UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE INGENIERÍA CIENCIAS, FÍSICAS Y MATEMÁTICA

CARRERA DE COMPUTACIÓN GRÁFICA

ESCANEO ÓPTICO Y RECONSTRUCCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS

TRABAJO DE GRADUACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN COMPUTACIÓN GRÁFICA

AUTORES:

JOHANA PATRICIA ALTAMIRANO TERÁN

XIMENA JHOANA RODRÍGUEZ VÉLEZ

TUTOR:

FSC. BAYARDO CAMPUZANO

QUITO – ECUADOR

2012

DEDICATORIA

Dedico a mi buen amigo y salvador Jesús. A mi madre y hermanos, si no fuera por ustedes nada de esto seria posible. A mi esposo, sé que este es el comienzo de grandes logros para nuestra familia; tú eres mi gran motivación, gracias por apoyarme en todo. Y por último la dedico a mis lindas sobrinas que se van a llegar mas lejos que nosotros.

Johana Altamirano

A Dios, mis padres, esposo e hija. Porque me comprendieron al haber elegido mi camino. Porque con su enseñanza, amor y confianza, fortalecieron mi vida. Porque siempre existieron palabras de apoyo, que me ayudaron a seguir adelante. Porque con sus esfuerzo y sacrificios, logré el triunfo que hoy les brindo. Con admiración y respeto.

Ximena Rodríguez

AGRADECIMIENTO

Agradezco en primer lugar al creador y dueño de mi vida, nuestro Padre Celestial, gracias por su amor y misericordia. Agradezco a mi madre, quien con mucho sacrificio, amor y dedicación lo entrego todo para verme surgir, gracias por tus consejos, eres muy virtuosa. Agradezco a mis hermanos por el apoyo que día a día me dieron. A mis profesores y director de tesis, por guiarme en esta etapa de aprendizaje.

Johana Altamirano

Agredezco sobre todo a Dios porque me ha dado la fuerza y el valor de seguir adelante en el cumplimiento de mis metas. A mi familia que ha sido un pilar fundamental para la realización del presente trabajo de grado y a mis profesores por todo el apoyo brindado durante todo éste proceso.

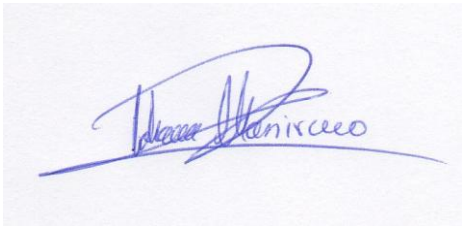
Ximena Rodriguez

AUTORIZACIÓN DE LA AUTORÍA INTELECTUAL

Nosotros, Johana Patricia Altamirano Terán y Ximena Jhoana Rodríguez Vélez en calidad de autores del trabajo de grado realizado sobre ESCANEO ÓPTICO Y RECONSTRUCCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS, por la presente autorizamos a la UNIVERSIDAD CENTRAL DEL ECUADOR, hacer uso de todos los contenidos que me pertenecen o de parte de los que contienen esta obra, con fines estrictamente académicos o de investigación.

Los derechos que como autor me corresponden, con excepción de la presente autorización, seguirán vigentes a mi favor, de conformidad con lo establecido en los artículos 5, 6, 8, 19 y demás pertinentes de la Ley de Propiedad Intelectual y su Reglamento.

Quito, a 7 de Diciembre del 2012



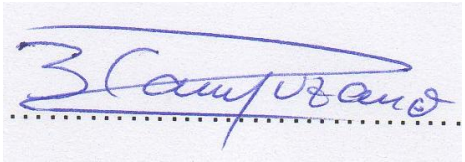
FIRMA	FIRMA
Johana Patricia Altamirano Terán	Ximena Jhoana Rodríguez Vélez
C.I. 1718418997	C.I. 1719232546

CERTIFICACIÓN

En calidad de Tutor del proyecto de Investigación: ESCANEO ÓPTICO Y RECONSTRUCCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS, presentado y desarrollado por los señores: Altamirano Terán Johana Patricia y Rodríguez Vélez Ximena Jhoana, previo a la obtención del Título de Ingeniero en Computación Gráfica considero que el proyecto reúne los requisitos necesarios.

En la ciudad de Quito, a los 7 días del mes de Diciembre del año 2012

FIRMA:



Fsc. Bayardo Campuzano

TUTOR

UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA, CIENCIAS FÍSICAS Y MATEMÁTICA
CARRERA DE INGENIERÍA EN COMPUTACIÓN GRÁFICA

Oficio Nº 647 -DC- IINF
Quito, D.M., 5 de diciembre de 2012

Señores
Ing. Guillermo ALBUJA
Dr. Hernán BENALCAZAR
Presente.

Señores Profesores:

A fin de dar cumplimiento a lo dispuesto en el" Reglamento para la Obtención de los Títulos Profesionales en la Facultad de Ingeniería, Ciencias Físicas y Matemática", aprobado por el H. Consejo Universitario, en sesión del 31 de octubre de 2011; agradeceré a usted, **calificar el Trabajo de Graduación** titulado: **"ESCANEO OPTICO Y RECONSTRUCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS** De las estudiantes Ximena Jhoana RODRIGUEZ VELEZ y Johana Patricia ALTAMIRANO TERAN requisito previo a la obtención del título de **INGENIERA EN COMPUTACIÓN GRÁFICA** en base al **Formulario del Resultado del Trabajo de Graduación**, que me permito remitirle.

Este formulario, deberá enviarse a la Secretaría General de la Facultad en un plazo no mayor a **ocho días.**

Atentamente,

Mat. Eduardo Dávila C.
DIRECTOR DE CARRERA
ING. EN COMPUTACIÓN GRÁFICA

Recibí conforme

Ing. Guillermo Albuja

Dr. Hernán Benalcázar

ELABORADO POR:	CARGO:	FIRMA:	FECHA:
Nancy Solís A.	Ayudante de Sec 2		05/12/2012

Teléfonos: 2558-833 - 2542-026 ext. 218 - Fax: 2226-039 Correo Electrónico: carrera.cg.fing@uce.edu.ec

CATEGORÍA A ACREDITADA CONEA

UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA, CIENCIAS FÍSICAS Y MATEMÁTICA
CARRERA DE INGENIERÍA EN COMPUTACIÓN GRÁFICA

RESULTADO DEL TRABAJO DE GRADUACION

CARRERA DE: **INGENIERÍA EN COMPUTACION GRAFICA**

Quito, 7 de Diciembre del 2012

Señor (ita **JOHANA PATRICIA ALTAMIRANO TERAN**

TEMA: **"ESCANEO OPTICO Y RECONSTRUCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS**

CALIFICACIÓN:

TRIBUNAL	PROFESOR (A)	NOTA SOBRE VEINTE		FIRMA
		NUMERO	LETRAS	
PROFESOR TITULAR	ING. GUILLERMO ALBUJA	19	Diecinueve	
PROFESOR TITULAR	DR. HERNAN BENALCAZAR	18	Dieciocho	
PROMEDIO		18,50	DIECIOCHO, CINCUENTA	

Dra. Katheryne Carrión Valdivieso
SECRETARIA ABOGADA, (E)

Nancy Solis

Teléfonos: 2558-833 - 2542-026 ext. 218 - Fax: 2226-039 Correo Electrónico: carrera.cg.fing@uce.edu.ec

CATEGORÍA A ACREDITADA CONEA

UNIVERSIDAD CENTRAL DEL ECUADOR
FACULTAD DE INGENIERÍA, CIENCIAS FÍSICAS Y MATEMÁTICA
CARRERA DE INGENIERÍA EN COMPUTACIÓN GRÁFICA

RESULTADO DEL TRABAJO DE GRADUACION

CARRERA DE: **INGENIERÍA EN COMPUTACION GRAFICA**

Quito, 7 de Diciembre del 2012

Señor (ita **XIMENA JHOANA RODRIGUEZ VELEZ**

TEMA: **"ESCANEO OPTICO Y RECONSTRUCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS**

CALIFICACIÓN:

TRIBUNAL	PROFESOR (A)	NOTA SOBRE VEINTE		FIRMA
		NUMERO	LETRAS	
PROFESOR TITULAR	ING. GUILLERMO ALBUJA	19	Diecinueve	
PROFESOR TITULAR	DR. HERNAN BENALCAZAR	18	Dieciocho	
PROMEDIO		18,50	DIECIOCHO, CINCUENTA	

Dra. Katheryne Carrión Valdivieso
SECRETARIA ABOGADA, (E)

Nancy Solis

Teléfonos: 2558-833 - 2542-026 ext. 218 - Fax: 2226-039 Correo Electrónico: carrera.cg.fing@uce.edu.ec

CATEGORÍA A ACREDITADA CONEA

CONTENIDO

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE FOTOS

RESUMEN

ESCANEO ÓPTICO Y RECONSTRUCCIÓN TRIDIMENSIONAL DE ROSTROS HUMANOS

La reconstrucción de objetos en tres dimensiones basándose en imágenes 2D no es un proceso que se haya realizado en nuestro país, es por ello que en éste trabajo de tesis tratamos de explotar las aplicaciones que la Geometría Computacional y la Iluminación Estructurada nos brinda para lograr nuestro objetivo que es la reconstrucción de rostros humanos a partir de dos fotografías. El proceso empieza con la proyección de un patrón de luzproporcionada por el proyectorsobre las facciones de la persona; con la luz sobre la cara se capturan dos fotos tanto izquierda como derecha, luego se realiza el procesamiento digital de lasimágenes quitándoles con este tratamiento el ruido e interferencias con lo que obtenemos datos únicamente del rostro.Los datos de la imagen corresponden a las coordenadas de cada una de las franjas que el patrón proyecta en el semblante de la persona y mediante procesos geométricos y matemáticos pasamos de una nube de puntos (X, Y) a una nube en (X, Y, Z). El mallado de los puntos se ejecutaa través del Algoritmo de Delaunay, el mismo que provee la reconstrucción tridimensional del rostro humano. Finalmente la texturización del modelo tridimensional se basa las propiedades de color de cada uno de los pixeles tomados para la reconstrucción.

DESCRIPTORES

ESCANEO ÓPTICO / RECONSTRUCCIÓN TRIDIMENSIONAL DE ROSTROS / ALGORITMO DE DELAUNAY / GEOMETRÍA COMPUTACIONAL /PROCESAMIENTO DIGITAL DE IMAGENES

ABSTRACT

OPTICAL SCANNINGANDTHREE-DIMENSIONAL RECONSTRUCTIONOF HUMAN FACES

The reconstruction of three-dimensional objects based on 2D images is not a process that has taken place in our country, for that reasons in this project we try to take the advantage that computational geometry and structured illuminationapplications give us to reach our goal which is the reconstruction of human faces from two photographs. The process begins with the provided by the projector over the factions of the person. When the light on the face is captured into two photos, both left and right sides we make digital processing of images by taking away the noise and interference and only retrieving data from the face. The image data corresponds to the coordinates of each of the projected pattern linesover and so through geometric and mathematical processes from a set of points (X, Y) to set (X, Y, Z). The meshing of the points runs through Delaunay algorithm, which provides the same three-dimensional reconstruction of the human face. Finally, the texturing of the three dimensional model is based on color properties of each of the pixels from the reconstruction.

KEYWORDS

OPTICAL SCANNING / THREE-DIMENSIONAL RECONSTRUCTION / DELAUNAY ALGORITHM / COMPUTACIONAL GEOMETRY / DIGITAL IMAGE PROCESSING

CAPÍTULO I

PLAN DE TRABAJO DE GRADO Y ANTECEDENTES

1.1. Introducción

La computación gráfica es la rama de las ciencias de la computación que se encarga del estudio, diseño y trabajo del despliegue de imágenes en la pantalla de un computador a través de las herramientas proporcionadas por la física, la óptica, la geometría, etc.

En la última década, las técnicas de modelaje en computación gráfica han evolucionado significativamente, tanto en la publicidad, el marketing, la industria cinematográfica, captura de imágenes e implementación de video juegos, etc. Para lo cual se han utilizado modelos basados en polígonos, superficies, líneas y puntos, sin embargo no han sido suficiente para representar las características tan complejas de los objetos y fenómenos naturales, ya que los modelos matemáticos utilizados a veces son poco manejables o controlables. Por eso se han desarrollado unas técnicas avanzadas de modelaje, con la finalidad de proveer mecanismos concisos, eficientes, flexibles y controlables para especificar y animar los objetos naturales para presentarlos por computador en tiempo real.

