiente especificado.
- **Capacidad de reemplazamiento -** Atributos del software relacionados con la oportunidad y esfuerzo de usar el software en lugar de otro software especificado en el ambiente de dicho software especificado.
- **Adaptabilidad -** Atributos del software relacionados con la oportunidad para su adaptación a diferentes ambientes especificados sin aplicar otras acciones o medios que los proporcionados para este propósito por el software considerado.
- Co-Existencia.

La sub característica Conformidad no está listada arriba ya que se aplica a todas las características. Ejemplos son conformidad a la legislación referente a usabilidad y fiabilidad.

Cada sub característica (como adaptabilidad) está dividida en atributos. Un atributo es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software.

Un producto software está definido en un sentido amplio como: los ejecutables, código fuente, descripciones de arquitectura, y así. Como resultado, la noción de usuario se amplía tanto a operadores como a programadores, los cuales son usuarios de componentes como son bibliotecas software.

El estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software. Haciendo esto así, sin embargo, se lleva a cada organización la tarea de especificar precisamente su propio modelo. Esto podría ser hecho, por ejemplo, especificando los objetivos para las métricas de calidad las cuales evalúan el grado de presencia de los atributos de calidad.

- Métricas internas son aquellas que no dependen de la ejecución del software (medidas estáticas).
- Métricas externas son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales. Idealmente, la calidad interna no necesariamente implica calidad externa y esta a su vez la calidad en el uso. Este estándar proviene desde el modelo establecido en 1977 por McCall y sus

colegas, los cuales propusieron un modelo para especificar la calidad del software. El modelo de calidad McCall está organizado sobre tres tipos de Características de Calidad:

- Factores (especificar): Describen la visión externa del software, como es visto por los usuarios.
- Criterios (construir): Describen la visión interna del software, como es visto por el desarrollador.
- Métricas (controlar): Se definen y se usan para proveer una escala y método para la medida.

ISO 9126 distingue entre fallo y no conformidad. Un fallo es el incumplimiento de los requisitos previos, mientras que la no conformidad es el incumplimiento de los requisitos especificados. Una distinción similar es la que se establece entre validación y verificación.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1 DESARROLLO DE INTERFAZ DE PRUEBAS

Se ha desarrollado un programa que permita realizar la rotación de imágenes, implementando el algoritmo de rotación basado en matrices, la interfaz se muestra:

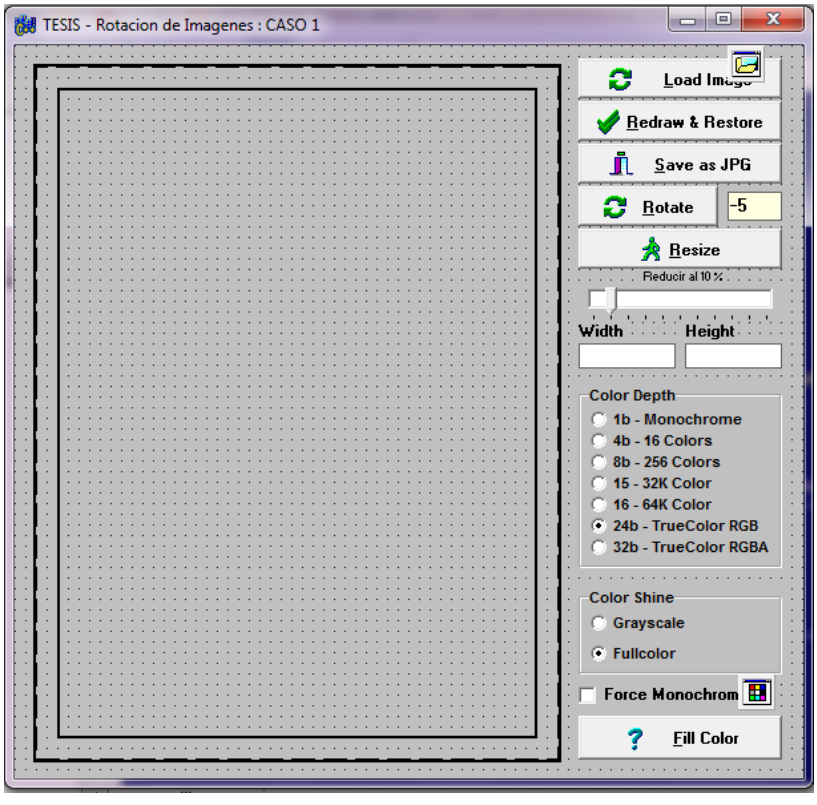

Figura 8. interfaz del ejemplo demostrativo adaptado en C/C++

Seguidamente detallamos la fórmula que se ha usado.