La construcción de modelos 3D desde vistas 2D es un campo de investigación que se basa en tener dos o más diferentes puntos de vista que son conjugados para lograr un modelo 3D. El enfoque de la visión estéreo es la estructura del método de la luz que se utiliza para la obtención de imágenes planas además de la utilización de una cámara y un proyector, ejecutándose en un computador. La generación de puntos de coordenadas 3d del rostro, nos permiten crear el mallado para la reconstrucción del cuerpo es decir la modelación del objeto, para su posterior animación que se realiza íntegramente por computadora mediante la aplicación de transformaciones básicas en los tres ejes

(XYZ), Rotación, Escala o Traslación según lo que se considere más adecuado para lograr los efectos que se proponga además de otros aspectos como son la iluminación, renderización y texturas con la finalidad de mostrar una escena con un alto grado de realismo.

Escanear los rostros humanos para mostrar su forma tridimensional en el computador constituye una de las técnicas de Visión Artificial más usados en la cinematografía y en la creación de videojuegos presentando personajes con rostros idénticos a los reales en un entorno casi verdadero. El costo de este proceso es sumamente alto por lo que empresas líderes en animación son las únicas en establecer esta técnica como medio de realización de sus producciones.

1.2. Plan del trabajo de grado

1.2.1. Antecedentes

La Universidad Central del Ecuador es la única universidad que ha incluido en sus carreras universitarias la Ingeniería en Computación Gráfica con lo que los temas orientados a moldeamiento, diseño y obtención de objetos en 3D no son un tema que ha sido explotado en toda su magnitud.

La Reconstrucción 3D de rostros a partir de un modelo real constituye el punto de partida para la obtención de animaciones que ayuden a tener un alto grado de realismo en los personajes, además de ser un sistema pionero en el mundo de la animación tridimensional a nivel de la Universidad Central , incursionando de esta forma en la digitalización en 3D de objetos usados en estos entornos virtuales, cabe mencionar que se pone en práctica la matemática aplicada a la cual nuestra carrea está orientada.

1.2.2. Planeamiento del problema

Habitualmente, el escaneo tridimensional es un proceso costoso que requiere de hardware especializado, el cual no resulta ni barato ni sencillo de manejar.

Los sistemas de escaneo en tres dimensiones requieren una elevada precisión, para obtener un resultado casi real y poder representar personajes computacionalmente.

Por lo cual son necesarios métodos que tienen una velocidad de adquisición de datos muy baja, ya que se necesita tomar cada posición que se quiera digitalizar y el procesamiento de imágenes requiere de algoritmos complejos y métodos matemáticos que demandan un alto procesamiento para el ordenador.

En consecuencia el tiempo de escaneo, procesamiento y animación resulta relativamente elevado y costoso, con instrumentos difíciles de adquirir e implementar.

1.2.3. Hipótesis del proyecto

La implementación de un escaneo tridimensional de rostros, permitirá establecer un sistema ideal para la obtención de personajes para diversos escenarios como entornos virtuales, videojuegos o interfaces de máquinas humana con una exactitud aceptable en la que se pueda tener un máximo acercamiento a la realidad del modelo escaneado teniendo en cuenta que es un sistema que a nivel de hardware resulta práctico y sencillo de administrar.

1.2.4. Objetivos

1.2.4.1. Objetivo general

Reconstruir rostros humanos en el computador, mediante un escáner óptico tridimensional a partir de la captura de imágenes, a través del uso de iluminación estructurada; la cual consiste en proyectar un patrón de luz sobre esta y capturar imágenes usando hardware fácil de incorporar a un bajo costo.

1.2.4.2. Objetivos específicos

- Utilizar hardware fácil de manejar, mediante el uso de un computador, una cámara y un proyector, para el escaneo tridimensional de rostros.
- Implementar el procesamiento digital de las imágenes, para el tratamiento de fotografías, optimizando el tiempo de proceso y de esta manera obtener las coordenadas dadas por los patrones de escaneo.
- Generar el mallado de la superficie escaneada utilizando métodos y algoritmos numéricos más adecuados que reduzcan el tiempo de reconstrucción de la superficie.
- Implementar el conocimiento obtenido a los programas de visualización más adecuados para su posterior animación y aplicación.
- Lograr con este trabajo el inicio de la industria de computación gráfica aplicada a la animación de películas 3D y los videojuegos.

1.2.5. Justificación e implementación

1.2.5.1. Justificación

La matemática proporciona una infinidad de herramientas para poder realizar todo tipo de análisis orientándonos al conocimiento de parámetros y comportamientos que se pueden observar cuando se aplican diferentes modelos matemáticos con la finalidad de obtener un máximo grado de realismo en la obtención de rostros humanos en 3D, destacando que este sistema es la base para la realización de otros tipos de aplicaciones encaminados a los diferentes ámbitos como educación, medicina, arte, ciencia, entretenimiento, geografía etc. estableciendo así un medio interactivo donde se crea la relación de representaciones y formas gráficas.

El sistema pretende que a través de algoritmos establecidos con soluciones matemáticas se pueda realizar de una manera práctica y eficaz la reconstrucción tridimensional de su rostro con resultados realistas de su figura y en el futuro en un objeto, con el fin de despertar el interés tanto de herramientas de hardware y software para la introducción en el mundo 3D.

La técnica de escaneo tridimensional constituye un proceso con un costo sumamente elevado, pero mediante este trabajo de grado se reduce el valor económico de la aplicación, pues es fácil de implementar ya que con el proyector y la cámara web se puede obtener el resultado esperado sin tener que recurrir a hardware especializado mostrando de esta forma que es un sistema de muy poca inversión económica.

1.2.5.2. Implementación

La implementación de un sistema de visión artificial dinámico, que sea capaz de representar un rostro en 3D teniendo en cuenta los recursos

utilizados para el desarrollo tanto de software como de hardware mencionando lo siguiente:

El hardware utilizado por nuestro sistema consta de dispositivos regulares, dos cámaras, un proyector y un ordenador, los cuales conforman la parte fundamental del sistema; teniendo en cuenta que es a partir de estos que obtenemos toda la información requerida para el proceso y el análisis de las imágenes obtenidas.

La persona a la que se escaneará el rostro únicamente necesita estar en la posición adecuada para que el sistema pueda obtener la información necesaria y devolver los resultados deseados, es decir, obtener el rostro en 3D.

Para concluir se mostrara un aplicación del mismo, se podrá visualizar en una animación el escaneo tridimensional del rostro.

1.2.6. Metodología

El desarrollo del proyecto a realizarse será en base a la metodología **UML** que ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos y funciones del sistema, además de aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas el que se va a utilizar es el diagrama de clases.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de

la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

- Requerimientos en entidades y actuaciones
- La arquitectura conceptual de un dominio
- Soluciones de diseño en una arquitectura
- Componentes de software orientados a objetos

1.2.7. Alcance

El sistema de escaneo de rostro tridimensional tiene como principales funciones las siguientes:

- Instalación del Hardware adecuado, el cual consta de dos cámaras y un proyector conectados a un ordenador para su interacción.
- El escaneo tridimensional se limita únicamente a la reconstrucción del área conformada por el rostro de la persona.
- Captura de imágenes del rostro humano para la obtención de la textura.
- Captura la forma tridimensional del rostro.
- Procesamiento de la imagen para la recolección de los puntos de coordenadas del rostro y su posterior mallado.
- Reconstrucción de la forma del rostro, exportando los datos obtenidos a un programa de animación elegido para la visualización.
- Animación del rostro para presentar el resultado obtenido en una aplicación de entretenimiento.
- El sistema no tiene la funcionalidad de reconocer la persona a quien se le hizo la toma de imágenes.

- El rostro 3d no tiene un patrón de animación establecido; por lo que no se podrá animar simultáneamente en la captura de la imagen. Su configuración será independiente del sistema de escaneo tridimensional.

1.3. Visión artificial o gráficos por computador

Los gráficos por computador es una sub-rama de la inteligencia artificial que consiste en programar un computador para que pueda mostrar e interpretar las imágenes de una escena de acuerdo a sus características.[1]

En los últimos años la forma de implementación de imágenes por medio del computador ha evolucionado significativamente gracias a la Computación Gráfica. Las características de los objetos de una escena se los pudo mostrar mediante modelos basados en líneas, polígonos, superficies y puntos, los cuales al ser transformados con procesos matemáticos dan una idea clara los objetos reales que se interpretan a través del computador con la finalidad de poseer el control total de los mismos, es decir, modificar, alterar, combinar sus propiedades hasta llegar a tener una escena en donde el movimiento y animación de los objetos constituye la base de la interpretación de las imágenes en el computador.

1.3.1. Computación gráfica

La computación gráfica es la rama de las ciencias de la computación que se encarga del estudio, diseño y el trabajo de despliegue de imágenes en la pantalla de un computador a través de las herramientas proporcionadas por la física, la óptica, la térmica, la geometría, etc.[2]

[1] http://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
[2] http://es.wikipedia.org/wiki/Computaci%C3%B3n_gr%C3%A1fica

1.3.2. Gráficos en 2D

Constituyen una serie de modelos bidimensionales tales como: tipografía, dibujos, fotos, cartografía, texto, impresiones, los cuales muestran una representación de un objeto del mundo real.

Los gráficos 2D por computadora se han iniciado en la década de 1950, basándose en dispositivos de gráficos vectoriales. Éstos fueron suplantados en gran parte por dispositivos basados en gráficos raster en las décadas siguientes.[2]

Gráficos vectoriales:

Almacenan datos geométricos con mucha precisión y estilo como coordenadas de puntos, uniones, intersecciones, los mismos que forman líneas, polígonos, trayectorias etc. Además cabe mencionar que también proveen ciertas características que ayudan a la identificación y agrupamiento de estas formas básicas como lo son el color, el ancho, el relleno de las formas. La mayoría de este tipo de gráficos necesitan ser convertidos a imágenes raster o mayormente conocidas como mapa de bits para su correcta interpretación.

Los gráficos de tramas o raster:

Conocidos como Mapa de Bits los cuales se muestran como una matriz bidimensional de pixeles. Cada pixel tiene un valor específico como por ejemplo brillo, transparencia o una combinación de los mismos conocido como intensidad. Una imagen raster tiene un tamaño finito definido por el número de filas y columnas que la matriz posea, dichos tamaños se conocen como la resolución de la imagen medidos en pixeles. La combinación de gráficos vectoriales y raster dan paso a la formación de nuevas imágenes que poseen un formato específico según las necesidades que se quieran suplir. Ejm. (bmp, gif, jpeg, png, etc).[2]

Un formato de archivo gráfico es el modelo que se usa para almacenar la información de una imagen en un archivo. Entre los más utilizados actualmente se muestran los siguientes:

Bitmap (BMP): Creado por Microsoft, es el formato nativo para gráficos bitmap en Windows. No utiliza compresión, por lo tanto almacena la información de la imagen de manera ineficiente pero exacta.

Graphics Interchange Format (GIF): Creado por Unisys, muy popular y adecuado para el almacenamiento de imágenes con pocos colores, como logotipos, títulos o fotos sencillas. Permite representar colores de entre una paleta de 256 como máximo. Uno de estos colores puede ser definido como transparente, y el área pintada con este color revelará lo que hay por debajo de la imagen. Otra ventaja del formato GIF es que puede almacenar varias imágenes en un solo archivo, lo que permite la creación de animaciones en base a cuadros temporizados.

Joint Photographic Experts Group (JPEG): Formato creado por el comité del mismo nombre que permite la compresión de imágenes fotográficas a una gran profundidad de colores (millones de ellos). Esto lo convierte en un formato más adecuado que el GIF para el almacenamiento de fotografías. Como extensión del nombre de archivo se usa, indistintamente, JPG o JPEG.

Portable Network Graphics (PNG): Formato de archivo abierto, relativamente nuevo, diseñado para remplazar al GIF en todo menos en lo que se refiere a animaciones. Ofrece transparencia variable (alphachannels), corrección de gamma (control de brillo entre diferentes plataformas) y un grado ligeramente mayor de compresión que el GIF.[3]

[2]http://es.wikipedia.org/wiki/Computaci%C3%B3n_gr%C3%A1fica

[3]BURGE, Wilhelm, J. BURGE, Marck, Digital Image Processing and Algorithmic, 3ra Edición 564p, Austria/Springer, 2008. Pág. 2,3,9

1.3.3. Gráficos en 3D

Los gráficos en 3d son un método avanzado de almacenamiento de las figuras simples de los gráficos en 2d pues almacena la posición de puntos, líneas y las típicas caras que conforman un polígono en un espacio de tres dimensiones.

Los polígonos tridimensionales forman la base fundamental para la creación de gráficos 3d ya que se almacenan en puntos, los mismos que tienen tres coordenadas X, Y, Z y éstos a su vez forman líneas las mismas componen las caras de los polígonos tridimensionales.