$$P' = \begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Figura 9: Formula matricial para la rotación de puntos geométricos bidimensionales. (García, 2008)

El algoritmo adaptado en lenguaje C++ deberá tener en cuenta que para una mejor optimización de los puntos estos serán simplificados a una ecuación, una vez realizados los cálculos se debe proceder con el retorno de los valores para ello usaremos punteros implícitos en su implementación:

```
01   void TFrrmMain::FncRotacion(
02         float x, float y,
03       float *rx, float *ry,
04         float angulo  )
05                 {
06       #define PI 3.141592654
07
08     angulo = ( PI * angulo ) / 180.0;
09   *rx = (x * cos(angulo)) - (y * sin(angulo));
10   *ry = (x * sin(angulo)) + (y * cos(angulo));
11                 }
12
```

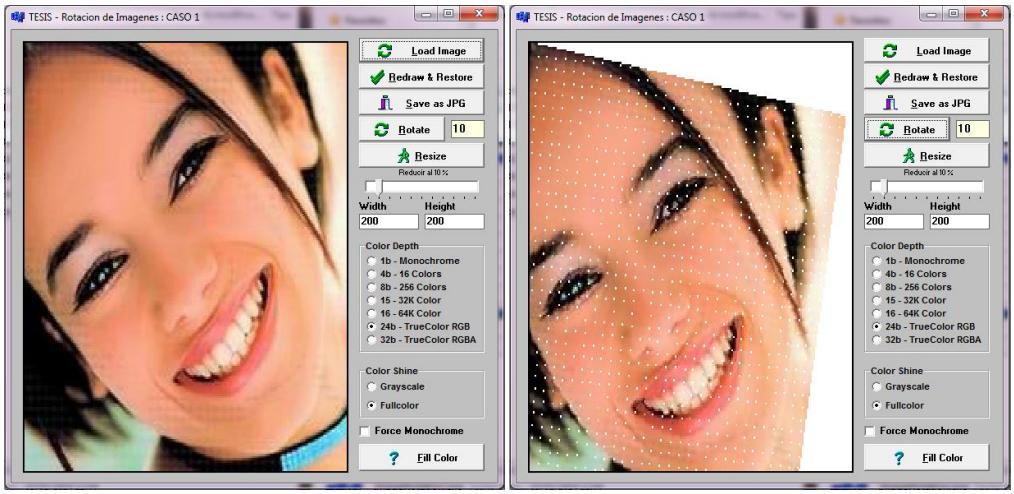

Figura 10. Caso 1 imagen original y seguidamente imagen rotada en 10 grados en sentido horario.

Evaluación de desempeño

Estas pruebas se realizaron en una PC Intel Core 2 Duo de 2,3 GHz con memoria RAM de 2GB sobre Windows 7

Tabla 1. Evaluación de desempeño de rotación de imágenes empleando un algoritmo general

Tamaño de Imagen	Profundidad de Color	Total de Iteraciones	Tiempo de procesamiento
200 x 200	**32 bits RGBA**	**40,000**	**1,200 ms**
800 x 700	**32 bits RGBA**	**560,000**	**12,000 ms**

Notas: en base de pruebas con el software de elaboración de prueba

Como puede observarse el proceso realmente es lento incluso con una imagen pequeña, de 200 x 200 pixeles de ancho y alto, lo que nos lleva a determinar que en el peor caso de una imagen de 5000x6000 a una iteración de

30’000,000 tomaría 10 minutos lo que es aún más decepcionante para los resultados de este algoritmo general.

4.2 PROPUESTA DE MEJORA DEL ALGORITMO

Se ha desarrollado un esquema basado en el concepto de divide y vencerás en el que no se toma el bloque completo para procesar, sino que se separa adecuadamente en bloques pequeños para el procesamiento.

Pero aprovechando el procesamiento en paralelo de las computadoras actuales se hace uso de las Hebras o Threads de procesamiento, convirtiendo así este en un algoritmo multihebra. Visualmente tendríamos la siguiente distribución.

Figura 11: Ejemplo grafico de la separación de bloques para ser procesado por cada hebra pre-programada, debe tenerse en cuenta que serán pedazos aún más pequeños para procesar a la mayor velocidad posible.

```
01  Graphics::TBitmap  *pBmp = new Graphics::TBitmap();
02
03  struct ThreadArgs{
04      int Ancho, Alto,
05      int Inicio;         // desde que parte
06      int CuantasFilas;  // Hasta que punto
07  };
08
09  void __cdecl thRotarDistruido( void *pArgs )
10  {
11      // conversion
12      //
13      ThreadArgs *arg = (ThreadArgs *) pArgs;
14
15      // no serán mas de 20 filas por Hilo, asi la imagen
16  tenga un
17      // ancho de 6000 = 120,000 iteraciones en un Hilo
18  harán 0,45
19      // segundos en paralelo... con 250 hilos
20
21      for( int i= arg->Inicio ; i < arg->CuantasFilas ;
22  i++ )
23      {
24           for( int j=0; j<pBmp->Width; j++ )
25           {
26              float ni;
27              float nj;
28
29              // get angle rotation
30              FncRotacion( j, i, &nj, &ni, Angle );
31              pBmp->Pixels[nj][ni]     =     pBmp->Canvas-
32  >Pixels[j][i];
33           }
34      }
35
36      _endthread();
37  }
38
39
40
41  void     __fastcall    TFrrmMain::BtnRotateClick(TObject
42  *Sender)
43  {
44      float Angle = atof( EdtAngle->Text.c_str() );
45
46      Graphics::TBitmap  *pBmp = new Graphics::TBitmap();
47
48      // si la imagen es pequeña toma 5 pixels de alto
49  sino 20
50      // esto para cad Hilo de ejecucion aseguramos 40
51  Hilos
```

```
52      // como minimo hasta 250 hilos en el peor de los
53  casos
54      // acelerando enormemente el proceso de rotacion.
55      //
56      int alto = ( (pBmp->Height<200)?5 : 20 );
57
58      for( int i=0; i<pBmp->Height/alto; i++ )
59      {
60          // troceamos la imagen y creamos argumentos
61          ThreadArgs *arg = new ThreadArgs;
62
63          arg->Ancho = pBmp->Width;
64          arg->Alto = pBmp->Height;
65          arg->CuantasFilas = alto;

            // se crea un hilo en paralelo y envia un bloque
            // mas no la imagen completa, optimizandolo
            _beginthred(  thRotarDistruido, NULL, arg );
      }
  }
```

A continuación, describimos el algoritmo en lenguaje C++ para ejecutar un Thread o hilo y dejarlo ejecutando en paralelo junto a otras posibles 250 hebras.

Evaluación de desempeño

Estas pruebas se realizaron en una PC Intel Core 2 Duo de 2,3 GHz con memoria RAM de 2GB sobre Windows 7.

Tabla 2. Evaluación de desempeño de rotación de imágenes empleando un algoritmo multihebra

Tamaño de Imagen	Cantidad de Hilos creados	Iteraciones por Hilo	Tiempo de procesamiento
200 x 200	**40**	**8,000**	**100 ms**
800 x 700	**80**	**16,000**	**300 ms**
5000x3500	**250**	**100,000**	**1,800 ms**

Notas: en base de pruebas con el software de elaboración multihebra

Como puede observarse el proceso de rotación bajo esta mejora del algoritmo por nuestro algoritmo multihebra produce mejores resultados incluso en el peor caso al demorar casi 2 segundos se debe a unas imágenes completamente grandes que con el algoritmo anterior hubiera demorado minutos de procesamiento.