Actualmente los gráficos en 3d no solo muestran una escena basándose en polígonos, sino que también se manipulan características que se pueden asemejar a la realidad como lo son las sombras y texturas, cada una de las cuales posee técnicas para lograr una renderización lo más cercana posible a la realidad. Las técnicas que realizan éste procedimiento son las siguientes:

Texturizado (TEXTURING):

Consiste en montar una imagen en un formato de conjunto de pixeles en las caras de los polígonos, ésta imagen se llama textura.

Sombreado (SHADING):

Consiste en realizar un proceso de simulación en computadora de cómo las caras de los polígonos se comportan cuando son iluminados por una o varias fuentes de luz virtual, éste es un proceso de cálculo exacto que muestra las técnicas de sombreado.

1.3.3.1. Principales transformaciones geométricas

Los procedimientos básicos de los cuales se derivan todos los movimientos y animaciones de una escena se basan en las tres transformaciones principales que son: traslación, rotación y escalamiento.

Traslación: Es una transformación que permite modificar la posición de un objeto a lo largo de una trayectoria partiendo de una posición (X, Y) original para moverlo a una nueva posición (X', Y').El par de distancia de traslación (tx', ty') se llama vector de traslación o vector de cambio.

Rotación: Es una transformación que permite modificar la posición de un objeto a lo largo de la trayectoria de una circunferencia en el plano de XY. Para generar una rotación especificamos un ángulo de rotación y la posición XR' YR' del punto de rotación o punto pivote en torno al cual se gira el objeto.

Escalamiento: Una transformación de escalamiento altera el tamaño de un objeto. Se puede realizar esta operación para polígonos al multiplicar los valores de coordenadas (X, Y) de cada vértice por los factores de escalamiento SX y SY para producir las coordenadas transformadas (X', Y').

1.4. Aplicaciones

Actualmente existen muchas aplicaciones de la Computación Gráfica, especialmente en campos de la ingeniería, la investigación, arte y tratamiento fotográfico.

- Animación cuadro por cuadro o animación basada en la física. (Interfaces Gráficas de Usuario GUI)
- Películas, efectos especiales, juegos, etc.
- Las actividades de ingeniería pueden dividirse en cinco áreas: diseño, análisis, dibujo, fabricación, construcción, simulación, procesamiento y control de calidad (CAD-CAM).

- Visualización de imágenes médicas, entrenamiento y simulación.
- El artista siempre ha podido escoger los medios de expresión adecuados para su talento. Las gráficas por computadora son otro medio del que dispone para ampliar su libertad de expresión.
- El beneficio más grande que a aporta las computadoras y las gráficas que realizan en un ambiente educativo. En cuanto a la capacitación se diseñan sistemas inteligentes especiales.
- Diseño de interiores y ambientes arquitectónicos, realidad virtual y aumentada.
- Sistemas de video y películas animadas o cortometrajes, cuyos modelos se basan en la geometría computacional (Splines y Elementos Finitos).
- Procesamiento de imágenes en la industria del entretenimiento, animación, diseño geométrico asistido por computadora, mano factura asistido por computadora, física de video juegos para efectos especiales.
- Acústica y óptica geométrica.
- Ingeniería en diseño industrial.
- Termodinámica.
- Electromagnetismo.
- Modelos de control de la calidad y protección ambiental.
- Modelos de marketing para su publicidad.
- Modelos en hidrología de aguas superficiales, presas hidráulicas, dinámica de ríos.
- Modelo de sistemas hídricos y cambios climáticos.
- Industria química, alimenticia, farmacéutica.
- Simuladores de reservorios, petrolíferos.
- Animaciones en análisis de riesgos.
- Historietas.

CAPÍTULO II

VISIÓN ESTÉREO

2.1. Visión estereoscópica en la reconstrucción 3D

Estéreo es el término que se utiliza en visión para describir más de una vista de una misma escena, con varias imágenes tomadas desde distintos puntos de vista con lo cual se puede llegar a conseguir una característica tridimensional de la escena.[4]

En el proceso de reconstrucción 3D de un objeto hay que tomar en cuenta una serie de elementos como son: webcam o dispositivo de captura de imágenes, procesamiento digital de los datos de la imagen y el ambiente donde se desarrolla la reconstrucción entre otros. [5]

2.2. Luces e iluminación

Un sistema que tiene como principal función el análisis y tratamiento de las imágenes en un computador, debe basarse y constituirse como un sistema que posee una fuente de luz específica y los programas necesarios para procesar la información que dichas imágenes proporcionan.

Los rayos que son proyectados por los objetos deben ser capturados por un medio electrónico, en este caso las webcams que transforman la imagen capturada en una señal digital, la misma que puede visualizarse por medio de un monitor de video. Una vez obtenida la señal digital se puede dar paso a técnicas especializadas de procesamiento de imágenes y de segmentación de los datos que ellas manifiestan con el propósito de realizar una interpretación y reconocimiento de patrones.

---

[4]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 61

[5]http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol10/vol10issue5Sept2012/10TLA5_15Narvaez.pdf

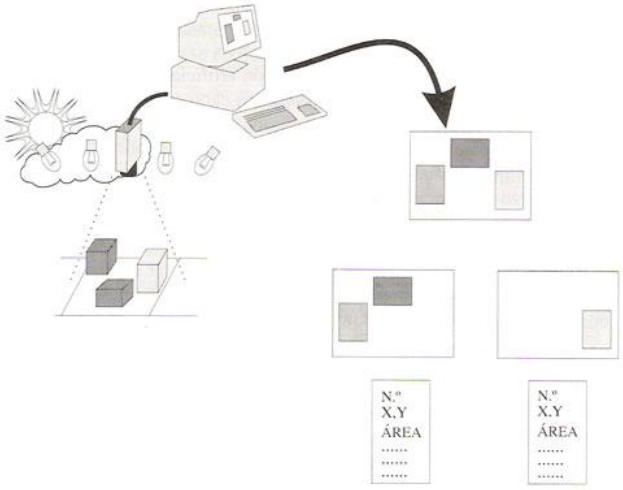


Figura 1: Elementos de un sistema de visión por computador[6]

Luz: "Del latín lux, la luz es el agente físico que permite que los objetos sean visibles. El término también se utiliza para hacer mención a la claridad que irradian los cuerpos, a la corriente eléctrica y la herramienta que sirve para alumbrar."[7]

La luz puede ser percibida por el sentido de la vista y la interacción con la materia permite tener un estudio más amplio de su naturaleza.

La luz que se proyecta en los diferentes objetos ayuda al descubrimiento del tipo de material del cual está hecho el mismo, proporcionando ciertas características como lo son el color, tipo, textura de la materia del que está hecho el objeto.

2.2.1. Iluminación

La iluminación es la parte más importante de un sistema de Visión por Computador, pues reduce el estudio y la interpretación de la realidad capturada. La iluminación natural provee una serie de datos no deseados como son las sombras, brillos etc. Por ésta razón es necesario establecer

[6]ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 11
[7]http://definicion.de/luz/

y reconocer que tipos de luz son adecuados para diferentes tipos de objetos según la escena que se va a estudiar.

2.2.1.1. Características de los objetos

Cuando hay un haz luminoso sobre un material pueden ocurrir tres cosas:

1. Que la luz sea reflejada en su totalidad (espejo) (Propiedad Reflexiva)
2. Que la luz sea absorbida por completo (cuerpo negro) (Propiedad Absorbente)
3. Que la luz pase a través de él (cristal) (Propiedad Transmitiva)[8]

Propiedades reflexivas:

Según se refleje el haz de luz se muestran las siguientes:

- **Materiales especulares:** El ángulo del rayo reflejado es igual al ángulo con el que incide el haz luminoso.
- **Materiales difusos:** Los rayos reflejados los hacen en cualquier dirección. Se usan para definir sombras marcadas.
- **Materiales reflectores:** "El rayo se refleja en la misma dirección pero en sentido opuesto al incidente".[8]
- **Materiales selectivos al espectro:** Dependiendo del haz de luz algunos rayos son absorbidos y otros son reflejados (materiales que tengan cualquier color).
- **Materiales no selectivos al espectro:** Todos los rayos de luz son reflejados.

Propiedades absorbentes:

Si un material absorbe toda la longitud de onda de luz proyectada este será de color negro y en caso de no absorber ninguna el objeto será de color blanco.

[8]ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 12

Propiedades transmitivas:

La luz pasa a través del material dependiendo de esto se tiene:

- **Materiales transparentes:** La luz pasa a través del material y casi no pierde la intensidad de la misma.
- **Materiales translúcidos:** La gran parte de luz atraviesa el objeto pero se expande en todas direcciones.
- **Materiales selectivos al espectro:** Dependiendo de la longitud de onda algunos rayos de luz son transmitidos y otros no.[9]

2.2.1.2. Tipos de iluminación

Los tipos de iluminación son los siguientes:

2.2.1.3. Iluminación direccional

Es aplicar una luz directamente al objeto, se utiliza en el reconocimiento y localización de piezas. La forma de la luz que se va a proyectar depende del objeto que se va a analizar, con lo cual se consigue tener una percepción más clara del objeto según las sombras que éste proyecte según la textura del mismo.

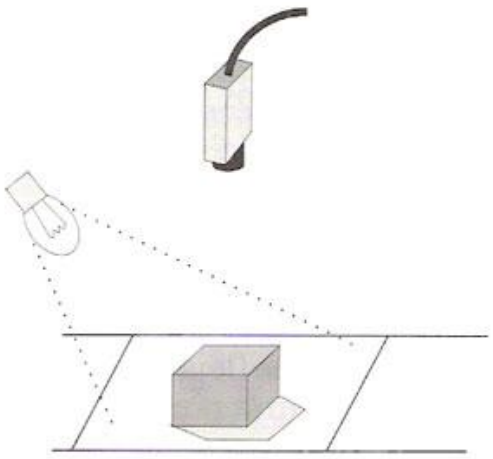


Figura 2: Iluminación Direccional[10]

[9]ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág.13

2.2.1.4. Iluminación difusa

Consiste en tratar que todos los haces de luz incidan en el objeto en todas las direcciones posibles. Su utilización más frecuente es en las imágenes que se obtienen mediante microscopios para el análisis del material.

Las fuentes de luz que proporcionan este tipo de iluminación son:

- "Difusor semiesférico de color blanco mate"
- "Fluorescentes circulares"
- "Anillos de fibra óptica"
- "Anillos de diodos led"

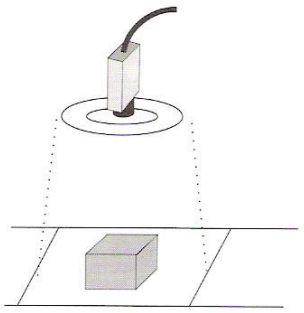


Figura 3: Iluminación Difusa utilizando una fuente circular[11]

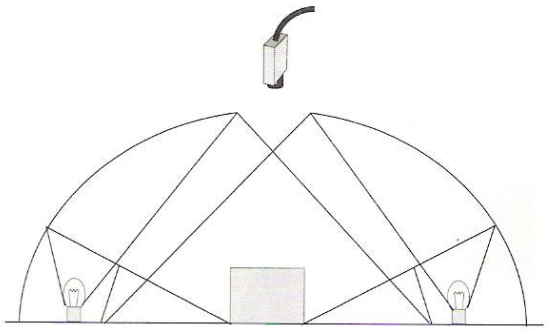


Figura 4: Iluminación Difusa utilizando una fuente semiesférica[11]

2.2.1.5. Iluminación a contra luz

Se define como la iluminación del objeto por detrás haciendo que el objeto, la cámara y la luz queden alineadas. Este tipo de iluminación genera únicamente imágenes en escala de grises lo cual no contribuye con la visualización de los detalles en el objeto.

[10] ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 14, 15

[11] ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 15

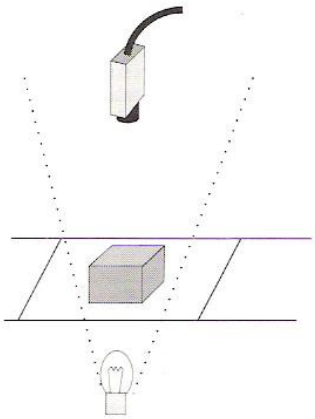


Figura 5: Iluminación a contra luz[12]

2.2.1.6. Iluminación oblicua

Es un caso particular de la iluminación direccional ya que lo más importante en este tipo de iluminación es la generación de sombras que ayudan a especificar la tridimensionalidad del objeto.

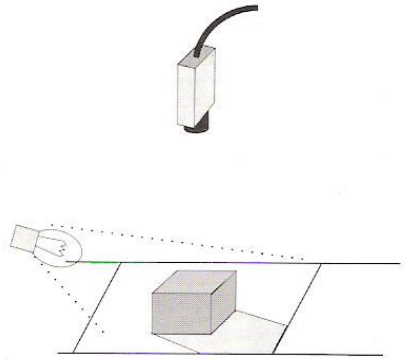


Figura 6: Iluminación oblicua[13]

2.2.1.7. Iluminación estructurada

Consiste en la proyección de un patrón de datos sobre la superficie de trabajo y la distorsión de ésta proyección señala la presencia de un objeto. Esté patrón puede ser puntos, líneas, rejillas, etc. Toda la

[12] ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 16
[13] ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001, Pág. 17, 18

información posible de la distorsión de la imagen proyectada sobre el objeto nos da un conjunto de características tridimensionales del cuerpo que se está estudiando. [Ver Anexo 1]

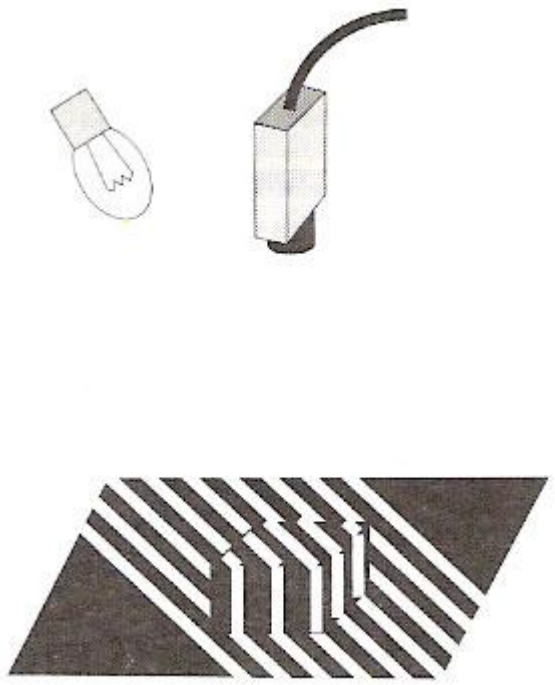


Figura 7: Iluminación estructurada[13]

La iluminación estructurada representa el pilar fundamental para el desarrollo de éste trabajo de grado es por ello, ya que es de ésta iluminación que se va a obtener la forma del cuerpo tridimensionalmente.

El ángulo de inclinación es el que permite establecer y determinar más claramente las elevaciones y rigurosidades que presenta el cuerpo que se va a reconstruir.

En primera instancia del siguiente sistema de visión por computador constituido por la cámara y el rayo de luz se presentan en el escenario sin ningún objeto de estudio (Gráfico1), en el cual se puede apreciar que el rayo incidente de luz no tiene ninguna distorsión que permita determinar que existe un objeto en la escena, pues se proyecta como una sola línea como muestra el Gráfico2.

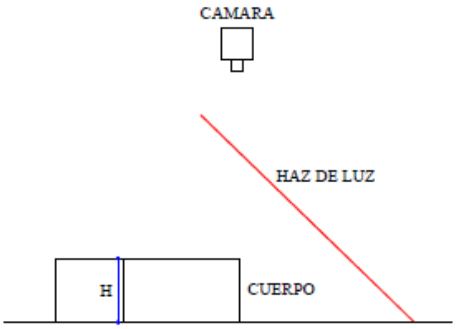


Gráfico 1: Sistema de visión estéreo

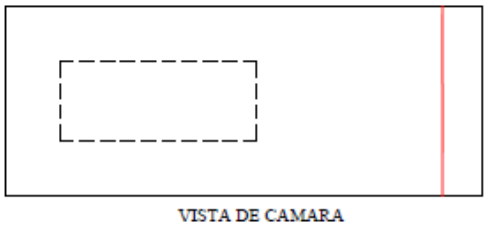


Gráfico 2: Vista de cámara

Con la introducción del cuerpo en la escena la imagen del haz de luz desde el ángulo en que se encuentra la cámara (Gráfico3), se observa que la proyección del mismo sufre una distorsión en el cual se puede apreciar un desplazamiento de la luz del entre la línea que se proyecta sobre la superficie del cuerpo y la superficie de la base del sistema (Gráfico4), teniendo entonces una variable D que se traduce como la profundidad del cuerpo que representa la coordenada en Z del objeto que se está estudiando; es decir ciertas características tridimensionales del objeto.

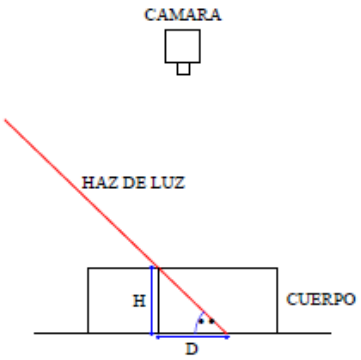


Gráfico 3: Sistema de visión estéreo aplicando el haz de luz sobre un objeto

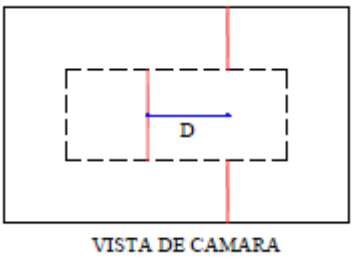


Gráfico 4: Vista de la cámara

Para determinar el ancho y el alto del cuerpo que se muestra en la escena basta con realizar el proceso matemático adecuado que ayude a la obtención de los datos tridimensionales. Para ello mostramos la geometría que existe en la escena.

Partimos de:

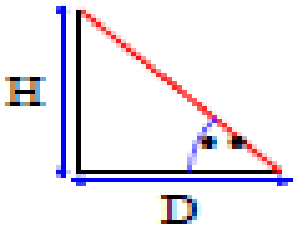


Gráfico 5: Rectángulo en el cual se basa la visión estereoscópica

Donde, el **α** está dato el ángulo de proyección de la luz con respecto al cuerpo, **H** es la altura del objeto y **D** es la distancia de distorsión del haz de luz.

Entonces,

—

Donde,

Para determinar el valor de la altura del cuerpo se necesita introducir el dato de una constante que viene dado por los datos de calibración del cuerpo como lo son la altura **Hc** **(**conocido**)** y **Dc (**dato que puede ser medido**)** siendo entonces,

—

—

2.3. Fuente de luz

"Se entiende por fuente de luz a aquellos cuerpos que la generan, sea por que la producen o la reflejan. Una forma de clasificarlas es dividirlas en naturales o artificiales.Son primarias si producen la luz por medio de procesos internos como el sol y las estrellas, las secundarias producen la luz por reflexión como es el caso de la luna o de cualquier superficie que la reflejen."[14]

Para nuestro sistema de visión por computador utilizamos la luz que emana un proyector ya que es por medio de ésta fuente de luz artificial que se puede conseguir la visualización del patrón establecido sobre el objeto de estudio, en nuestro caso es el rostro humano. [Ver Anexo 2]

2.4. Webcam

Son dispositivos de captura de video de baja resolución, que pueden transmitir imágenes en vivo o una pequeña porción de video. Este es un dispositivo de entrada mediante el cual se puede ingresar información de la realidad al computador para su posterior tratamiento.

Funcionamiento de una webcam: "La luz de la imagen pasa por la lente, esta se refleja en un filtro RGB, el cuál descompone la luz en tres colores básicos: rojo, verde y azul. Esta división de rayos se concentra en un chip sensible a la luz denominado CCD ("ChargedCoupledDevice"), el cuál asigna valores binarios a cada píxel y envía los datos digitales para su codificación en video y posterior almacenamiento ó envío a través de internet por medio de programas de mensajería instantánea".[15]

Para su utilización es suficiente conectar un cable USB desde la webcam al computador, con ello podemos visualizar la imagen en tiempo real que se va a capturar o a transmitir. En general el nombre de webcam se

[14]http://fisica-1.lacoctelera.net/post/2005/11/18/concepto-luz
[15]http://www.informaticamoderna.com/Camara_web.htm

atribuye al software que ésta posee y que permite su funcionamiento en la red.

2.4.1. Especificaciones técnicas de EFACE 2050AF

- **Botón de disparo**

  Puede presionar el botón de disparo para tomar imágenes en CrazyTalk CamSuite PRO tras instalar el Genius Button Manager del CD.

- **Lente con auto enfoque**

  Lente precisa de autoenfoque que mantiene el objeto siempre enfocado.

- **Indicador LED**

  EFACE 2050AF está conectado al puerto USB del ordenador y el LED se iluminará cuando esté utilizando el programa de vídeo.

- **Pivote y base giratoria**

  La base giratoria ajustable se adapta con facilidad a cualquier objeto - panel de notebook, CRT, y monitores LCD.[16]

- **Resolución**
  EFACE 2050AF tiene una resolución de video VGA (640 x 480px).

"La EFACE 2050AF es compatible con USB que cuenta con el sistema de conexión plug&play para Windows 7, Vista, XP SP2 ó Mac 10.4.9 y superior sin requerir un controlador de dispositivo." [16]

Para nuestro sistema de visión computacional incorporamos dos cámaras que ayudan a la captura de las imágenes con una resolución aceptable con las cuales vamos a trabajar para la obtención de los resultados requeridos en éste proyecto. [Ver Anexo 3]

[16] Cd Informativo adjunto en las cámaras a utilizarse

2.5. Análisis bifocal

El análisis bifocal se basa en un sistema constituido por dos cámaras que toman dos imágenes distintas del mismo objeto, éstas imágenes pueden ser capturadas en iguales o distintos tiempos; además hay que considerar que la separación de las cámaras puede ser simétrica o no dependiendo de la profundidad de captura del dispositivo que se esté utilizando.[17]

"La geometría de dos vistas es conocida como la Geometría Epipolar es decir, que las líneas de barrido del par son líneas epipolares. Esta condición se cumple cuando los dos ejes de la cámara de un sistema estereoscópico son paralelos entre sí y perpendiculares a la base."[18]

"Un punto 3D *M* es visto en las dos imágenes como m_1 y m_2 (Ver Figura8 a). Como se estudio en el capitulo anterior, la imagen es definida como la proyección del espacio 3D en un plano de imagen 2D por medio de un centro óptico. Los centros ópticos en este caso son C_1y C_2. A partir de m_1 solamente no se puede saber exactamente la ubicación de *M*, ya que en el proceso de proyección se ha perdido la información de profundidad. Sin embargo, se puede afirmar que *M* debe estar en el rayo que nace del centro óptico C_1 para formar m_1, es decir *M* pertenece a la recta ($m_{1, C1}$). Esta situación es mostrada en la Figura 8 b, donde varios puntos (*M* incluido) pertenecientes a la recta ($m_{1, C1}$) puede ser los que forman el punto m_1 (ver Figura8 c). Se observa que m_2 es uno de estos puntos proyectados, sin embargo a partir de m_1 solamente no se puede saber la ubicación exacta de m_2, solo se puede afirmar que m_2 pertenece a la proyección de la recta ($m_{1, C1}$) realizada por segundo centro óptico C_2en la imagen 2.la proyección de la recta, se denomina *línea epipolar* y se puede apreciar en la Figura8".[19]

[17] MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 61

[18] http://es.scribd.com/doc/36263964/102/Geometria-epipolar

[19] MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 62

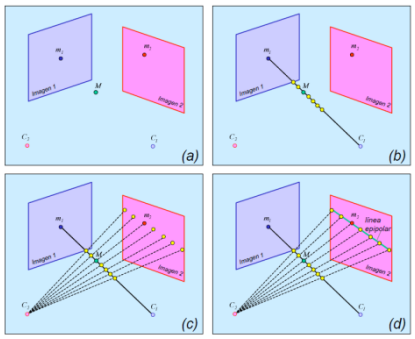


Figura 8: Geometría Epipolar[20]

"Una segunda representación de la Geometría Epipolar se aprecia en la Figura 9, en la que los planos de imagen están entre los centros ópticos y el punto 3D *M*. al igual que en la representación anterior, las representaciones de *M* son m_1 y m_2 en la primera y segunda imagen respectivamente. En esta configuración se observa también al mismo fenómeno: a partir de m_1y C_1los posibles puntos correspondientes a m_2 en la segunda imagen se obtienen entonces mediante la proyección de esta recta por el centro óptico C_2 en la segunda imagen. Esta recta en la imagen 2 es la línea epipolarl_2. "[21]