4.3 DESARROLLO DE INTERFAZ DE PRUEBAS

Un segundo proyecto con las modificaciones del algoritmo:

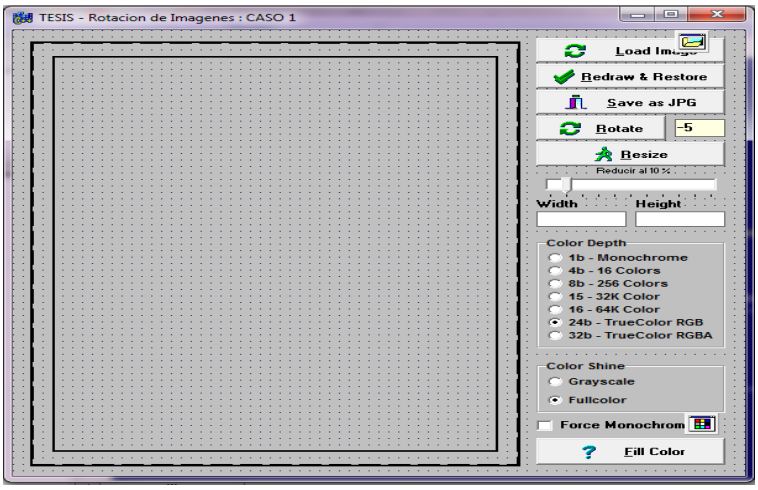

Figura 12. Interfaz ejemplo demostrativo adaptado en C/C++ con multihebra

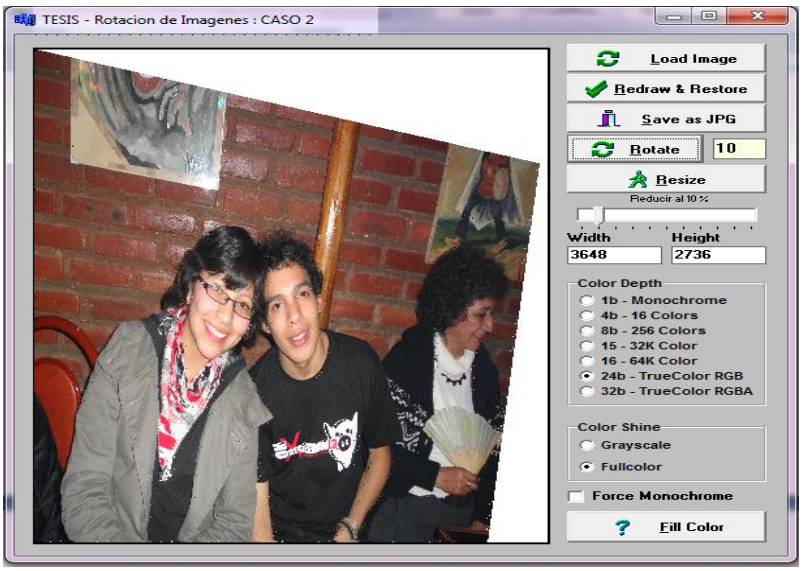

Figura 13. Iinterfaz rotando en tiempo real y con el algoritmo multihebra.

4.4 DEFINICIÓN DE CREACIÓN DE HILOS EN WIN32

```
uintptr_t _beginthread( // Código Nativo
   void( __cdecl *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);
uintptr_t _beginthread( // Código Nativo
   void( __clrcall *start_address )( void * ),
   unsigned stack_size,
   void *arglist
);
uintptr_t _beginthreadex( // Código Nativo
   void *security,
   unsigned stack_size,
   unsigned ( __stdcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);
uintptr_t _beginthreadex( // CÓDIGO DE GESTIÓN
   void *security,
   unsigned stack_size,
   unsigned ( __clrcall *start_address )( void * ),
   void *arglist,
   unsigned initflag,
   unsigned *thrdaddr
);

//Win32 API MSDN
HANDLE WINAPI CreateThread(
  _In_opt_   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_       SIZE_T dwStackSize,
  _In_       LPTHREAD_START_ROUTINE lpStartAddress,
  _In_opt_   LPVOID lpParameter,
  _In_       DWORD dwCreationFlags,
  _Out_opt_  LPDWORD lpThreadId
);
```