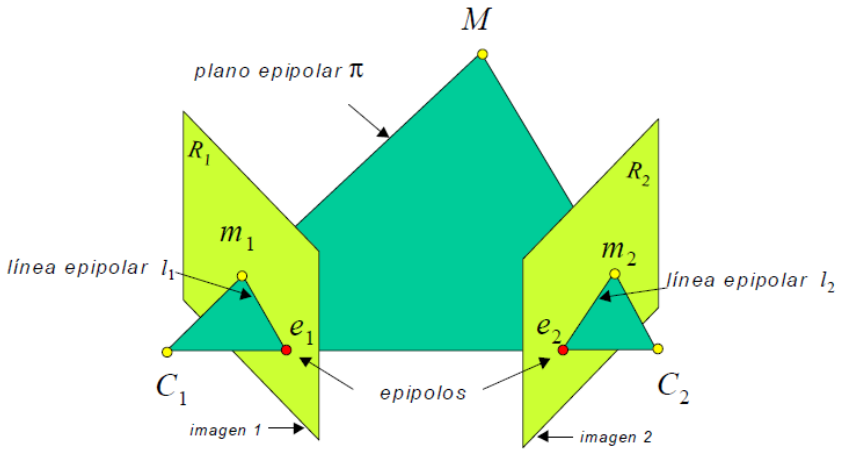


Figura 9: Líneas Epipolares y Epipolos[22]

Entonces con éstos antecedentes se define:

[20]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 62
[21]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág.63
[22]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 64

Plano epipolar:

$$\begin{aligned} l_1 &= \pi \cap R_1 \\ l_2 &= \pi \cap R_2 \end{aligned}$$

Figura 10: Definición Plano Epipolar[22]

Epipolos:

$$\begin{aligned} l_1 \cap l'_1 &= e_1 \\ l_2 \cap l'_2 &= e_2 \end{aligned}$$

Figura 11: Definición de Epipolo[23]

2.5.1. Análisis geométrico de dos vistas

"Para dos vistas se tiene entonces el punto 3D *M* que es visto como m_1 y m_2. Como para cada imagen hay una matriz de proyección se obtiene el siguiente sistema de ecuaciones.

$$\begin{cases} \lambda_1 \mathbf{m}_1 &= \mathbf{AM} \\ \lambda_2 \mathbf{m}_2 &= \mathbf{BM} \end{cases}$$

Figura 12: Sistema de Ecuaciones para Dos Vistas[23]

Donde **A** y **B** son las mismas matrices de proyección de las imágenes 1 y 2 respectivamente y $\mathbf{m_1}$y $\mathbf{m_2}$ son las representaciones homogéneas de

[23]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 65, 66, 67, 68

m_1y m_2. Las coordenadas de **M** en ambas ecuaciones están referidas al mismo sistema de coordenadas. "[22]

"Definido la matriz F de 3 x 3 elementos como:

$$F = [\mathbf{BC_1}] \times \mathbf{BA}^+$$

Se puede expresar la línea epipolar como

$$l_2 = \mathbf{Fm_1}$$

sim_2 pertenece a esta recta entonces $\mathbf{m_2}^T \boldsymbol{l_2} = 0$, o bien

$$\mathbf{m_2}^T \mathbf{Fm_1} = 0 \text{ [1]}$$

La matriz **F** es conocida como la *Matriz Fundamental* y es de gran importancia para el análisis de dos vistas, ya que **F** es constante para una geometría bifocal dada, no depende de m_1, *m2* ni *M*."[23]

Propiedades de la matriz fundamental

"La Matriz Fundamental **F** tiene las siguientes propiedades

i) Las representaciones homogéneas de las líneas epipolares l_1 y l_2 se definen como:

$$l_2 = \mathbf{Fm_1}$$

$$l_1 = \mathbf{F}^T \mathbf{m_2}$$

ii) La restricción epipolar es

$$\mathbf{m_2}^T \mathbf{Fm_1} = 0$$

iii) La matriz **F** es homogénea, ya que *k**F*** para*k* 0 también puede ser utilizada en los cálculos anteriores.

iv) El determinante de **F** es cero , ya que

$$= [e_2] \times \mathbf{BA}^+$$

Como el determinante de **F** es cero, y **F** es homogénea se dice que **F** tiene solo siete (de los nueve) elementos de **F** son

linealmente independientes, los otros dos pueden ser calculados como función de los otros siete.

v) La matriz **F** es constante para una geometría bifocal dada, no depende de m_1, $m_2$ni *M,* solo depende de sus matrices de proyección **A** y **B.**

vi) Los epipolos y la matriz fundamental están relacionadas de la siguiente manera:

$$Fe_1 = 0 \qquad y \qquad F^T e_2 = 0$$

Siendo **0**= $[0\ 0\ 0]^T$. Estas ecuaciones sirven para calcular los epipolos ya que se pueden asumir que como e_1 y e_2 son representaciones homogéneas, su tercera componente es uno. La relación anterior se puede deducir a partir de la condición epipolar: si se tiene un punto m_1 cualquiera en la imagen uno, se sabe que su línea epipolar 2 pasa por el apípalo e_2 esto quiere decir que se cumple e_2^T **$Fm_1 = 0$.**

Como esta condición se cumple para cualquier $m_1$entonces se puede afirmar que e_2^T **Fm_1** = 0.

Como esta condición se cumple siempre para cualquier m_1 entonces se puede afirmar que e_2^T**F**= [0 0 0] o bien **$F^T e_2 = 0$.** El mismo razonamiento se puede hacer para el epipolo e_1 como lo que se obtiene $Fe_1 =$ **0.”**[24]

[24]MERY, domingo, Visión Artificial, 3era Edición, Universidad Santiago de Chile, 2002, Pág. 70

CAPÍTULO III

MANIPULACIÓN DE LAS IMÁGENES

3.1. Procesamiento digital de imágenes

El Procesamiento digital de imágenes consiste en manipular y estudiar los pixeles de una imagen, para lograr tener toda la información posible de los mismos para su posterior interpretación. Los distintos tipos de programas asociados con la modificación de imágenes proveen una serie de complejos algoritmos que ayudan la identificación de los datos que se requiere obtener de las capturas de la realidad o escena en estudio.

Las modificaciones o mejoramientos de una imagen se pueden realizar afectando las siguientes características:

- Modificación de color
- Modificación de pixeles
- Generación de efectos.

Modificación del color: Constituye la modificación del color de cada uno de los pixeles de la imagen sin tener que cambiar la posición de los mismos, es decir se juega con las variaciones de los niveles de ROJO, VERDE Y AZUL que son los tres colores fundamentales que forman el pixel de una imagen.

Modificación de pixeles: Consiste en la manipulación de los pixeles cambiándolos de posición dentro de la misma imagen,

Generación de efectos: Es la parte en la que se juegan con el color y el cambio de posición de los pixeles

3.1.1. Generalidades de una imagen

Imagen es la representación figurativa de una cosa como un sentido amplio de expresión. Es la captura de la realidad que se la puede percibir por medio de los sentidos. Las imágenes son captadas por nuestra vista y permanecen allí, para luego verse plasmadas sobre un lienzo, papel, etc. También pueden ser capturadas por una lente óptica o mediante un espejo, es decir son copias de la realidad inverosímiles ya que una foto y un dibujo no son lo mismo pero captan la característica de una escena.

Existen imágenes físicas visibles e imágenes físicas invisibles que poseen las mismas características que las visibles y que están fuera del rango del ojo humano, tales como imágenes infrarrojas o ultravioletas, las mismas que para ser captadas requieren filtros especiales y obtener una visualización de éstas.

Las imágenes físicas visibles pueden ser permanentes como un póster, documento impreso, fotografía o transitorias como las que se generan a través de un monitor.

Una imagen digital 2D consta de filas y columnas cuya intersección se denomina pixel.

Ejemplo:
Una imagen 2D monocromática (imagen de dos colores), está dada por una función donde denotan las coordenadas del pixel y cuya transformación mediante ***f*** muestra el valor de la intensidad de iluminación de ese punto. En caso de imágenes acromáticas (imagen en escala de grises) la función ***f*** está dada por el nivel de gris.
Una imagen 2D cromáticas, el valor de la intensidad viene dado por funciones simples de dos variables ***fr (x, y), fg(x, y)*** **y** ***fb(x, y),*** que

expresan la intensidad de iluminación de un punto, en el mismo ámbito, para los tres componentes cromáticos primarios: rojo, verde y azul.[25]

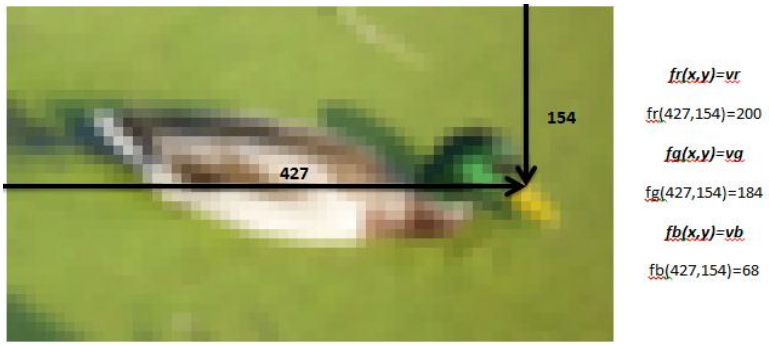


Foto 1: Representación de pixel en imagen cromática

Las imágenes digitales tienen varias características a considerar:

Profundidad del color

"La profundidad de color o bits por pixel (bpp) es un concepto de la computación gráfica que se refiere a la cantidad de bits de información necesarios para representar el color de un píxel en una imagen digital. Debido a la naturaleza del sistema binario de numeración, una profundidad de bits de *n* implica que cada píxel de la imagen puede tener 2^n posibles valores y por lo tanto, representar 2^n colores distintos."[26]

Foto 2: Profundidad del color en bits

[25]http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol10/vol10issue5Sept2012/10TLA5_15Narvaez.pdf
[26]BURGE, Wilhelm, J. BURGE, Marck, Digital Image Processing and Algorithmic, 3ra Edición 564p, Austria/Springer, 2008. Pág. 2

Tamaño del archivo

Corresponde a la cantidad real de información que contiene el archivo y se traduce como la cantidad de espacio que ocupa en el disco, éste tamaño se mide en bits, bytes o múltiplos de éstas unidades de medida.

Resolución

Es el número de pixeles definido a lo ancho y a lo largo de una imagen, es decir la cantidad de filas y columnas que posee la imagen.

3.1.2. Procesamiento puntual

Las operaciones puntuales se basan en el valor de cada pixel, transformando los pixeles de una imagen y cuyos cambios se aplican sobre ellos mismos. Las modificaciones puntuales son del brillo y el contraste que se suma una cierta intensidad al valor de cada pixel. Dentro de este grupo, la operación conocida como umbral permite resaltar puntos determinantes en la configuración de la estructura de un objeto.

Brillo: Aumentar o disminuir el brillo de una imagen consiste en sumar o en restar un valor constante al valor de cada pixel sin que sobrepase los rangos de 0 a 255.[27]

Foto 3: Izq. Imagen Normal; Der. Imagen sin Brillos

Contraste: "Consiste en aumentar o disminuir la pendiente de la línea recta con pendiente a 45 grados que representa los grises, cuidando siempre de nunca rebasar los límites 0 y 255."[27]

[27]http://www2.uacj.mx/Publicaciones/GeneracionImagenes/imagenesCap8.pdf

Foto 4: Izq. Imagen Normal; Der. Imagen aumentada el Contraste

Umbral: Consiste en la separación de los pixeles dependiendo del valor del umbral, obteniendo una imagen en la que los pixeles con intensidad de gris con menos valor que el establecido en el umbral se les asigna una intensidad de cero, mientras que los pixeles que superan la intensidad establecido en el umbral toman una intensidad máxima.

Foto 5: Izq. Imagen Umbral 125; Der. Imagen Umbral 200

Código con punteros para la umbralización. [Ver ANEXO 4]

3.2. Segmentación de imágenes

Foto 6: Proceso de segmentación de imágenes[28]

[28]http://alojamientos.us.es/gtocoma/pid/tema4.pdf

“La **segmentación** en el campo de la visión artificial es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. La segmentación se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen. Más precisamente, la segmentación de la imagen es el proceso de asignación de una etiqueta a cada píxel de la imagen de forma que los píxeles que compartan la misma etiqueta también tendrán ciertas características visuales similares.”[29]

Un ejemplo claro de la segmentación es la separación de objetos del fondo de la imagen, ya que los pixeles que se agrupan son aquellos que poseen o rodean una intensidad de color específica que es distinta del fondo de la imagen.

Los algoritmos de segmentación se basan en una de estas dos propiedades básicas de los valores del nivel de gris: discontinuidad o similitud entre los niveles de gris de píxeles vecinos.[30]

3.2.1. Discontinuidad

Divide los pixeles de las imágenes basándose en los cambios bruscos de los niveles de gris, obteniendo de ésta forma lo que son pixeles aislados, líneas y bordes de los objetos o cuerpos de una imagen.