CONCLUSIONES

- Para analizar el costo computacional del proceso de rotación de imágenes, se ha implementado el algoritmo de rotación en el lenguaje C++ sobre un entorno de desarrollo Visual para mostrar los resultados, para el cálculo se ha hecho uso de un procesador de 2.3GHz sobre un procesador Intel Core 2 Duo, mostrando que el tiempo se incrementa dramáticamente mientras más grande se vuelve la imagen. Como se muestra en los cuadros mostrados.
- Hemos procedido en codificar la rotación geométrica en un algoritmo computacional en lenguaje C++ ejecutando el programa y apuntando el tiempo que demora en cada imagen, esto para analizar resultados y mostrar comparativamente la mejora con respecto al algoritmo de estudio y la propuesta de nuestro algoritmo multihebra.
- Se ha dividido el procesamiento en una hebra que será ejecutada en paralelo con otras 250 posibles hebras con la posibilidad de ejecutar hasta 1000 hebras por proceso padre, lo que nos reduce enormemente el proceso de cálculo, pues dejamos que el procesador nos ayude en el cálculo de los nuevos datos y posteriormente mostrarlos.

RECOMENDACIONES

- La flexibilidad del algoritmo constituyo el principal objetivo de esta investigación. El diseño de la plataforma y la posterior construcción del entorno de programación y del sistema reflectivo, han sido llevados a cabo tratando en todo momento de obtener un grado máximo de flexibilidad. Sin embargo, esta característica suele estar reñida con la eficiencia en tiempo de ejecución, lo cual puede mejorarse bajo un enfoque más óptimo.
- La programación multihebra solo se usa como tema académico aún no se le da el enfoque de aplicación en la UNA Puno, lo cual debe ser corregido, no todo consta en el desarrollo de Sistemas de Información hay muchas otras áreas de investigación Computacional, como hemos mostrado en esta investigación y mejorando el proceso de cálculo y optimización sobre una tarea habitual.

BIBLIOGRAFÍA

Beck, K. (2005). *Una Explicación de la Programación Extrema: Aceptar el Cambio.* Madrid: Addison-Wesley Iberoamericana España, S.A.

Bill, L., Berg, D. (2003). *Programación Multithread en Java. Madrid: Pearson Educación Prentice Hall.*

Bobak, F. (1993). *Programación Orientada a Objetos.* México DC: McGrawHill Latinoamérica.

Carranza, F. (2008). *Extracción de Características para la Recuperación de Imágenes Médicas por Contenido Utilizando las Wavelets de Gabor. UNT Trujillo: Facultad de Ciencias Físico Matemáticas*.

Chromatic, Apandi, T. (2003). *Extreme Programming Pocket Guide*. NY U.S.: O'Reilly Media.

Cuadros, A. (2005). *Beta-connection: Generating a family of models from planar cross sections.* N.Y.: ACM Transactions On Graphics

García, S. (2008). *Procesamiento Digital de Imágenes. Computers & Typesetting.* (2da ed.). Massachusetts: Editorial Addison-Wesley Reading.

RUP-Paper, R. S. W. (2010). *Rational Unified Process for Systems Engineering RUP SE 1.1.* N.Y.: Rational The Software Development Company.

Stallings, W. (2011). *Operating Systems: Internals and Design Principles*. (7ma Ed.). N.J.: Prentice Hall by PEARSON EDUCATION.

Ventura, R. (2009). *Sistema Inteligente para el reconocimiento de rubricas digitalizadas en trámite documentario.* Puno, Perú.

Vilca, M. (2009). *Estudio del refinamiento de Mallas Geométricas de Triángulos Rectángulos Isósceles.* Master's Thesis. Santiago: Universidad de Chile.

ANEXOS

Anexo 1. Código Fuente

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "SrcMain.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFrrmMain *FrrmMain;
//---------------------------------------------------------------------------
__fastcall TFrrmMain::TFrrmMain(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TFrrmMain::BtnOpenClick(TObject *Sender)
{
    if( dlgPicture->Execute() == True )
    {
        // Carga clasica, pero no editable
        //ImgPicture->Picture->LoadFromFile( dlgPicture->FileName );

        // Hacemos este proceso, para que la JPG transformada en
        // BMP, pueda adaptarse al Strech del control Image, asi
        // poder modificar el contenido grafico y guardarlo

        // es necesario determinar si es BMP o JPEG
        AnsiString strFile = dlgPicture->FileName.LowerCase();

        if( strFile.SubString( strFile.Length()-3, 4 ) == ".bmp" )
        {
            // 1. BMP ordinario, no hay problemas
            ImgPicture->Picture->LoadFromFile( strFile );
        }
        else
        {
            // 2. Cargar como JPG o JPEG y transformar a BMP
            TJPEGImage *pJpg = new TJPEGImage();
            pJpg->LoadFromFile( strFile );