3.2.2. Similitud

Es la unión de los pixeles que tienen similitud en sus características las más importante es la intensidad de color que cada pixel que conforma la imagen posee.

[29] http://es.wikipedia.org/wiki/Segmentaci%C3%B3n_%28procesamiento_de_im%C3%A1genes%29
[30] http://alojamientos.us.es/gtocoma/pid/tema4.pdf

3.3. Histogramas de color de una imagen

Los pasos mencionados anteriormente se basa en la determinación del histograma que muestra la imagen. El histograma viene dado por la cuantificación del número de pixeles de acuerdo a la intensidad de color, por ello que el proceso de segmentación se determina mediante agrupación de estos pixeles según su color. La separación de las partes de la imagen según su color depende de los valores de los picos más bajos de un histograma, quienes determinan el rango en la que se encuentran agrupados un grupo de pixeles.[31]

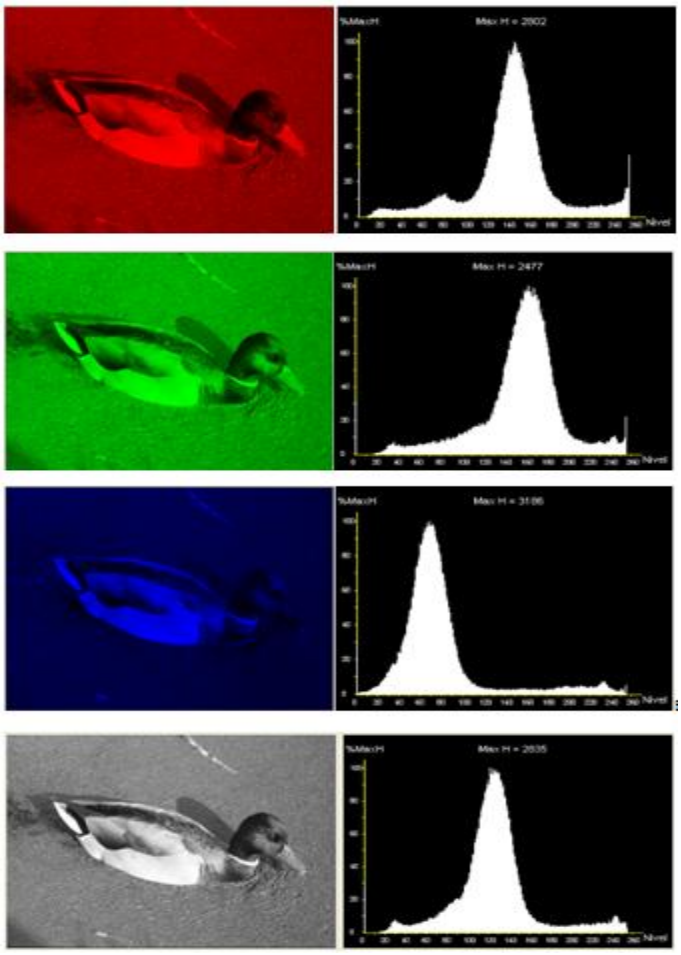


Foto 7: Histogramas de imágenes de color

[31]BURGE, Wilhelm, J. BURGE, Marck, Digital Image Processing and Algorithmic, 3ra Edición 564p, Austria/Springer, 2008. Pág. 38

3.4. Aplicaciones

Prácticas de la Segmentación:

- Medicina
    - Localización de tumores y otras patologías
    - Medida de volúmenes de tejido
    - Cirugía guiada por ordenador
    - Panificación del tratamiento
    - Estudio de la estructura anatómica
- Localización de objetos en imágenes de satélite
- Detector de huella digital
- Reconocimiento de caras
- Reconocimiento de iris
- Sistemas de control de tráfico
- Visión por computador

CAPÍTULO IV

TRIANGULACIÓN Y MALLADO MEDIANTE ALGORITMOS MATEMÁTICOS

4.1. Obtención de puntos

El proceso de obtención de puntos se basa en la identificación de las partes fundamentales de la fotografía, es decir la recolección de las coordenadas de cada una de las líneas que se proyectan en el rostro. Para ello se realiza un proceso de barrido de la imagen pixel a pixel, determinando cual es el pixel que pertenece a la línea del rostro, éstos están identificados por el color verde que se refleja en la cara de la persona como muestra la Foto8. Los puntos marcados con rojo son las coordenadas pertenecientes a rebanadas de color que se van a almacenar para su posterior tratamiento.

Las coordenadas corresponden a datos (X, Y), que se obtiene con un recorrido de la fotografía de izquierda a derecha y de arriba hacia abajo, llenando una matriz que contiene éstos datos organizados según la franja de color que se está barriendo.

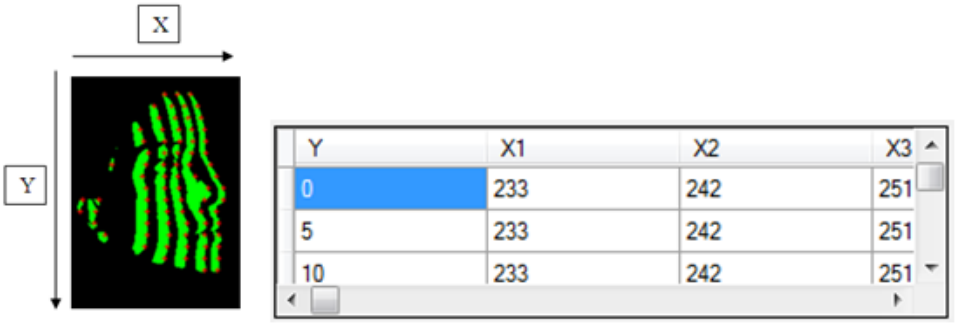


Foto 8: Barrido de imagen identificación de puntos.

4.2.Correspondencia de puntos

“El problema de la correspondencia consiste en encontrar, dadas dos o más imágenes de una misma escena tridimensional tomadas desde

diferentes puntos de vista, un conjunto de puntos en una de las dos imágenes donde cada uno de ellos pueda ser identificado con el mismo punto en la otra imagen."[32]

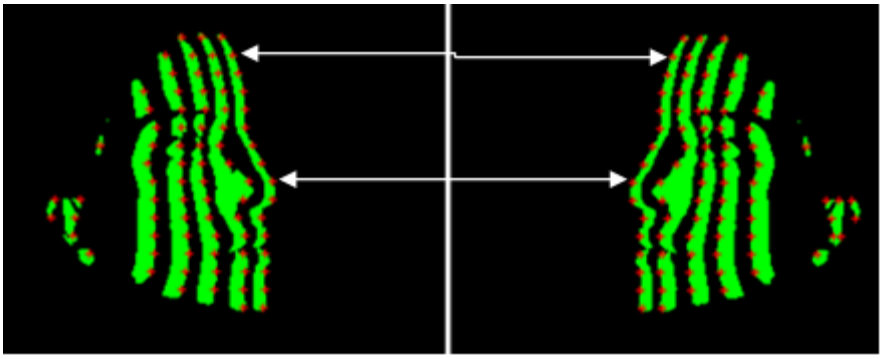


Foto 9: Correspondencia de puntos

En los sistemas estéreo, en que se obtiene un par de imágenes el problema de la correspondencia viene dado por la obtención de un punto X en la imagen izquierda y su correspondiente punto X' en la imagen de la derecha. Como las cámaras que forman el sistema estéreo tienen perspectivas distintas, el mismo punto se proyecta en localizaciones diferentes.

Los elementos más comúnmente empleados para el cálculo de la correspondencia son:

- **Pixel:** Utilización de las características del pixel como lo son: el nivel de gris, niveles de gris de la vecindad (pixeles adyacentes).
- **Borde:** Características de los segmentos rectilíneos que podría tener una imagen como son: longitud, orientación, nivel gris medio.
- **Pixel + borde**

Cabe mencionar que un sistema de visión estero está sujeto a ciertas restricciones que regulan el cálculo de la correspondencia entre puntos como lo son:

[32]http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol10/vol10issue5Sept2012/10TLA5_15Narvaez.pdf

La similitud: Se refiere a que un pixel de la imagen derecha posee las mismas propiedades en la imagen izquierda como es la del color.

La unicidad: Corresponde la representación única de un punto de una imagen en la otra, pues no existe la posibilidad de representar un pixel de una fotografía en varios pixeles de la otra foto.

La continuidad de la disparidad: Manifiesta que los pixeles de los objetos se encuentran a posiciones parecidas entre ambas imágenes y por ende la diferencia de las distancias entre ellos es similar.

La epipolaridad: Se define como la proyección de un mismo punto de la escena en distintas imágenes siguiendo una línea denominada línea epipolar.

Para hacer mención a los algoritmos utilizados para la correspondencia de puntos los clasificaremos en tres clases:

- Basados en las Características
- Basados en la Correlación

Estas dos clases de algoritmos son similares teóricamente lo que cambia en realidad es la forma de implementación de los mismos. **Ej.** La correlación es un método que se aplica a todos y cada uno de los pixeles de la imagen, en cambio los algoritmos basados en las características seleccionan un grupo aislado de pixeles que comparten ciertas características en común.[33]

4.2.1. Métodos basados en las características

Los métodos basados en las características lo que buscan es la similitud de pixeles y vecinos como lo es la intensidad del color y su adyacencia con el pixel, lo que lleva a obtener una medida suficiente para la

[33]http://iie.fing.edu.uy/investigacion/grupos/gti/cursos/egvc/material/tema-5.pdf

interpretación de la correspondencia entre áreas de dos imágenes del mismo cuerpo pero de distintos ángulos.

4.2.2. Métodos basados en correlación

“En los métodos basados en correlación, los elementos a emparejar son ventanas de la imagen de tamaño fijo y el criterio de similitud es una medida de la correlación entre las ventanas en las dos imágenes. El elemento en correspondencia esta dado por la ventana que maximiza el criterio de similitud dentro de una región de búsqueda” [32]

4.3.Interpolación de puntos

“Se denomina **interpolación** a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.”[34]

En nuestro proyecto la interpolación de los puntos está definida por los pixeles pertenecientes a cada línea de color como muestra la siguiente imagen.

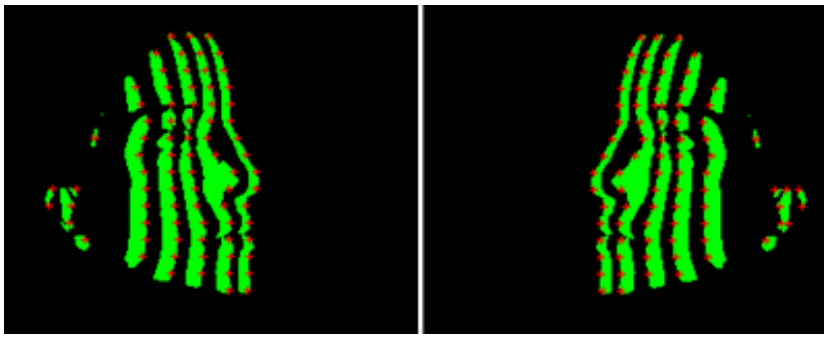


Foto 10: Imagen de interpolación de puntos en nuestro proyecto

[32]http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol10/vol10issue5Sept2012/10TLA5_15Narvaez.pdf
[34]http://es.wikipedia.org/wiki/Interpolaci%C3%B3n

La unión de los pixeles representativos para cada color de la imagen constituye una sección trasversal del rostro humano, lo que se vendría a interpretar como una rebanada de la cara y su correspondiente representación 3D, por lo que el proceso de interpolación se lo define al momento de proyectar nuestro patrón de colores con la luz puntual que en nuestro caso está dado por el infocus.

4.4. Mallas geométricas

Una malla es un conjunto de caras poligonales (cuadriláteros, triángulos o tetraedros) que definen una superficie en el espacio. Una malla tiene asociada un conjunto de elementos topológicos tales como: vértices, aristas y caras poligonales.[35]

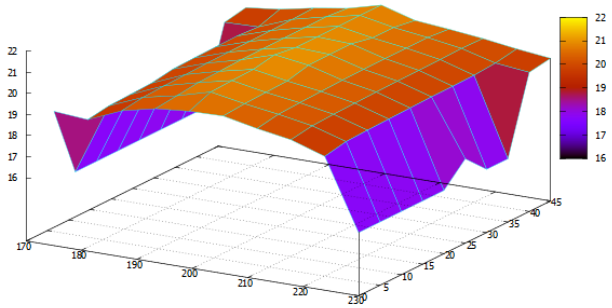


Gráfico 6: Ejemplo de malla tridimensional

Las mallas geométricas se pueden clasificar en estructuradas y no estructuradas.