            // GrayScale only in JPG
            if( GrpColorShine->ItemIndex == 0 ) pJpg->Grayscale = True;
            else pJpg->Grayscale = False;

            Graphics::TBitmap  *pBmp = new Graphics::TBitmap();
            //pBmp->PixelFormat = pf24bit; // por defecto
            pBmp->Width = pJpg->Width;
            pBmp->Height = pJpg->Height;
            pBmp->Canvas->Draw( 0, 0, pJpg );
            ImgPicture->Picture->Bitmap = pBmp;

            delete pJpg;
            delete pBmp;
        }

        //
        switch( GrpColorDepth->ItemIndex )
        {
            case 0: ImgPicture->Picture->Bitmap->PixelFormat = pf1bit; break;
// Monochrome
            case 1: ImgPicture->Picture->Bitmap->PixelFormat = pf4bit; break;
// 16    4
```

```
            case 2: ImgPicture->Picture->Bitmap->PixelFormat = pf8bit; break;
// 256   8
            case 3: ImgPicture->Picture->Bitmap->PixelFormat = pf15bit; break;
// 32K  5:5:5
            case 4: ImgPicture->Picture->Bitmap->PixelFormat = pf16bit; break;
// 64K  5:6:5
            case 5: ImgPicture->Picture->Bitmap->PixelFormat = pf24bit; break;
// RGB  8:8:8
            case 6: ImgPicture->Picture->Bitmap->PixelFormat = pf32bit; break;
// RGBA 8:8:8:8
        }

        // Obtiene mas brillo
        ///ImgPicture->Picture->Bitmap->PixelFormat = pf24bit;

        if( ChkDoMono->Checked )
            ImgPicture->Picture->Bitmap->Monochrome = True;

        // Ahora ya son BMP manipulables
        EdtWidth->Text = ImgPicture->Picture->Bitmap->Width;
        EdtHeight->Text = ImgPicture->Picture->Bitmap->Height;
    }
}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::BtnRedrawClick(TObject *Sender)
{
    int arLeft   = 40;
    int arTop    = 50;
    int arRight  = 60;
    int arBottom = 40;

    /*
    int arLeft   = ShpEdge->Left;
    int arTop    = ShpEdge->Top;
    int arRight  = ImgPicture->Picture->Bitmap->Width-ShpEdge->Width;
    int arBottom = ImgPicture->Picture->Bitmap->Height-ShpEdge->Height;
    */

    int arWidth = ImgPicture->Picture->Bitmap->Width;
    int arHeight = ImgPicture->Picture->Bitmap->Height;

    ImgPicture->Canvas->Brush->Color = dlgFillColor->Color;

    // llenar arriba y abajo
    ImgPicture->Canvas->FillRect(   TRect(   0,   0,   arWidth,   arTop)   );
// Top
    ImgPicture->Canvas->FillRect(   TRect(   0,   arHeight-arBottom,   arWidth,
arHeight) );  // Bottom

    // llenar izquierda y derecha
    ImgPicture->Canvas->FillRect(   TRect(   0,   0,   arLeft,   arHeight)   );
// Left
    ImgPicture->Canvas->FillRect( TRect( arWidth-arRight, 0, arWidth, arHeight)
);    // Right
}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::BtnSaveClick(TObject *Sender)
{
    TJPEGImage *pOutJpg;

    try
    {
        pOutJpg = new TJPEGImage();
        pOutJpg->CompressionQuality = 65;
        //pOutJpg->Scale = jsHalf;
        //ImgPicture->Canvas->StretchDraw( );
        pOutJpg->Assign( ImgPicture->Picture->Bitmap );
```

```
        pOutJpg->SaveToFile( "Exported.jpg" );
    }
    __finally
    {
        delete pOutJpg;
        ShowMessage( "Saved OK !" );
    }
    //Image4->Canvas->CopyRect(        TRect(0,0,50,50),        Image1->Canvas,
TRect(0,0,50,50)  );
}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::FormCreate(TObject *Sender)
{
    ;
}
//---------------------------------------------------------------------------
int pX, pY;
void __fastcall TFrrmMain::ShpEdgeMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    pX = X;
    pY = Y;