4.4.1. Mallas estructuradas

Las mallas estructuradas se componen por celdas o caras que tiene un mismo tamaño y tipo como lo son polígonos regulares (triángulos, cuadrados, rectángulos, etc.). Estas mallas poseen una dimensión definida traduciendo esto como la cantidad de nodos en cada celda de la malla.[35]

[35]http://www.ubiobio.cl/miweb/webfile/media/182/resumen/resumenpedrorodriguez.pdf

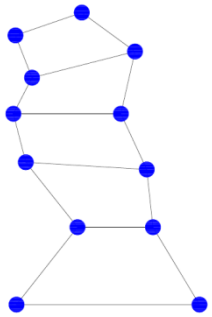


Gráfico 7: Malla geométrica estructurada

4.4.2. Mallas no estructuradas

La malla no estructuradas están constituidas por celdas o caras no son del mismo tipo o tamaño además de no tener definida una dimensión.[35]

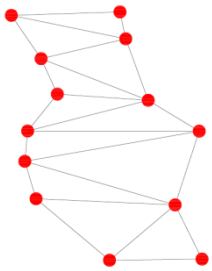


Gráfico 8: Malla geométrica no estructurada

“La triangulación computacional de un conjunto de puntos del plano constituye actualmente una técnica básica en aplicaciones de ingeniería moderna, tales como en modelación de terrenos, reconstrucción, reconocimiento de objetos y en el análisis de problemas físicos. Un algoritmo muy utilizado para la generación de mallas no estructuradas es el algoritmo de Triangulación de Delaunay.”[35]

4.5. Triangulación de Delaunay

Uno de los primeros algoritmos de generación de mallas fue propuesto por el matemático ruso Boris Nikolaevich Delaunay. Este algoritmo de triangulación se conoce como el algoritmo o la triangulación de Delaunay a través del cual se puede obtener, a partir de un conjunto de puntos en el

espacio, una triangulación cuyos elementos son lo más equiláteros posible.

Para muchos problemas en el campo de la ingeniería y de la ciencia, la triangulación de Delaunay no es suficiente para modelar un problema y resolverlo numéricamente. En muchos casos se necesita insertar nuevos puntos y elementos para mejorar la calidad de la malla, a través de un proceso denominado refinamiento.

4.5.1. Triangulación de una nube de puntos

La triangulación de una nube de puntos guarda una estrecha relación con el diagrama de Voronoi. Así, se tiene la siguiente.

Definición: "El punto X está dentro de la región de puntos más cercanos al punto pi. La región de Voronoi del punto pi la denominaremos V (pi). La unión de todas las regiones de Voronoi es el diagrama de Voronoi"[36]. El Gráfico9 muestra una descripción de ésta definición.

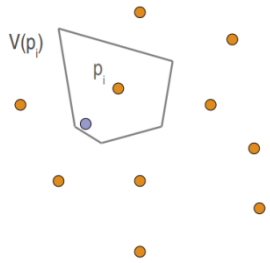


Gráfico 9: Diagrama de Voronoi

Definición: "Dado que un conjunto S= {s_1,…, s_n} de puntos de R^2, el dual geométrico de su diagrama de Voronoi es una triangulación de los puntos S. Dicha triangulación se llama triangulación de Delaunay."[37]

Propiedades

- "Minimiza el máximo radio de una circunferencia circunscrita.

[36] http://wwwdi.ujaen.es/asignaturas/gc/tema5.pdf
[37] http://personal.us.es/almar/docencia/practicas/triangulaciones/tema4.html

- Maximiza el mínimo ángulo (de hecho, la triangulación de Delaunay es equilátera)
- Maximiza la suma de los radios de las circunferencias inscritas.
- La distancia entre cualquier par de vértices a través de aristas de la triangulación es, a lo sumo, una constante por su distancia euclídea."[37]

Proposición: "Sea S= {s_1, …, s_n} un conjunto de puntos de R^2. Si proyectamos S sobre un paraboloide con ecuación $z=x^2+y^2$ (esto es, asociamos a cada punto s_i= (xi, yi) del plano el punto (x_i, y_i, $x_i^2 + y_i^2$) de R^3), entonces la proyección sobre R^2 de la parte inferior del casco convexo de los puntos en el paraboloide es la triangulación de Delaunay de S."[37]

"Definición: Dados dos triángulos que comparten un lado, formando un cuadrilátero convexo, se dice que el lado común es legal si maximiza el ángulo mínimo. El lado común es ilegal en caso contrario.

Definición: "Dados dos triángulos que comparten un lado, formando un cuadrilátero convexo, se denomina flip (giro) en la diagonal común al cambio de dicha diagonal por la otra en el cuadrilátero. Se dice que un flip es positivo si convierte una diagonal ilegal en legal."[37]

Gráfico 10: Diagonal ilegal en legal

Teorema: Una triangulación es de Delaunay si y solo si no contiene aristas ilegales.

Además, para saber si una diagonal es legal basta comprobar si la circunferencia que pasa por los puntos de la misma y un punto cualquiera del cuadrilátero al que pertenece no contiene al otro punto. En caso contrario, la diagonal es ilegal."[36]

4.5.2. Algoritmo Triangulación de Delaunay

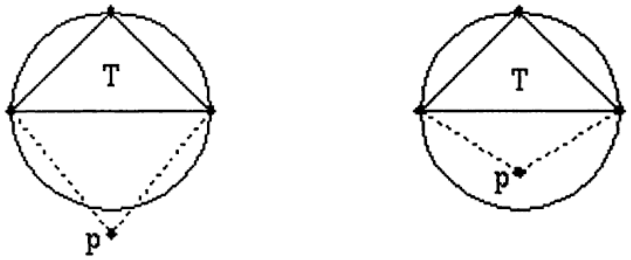


Gráfico 11: Algoritmo de Delaunay[37]

1. "Poner todas las aristas internas en una cola
2. Mientras la cola no esté vacía
   2.1 Sacar una arista, a, de la cola
   2.2 Si C_a tiene diagonal ilegal, hacer un flip positivo y añadir las aristas exteriores de C_a a la cola.

(C_a es el cuadrilátero con diagonal a)

Se puede disponer de una variante de este método, conocida como algoritmo incremental, en el que los puntos de la nube se introducen poco a poco, y la triangulación se optimiza tras cada inserción:

Entrada: T, triangulación de Delaunay, P, nuevo punto, p_{-1}, p_{-2}, p_{-3}, triangulo que contiene a T.

1. Encontrar el triángulo p_i, p_j, p_k en T en el que se encuentra P
2. Si p_i p_j p_k contiene a P
   2.1 Añadir las aristas de P a p_i, p_j, p_k
   2.2 Legalizar las aristas p_ip_j, p_ip_k, p_jp_k
3. Si no (está sobre una arista, p_ip_j por ejemplo)
   3.1 (Sea p_l el otro punto del triángulo que contiene a p_ip_j)
   Añadir las aristas de P a p_i, p_j, p_k, p_l
   3.2 Legalizar las aristas p_ip_l, p_lp_j, p_jp_k, p_kp_i

(Legalizar la arista p_ip_j) cuando se introduce p:

1. Si p_ip_j es ilegal
   1.1 (Sea $p_ip_jp_k$ el triángulo adyacente a pi_pí) Remplazar $p_ip_j$porpp_k
   1.2 Legalizar p_ip_k, p_jp_k)

Existen muchos criterios de optimización. Uno de los más famosos se conoce como triangulación de peso mínimo y consiste en buscar una triangulación de a nube de puntos que minimice la longitud total de las aristas."[37]

CAPÍTULO V

TEXTURIZACIÓN Y MALLADO

5.1. Interpretación visual

Es la forma en la que el cerebro humano interpreta las formas y los colores que son percibidos por el sentido de la vista.[38]

5.1.1. Forma de los objetos

Es una de las propiedades esenciales de los objetos, que muestran visualmente un punto, una línea o un plano que tienen un volumen específico y cuyas características como el tamaño, el color y la textura son vistos sin ningún inconveniente por el ojo humano. Cabe mencionar que la definición de forma y figura no son sinónimos, pues figura es la interpretación plana de una forma delimitada por líneas y que no tienen volumen. En resumen toda forma está compuesta por un conjunto de figuras.[39]

Las formas pueden interactuar entres si mediante la unión, sustracción, intersección, coincidencia de materiales llegando a producir efectos espaciales.

"Cuando una figura origina una forma en una superficie bi-dimensional, se la puede representar de varias maneras sin alterar su tamaño, color, composición y dirección. Visualizar una forma requiere la utilización de puntos, líneas y planos que describan sus contornos, características de la superficie y otros detalles. Cada método produce un efecto visual diferente, aunque el contorno general de la forma permanezca igual."[39]

[38]DAVID, Ebert, MUSGARVE, F. Kenton, Texturing and Modeling,4ta Edición 687p, Amsterdam/Morgan, Kaufmann Publishers, 2003. Pág 9
[39]http://foros.gxzone.com/110105-forma_y_figura.html

5.1.2. Color

El color es otra de las propiedades fundamentales de los objetos que define la interpretación de los mismos en el espacio.

"La luz blanca puede ser descompuesta en todos los colores (espectro) por medio de un prisma. En la naturaleza esta descomposición da lugar al arco iris."[40]

Espectro de color visible por los humanos:

El espectro electromagnético está constituido por todos los posibles niveles de energía de la luz. De todos los colores que se pueden apreciar mediante el espectro de luz el ser humano solo es capaz de distinguir muy pequeñas longitudes de onda que van desde los 380 nm hasta los 780 nm, donde cada longitud es interpretada como un color diferente.

Para la representación cuantitativa de cada uno de los colores los modelos más utilizados son los siguientes:

- **MODELO RGB (R, G, B):** Es la representación de cada uno de los colores en sus respectivos componentes primarios que son: rojo, verde y azul cuya combinación provee la intensidad adecuada para cada color. RGB se traduce como los nombres de sus correspondientes en ingles (RED, GREEN, BLUE). Las medidas de estos colores primarios están dadas por una escala que va desde 0 hasta el 255. El color rojo se representa como (255,0,0), el color verde(0,255,0), el color azul (0,0,255), el negro que es la ausencia de color como (0,0,0) y el blanco que es la combinación de la máxima capacidad para cada componente como (255,255,255). La combinación de dos colores primarios a nivel máximo, 255, con un tercero en nivel 0 da lugar a los tres colores secundarios, amarillo es (255, 255,0), el cyan (0, 255,255) y el magenta (255, 0,255).[40]

[40]BURGE, Wilhelm, J. BURGE, Marck, Digital Image Processing and Algorithmic, 3ra Edición 564p, Austria/Springer, 2008. Pág. 2,3,9

- **MODELO CMY (C,M,Y,K):** Se basa en la absorción de la luz por un objeto. CMYK se traduce como el acrónimo de éstos colores en ingles (CYAN, MAGENTA, YELLOW=amarillo, KEY=negro).La mezcla de todos estos colores dan como resultado el color negro.
- **HEXADECIMAL**: Representación especial de los colores, es una especie de codificación interpretada y utilizada en la creación de páginas web, el sistema de numeración hexadecimal, además de los números del 0 al 9 utiliza seis letras con un valor numérico equivalente; a=10, b=11, c=12, d=13, e=14 y f=15. El color blanco está dado por #ffffff y el negro como #000000.

5.2. Textura

Foto 11: Texturas

"La textura hace referencia normalmente a los rasgos visuales representados en la superficie de un objeto que da carácter e identidad al mismo en la representación. Suelen ser pequeños rasgos visuales que definen la relación de "veracidad" entre el objeto real y el objeto representado. Así la textura de una imagen o un fragmento de imagen, suele dar identidad diferenciando al objeto representado. Las texturas pueden integrarse en el conjunto de la imagen, aportando una sensación ambiental y pasando muchas veces desapercibidas en la imagen o en los objetos representados."[38]

Existen dos categorías de textura:

- **Textura visual**: Es el resultado del comportamiento de la luz sobre las superficies de los objetos. La forma de ver dicha superficie nos da una perspectiva únicamente visual.
- **Textura táctil:** Intervienen lo que es el tacto y la vista, pues con éstos sentidos se definen la calidad de la superficie.

5.2.1. Propiedades de la texturización de objetos

Las principales propiedades que son necesarias considerar en el proceso de texturizado son las siguientes:

Difusión: Consiste en la forma en la que la luz es reflejada por el objeto y la forma en que es reflejada en varias direcciones dependiendo del material del objeto.

Reflexión: La Cantidad de luz que es reflejada por el objeto

Transparencia: Es la cantidad de luz que traspasa a un objeto

Refracción: Constituye el fenómeno del cambio de trayectoria deluz reflejada en un objeto.

Brillo: Representa la cantidad de luz que produce un objeto.