    Caption = String(X) + " - " + String(Y);
    SetCapture( this->Handle );

}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::ShpEdgeMouseMove(TObject *Sender,
      TShiftState Shift, int X, int Y)
{

    //Caption = String(X) + " - " + String(Y);

    // Drag
    if( Shift.Contains(ssLeft) )
    {
        //Caption = String(X) + " - " + String(Y);
        ShpEdge->Top = Y;
        ShpEdge->Left = X;
    }

}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::ShpEdgeMouseUp(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    ReleaseCapture();
}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::BtnColorClick(TObject *Sender)
{
    dlgFillColor->Execute();
}
//---------------------------------------------------------------------------
void __fastcall TFrrmMain::BtnResizeClick(TObject *Sender)
{
    float prcVal;

    switch( TrkSize->Position )
    {
        case 0:
        case 10: return;
        case 1: prcVal = 0.9; break;
        case 2: prcVal = 0.8; break;
        case 3: prcVal = 0.7; break;
        case 4: prcVal = 0.6; break;
```

```
            case 5: prcVal = 0.5; break;
            case 6: prcVal = 0.4; break;
            case 7: prcVal = 0.3; break;
            case 8: prcVal = 0.2; break;
            case 9: prcVal = 0.1; break;
        }

        int imgWidth = ImgPicture->Picture->Graphic->Width * prcVal;
        int imgHeight = ImgPicture->Picture->Graphic->Height * prcVal;

        ImgPicture->Canvas->StretchDraw( TRect(0,0,imgWidth,imgHeight), ImgPicture-
>Picture->Bitmap );
        ImgPicture->Picture->Graphic->Width = imgWidth;
        ImgPicture->Picture->Graphic->Height =  imgHeight;

        //ImgPicture->Picture->Bitmap->Canvas->Pixels

        // Ahora ya son BMP manipulables
        EdtWidth->Text = imgWidth;
        EdtHeight->Text = imgHeight;
}
//---------------------------------------------------------------------------

Graphics::TBitmap  *pBmp = new Graphics::TBitmap();

struct ThreadArgs{
    int Ancho, Alto,
    int Inicio;         // desde que parte
    int CuantasFilas;   // Hasta que punto
};

void __cdecl thRotarDistruido( void *pArgs )
{
    // conversion
    //
    ThreadArgs *arg = (ThreadArgs *) pArgs;

    // no seran mas de 20 filas por Hilo, asi la imagen tenga un
    // ancho de 6000 = 120,000 iteraciones en un Hilo harán 0,45
    // segundos en paralelo... con 250 hilos

    for( int i= arg->Inicio ; i < arg->CuantasFilas ; i++ )
    {
         for( int j=0; j<pBmp->Width; j++ )
         {
            float ni;
            float nj;

            // get angle rotation
            FncRotacion( j, i, &nj, &ni, Angle );
            pBmp->Pixels[nj][ni] = pBmp->Canvas->Pixels[j][i];
         }
    }

    _endthread();
}

void __fastcall TFrrmMain::BtnRotateClick(TObject *Sender)
{
    float Angle = atof( EdtAngle->Text.c_str() );

    Graphics::TBitmap  *pBmp = new Graphics::TBitmap();

    // si la imagen es pequeña toma 5 pixels de alto sino 20
    // esto para cad Hilo de ejecucion aseguramos 40 Hilos
    // como minimo hasta 250 hilos en el peor de los casos
    // acelerando enormemente el proceso de rotacion.
```

```
    //
    int alto = ( (pBmp->Height<200)?5 : 20 );

    for( int i=0; i<pBmp->Height/alto; i++ )
    {
        // troceamos la imagen y creamos argumentos
        ThreadArgs *arg = new ThreadArgs;

        arg->Ancho = pBmp->Width;
        arg->Alto = pBmp->Height;
        arg->CuantasFilas = alto;

        // se crea un hilo en paralelo y envia un bloque
        // mas no la imagen completa, optimizandolo
        _beginthred(  thRotarDistruido , NULL, arg );
    }
}

//---------------------------------------------------------------------------
```