5.3. Superficies y texturas

Las texturas representan una manera de simular la realidad de la superficie de un objeto.[41]

5.3.1. Métodos de texturización

Texturas por mapas de Bits:

Las texturas que están en formato de mapa de bits permiten disminuir el tiempo de cálculo computacional cuando son aplicadas a los objetos que

[41] http://www.eui.upm.es/~fmartin/Docencia/A3D/Teoria/Tema4/Presentacion.ppt

posteriormente se van a renderizar. Para la aplicación de ésta texturas el usuario debe tener un conocimiento medio de diseño 2D. Las texturas de mapas de bits se definen como aplicaciones de 2D a 3D. [41]

Texturas sobre superficies poliédricas

"Hay dos técnicas:

- Desenrollar el poliedro
- Proyectar sobre una superficie envolvente.

Desenrollar un poliedro: cierto tipo de poliedros se pueden desenrollar y obtener una superficie plana. A partir de ahí, se aplica el espacio de la textura sobre el poliedro desenrrollado. No todos los poliedros son desenrrollables. Muchas veces los algoritmos para desenrrollar son complicados y caros computacionalmente.

Proyección sobre superficies envolventes: las superficies envolventes son superficies intermedias sobre las que se proyecta primero el espacio de textura. Las superficies más usadas son el cilindro y la esfera. El método se generaliza a cualquier superficie simple. Una vez proyectada la textura sobre la superficie intermedia y estando dentro, el objeto se proyecta la textura sobre éste. "[41]

Mapas de relieve o de protuberancias

"Modelizan la rugosidad de un objeto por mapas de bits. Estos mapas son variaciones de las normales a la superficie. Producen una sensación muy fuerte de realismo. Se usa un vector D' que se suma a la normal N a una cara simplemente sumándola: N' = N+D." [41]

Texturas procedurales

"Son texturas definidas por fórmulas matemáticas. Muchas veces estas texturas simulan los comportamientos irregulares de las texturas de los materiales Se pueden animar fácilmente. Dan buenos resultados para las modelizaciones físicas de un buen número de parámetros: índice de

refracción, color, especularidad, etc. Proporcionan control global sobre la textura. No se puede controlar detalles finos adecuadamente.” [41]

CAPÍTULO VI

ANIMACIÓN Y RENDERIZADO

6.1. Conceptos básicos de 3D

La creación de objetos tridimensionales a nivel computacional constituye uno de los grandes avances de las imágenes por computadora, pues la representación de objetos en un ordenador se asemeja cada día más a la realidad. Esta reconstrucción tridimensional de formas y figuras se basa en una serie de cálculos matemáticos sobre entidades geométricas llegándolas a transformar de una visión en 2D a una visualización en tres dimensiones. Los objetos tridimensionales están formados por puntos, líneas, planos y volúmenes que en conjunto forman la interpretación realista de imágenes planas.

Los principales elementos para el modelamiento en 3D son las siguientes:

Sistema de Coordenadas: Es un grupo de datos que definir respectivamente la posición de cualquier punto de un espacio geométrico respecto de un punto central denominado origen de coordenadas.[42]

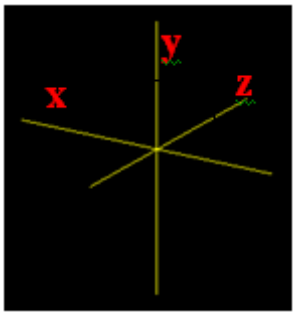


Foto 12: Eje de coordenadas

Los datos son distribuidos en el espacio según la especificación del nombre de los ejes, por lo general a estos ejes se los denota con X, Y, Z

[42]http://abc.mitreum.net/wp-content/uploads/clase2-parte1-teoria.pdf

identificando de ésta forma la posición de los objetos dentro del espacio antes mencionado.

Vector: “Es todo segmento de recta dirigido en el espacio. Cada vector posee unas características que son:

- Origen: Punto exacto sobre el que actúa el vector.
- Módulo: Longitud del vector
- Dirección: Orientación en el espacio de la recta que lo contiene
- Sentido: Se indica mediante una punta de flecha situada en el extremo del vector.”[41]

Segmento: Es la parte de una recta que está delimitado mediante dos puntos que encuentren sobre ella.

Puntos: Es la parte fundamental de la geometría que no tiene ni longitud ni área y que define una posición en el espacio.

Líneas: Es la sucesión continua de varios puntos.

Polígonos: Figuras delimitadas por líneas y puntos que se cierran éstas pueden estar formadas por lados iguales o desiguales. La representación de las mismas es en dos ejes X, Y por lo que son formas planas o figuras 2D.

Iluminación: “La iluminación correcta de la escena es fundamental para imprimirle realismo. La iluminación corresponde a todo un apartado dentro de la asignatura y en él aprenderemos a colocar las luces adecuadas y modificar sus parámetros para obtener el resultado deseado” [41]

Textura: Es una imagen que se ajusta a un objeto tratando de simular el mayor realismo posible.

Malla: Es una colección de polígonos 2D cerrados que tienen lados y vértices comunes.

Escena: “La escena en 3D es el archivo que contiene toda la información necesaria para identificar y posicionar todos los modelos, luces y cámaras para su renderización.” [41]

Material: Corresponde al aspecto que tiene un objeto es decir el tipo de superficie, ésta puede ser lisa, rugosa, con grumos, suave, dura, etc. Ej. (vidrio, metal, hielo).

Render: El render es el proceso de producir imágenes desde una vista de modelos tridimensionales, en una escena 3D.En palabras sencillas, es “tomar una foto” de la escena.

Animación: Es una sucesión continua de renderes que las que los objetos tienen modificado ciertas características como los son: escalamiento, rotación y traslado.

6.1.1. Modelado de objetos en 3D

Es la creación y formación de los elementos que intervienen en una escena.

El modelo en 3D

Es el documento en sí que contiene los datos necesarios para la renderización de una escena en 3 dimensiones.

Las características esenciales de estas imágenes o escenas es el detalle que tiene casa uno de los objetos como son:

- La geometría, forma del objeto.
- Tipo de superficie de los objetos, o sea, el color y el tipo de material que está determinando la manera de la cual está hecho.

“La geometría del modelo define las superficies del objeto como una lista de polígonos planos que comparten lados y vértices. El modelo por tanto describe una malla.” [41]

En el modelación de objetos en 3D hay que tener en cuenta que todas las caras, vértices y aristas deben estar íntimamente ligados para evitar tener huecos o discontinuidades en la superficie de las formas, éstos huecos en el objeto conllevaría a la construcción de una formación de irregularidades en su superficie dejando inconcluso el arte de la modelación en 3 dimensiones.

La discontinuidad en las superficies provoca la falta de detalle en los elementos de la escena por lo que el realismo no tiene mucho valor en éstos renderizados.

6.1.1.1. Tipos de modelos 3D

Modelos representados por polígonos:

En las computadoras el método más utilizado para la representación de formas en 3 dimensiones se basa en la generación de estructuras de polígonos. “Un cubo tiene 6 caras, cada una de ellas es un polígono —un cuadrado—; una pirámide se compone de 4 triángulos y una base cuadrada. El ejemplo más real lo podemos ver en un balón de fútbol, que se compone de 12 pentágonos y 20 hexágonos.” [41]

Modelos definidos por sus curvas matemáticas: NURBS

Son superficies curvas definidas matemáticamente, sin embargo existe programas especializados que permiten empezar el modelado con la generación de curvas, elipses, círculos etc.

Todas las formas que se crean mediante las NURBS generan fórmulas matemáticas que los programas para diseño y modelación como: 3dMax, Blender, Ilustrador, Photoshop, etc., calculan internamente, lo que contribuye a que los usuarios se eviten un proceso largo de cálculo para la creación de superficies de éste tipo. [41]

En un modelador no poligonal se disponen de diferentes tipos de herramientas (SPLINES, NURBS, CURVAS DE BÉZIER) para crear superficies de curvas complejas.

6.1.2. Composición de la escena

Foto 13: Escena[40]

Este proceso consta de una serie de parámetros importantes para su visualización como son: cámaras, luces, objetos, texturas que pueden ser estáticas o con un movimiento determinado que es la animación.

"La iluminación es un aspecto importante de la composición de la escena. Como en la realidad, la iluminación es un factor importante que contribuye al resultado estético y a la calidad visual del trabajo terminado. Por eso, puede ser un arte difícil de dominar. Los efectos de iluminación pueden contribuir en gran medida al humor y la respuesta emocional generada por la escena, algo que es bien conocido por fotógrafos y técnicos de iluminación teatral. " [41]

Iluminación de la escena

La intensidad de luz que se presenta en una escena depende del tipo de luz que se está incorporando en la misma, ya que la distancia y el ángulo de inclinación de la fuente de luz al objeto son factores fundamentales que determinan la visualización de la escena con características lo más realistas posible.

"Algunos objetos tienen superficies brillantes y algunos tienen superficies opacas o mate. Además, algunos objetos se construyen con materiales opacos, mientras que otros son más o menos transparentes. Un modelo de sombreado para producir intensidades realistas sobre superficies de un objeto debe tomar en consideración estas diversas propiedades. Cuando se observa un objeto, se percibe la intensidad de luz reflejada de sus superficies. La luz que se refleja de las superficies proviene de las diversas fuentes de luz que rodean al objeto. Si el objeto es transparente, también se percibe luz de cualquier fuente que pueda estar situada detrás del mismo. " [41]

Hay dos tipos de iluminación según la fuente de luz de la que provengan:

Luz Natural: "Proviene del sol y las estrellas por lo que es muy difícil de controlar ya que depende de muchos condicionantes como son la intensidad, la dirección, el color y la calidad. "[43]

Luz Artificial: "Puede ser **continua** (con su fuente de procedencia de las bombillas y los focos) o **discontinua** (flash). Con ella podemos controlar todas las cualidades que son imposibles tomar en cuenta con la luz natural. El inconveniente con el que nos encontramos es que limita el espacio iluminado." [42]

"Los factores que se necesita tener en cuenta para la iluminación de una escena

- **Número de fuentes luminosas**: Influye en el contraste y modelado de la imagen.
- **Difusión**: Esta determina la nitidez del borde de las sombras. Caracteriza la dureza o suavidad de la imagen. La **luz dura** produce efectos fuertes y espectaculares. Por otro lado laluz suave consigue que el volumen del motivo domine sobre las líneas. La luz dura procede de fuentes pequeñas y alejadas. Dependiendo de la distancia y el tamaño el grado de dureza será más o menos

[43]http://www.desarrollomultimedia.es/articulos/iluminacion.html

fuerte. Proporciona mayor grado de contraste y destaca la textura, forma y el color de los objetos. La luz suave es muy difusa y no proyecta casi sombras. La fuente luminosa es muy extensa y puede conseguirse rebotando luz sobre una superficie muy grande y próxima como el techo. Es la luz menos espectacular pero la más agradable y fácil de controlar.

- **Intensidad**: Dependiendo de esta se intensificarán los colores y los objetos.
- **Color**: Este viene definido por la longitud de la onda. Dependiendo del objeto la cámara captará diferentes longitudes de onda caracterizando a éste de color.
- **Dirección** de la luz: se mide teniendo en cuenta la posición de la cámara y el objetivo. Puede ser:
    - **Frontal**: Es la iluminación más fácil de usar y los resultados que obtenemos son buenos. Los colores que conseguiremos serán más brillantes. Las sombras que obtendremos van a estar detrás del sujeto por lo que no aparecerán en la imagen final.
    - **Lateral**: Con ella resaltamos el volumen y la profundidad de los objetos o sujetos. Apreciamos más la textura de forma que la imagen obtiene mucha fuerza. Las sombras que obtenemos pueden ocultar detalles.
    - **Contraluz**: Es muy difícil de utilizar, sin embargo si lo sabemos utilizar los resultados pueden ser muy buenos. La fuente de luz proviene de la parte posterior, de detrás, de tal forma que las sombras se proyectan hacia la cámara dando profundidad a la escena.
    - **Cenital**: La fuente de luz proviene desde arriba. Con esta, las partes inferiores del objeto quedan en sombra y hace que los detalles sean más vistosos y sobresalgan.

- **Por todas partes**: Se trata de una luz suave y uniforme. No se producen sombras y mejora mucho el aspecto de las personas. Produce colores muy sutiles.” [42]

6.2. Animación

Es el proceso de conceder a los objetos de la escena una sensación de movimiento.

La animación puede estar definida por un cambio en las características del objeto como lo son: la posición, el tamaño y rotación. Este proceso no solo se lo realiza en 3 dimensiones sino en dibujos planos. El objetivo fundamental es mostrar una escena lo más parecido a la realidad posible. Esto lleva al observador a presenciar un evento casi real sin tener que estar presente en la escena misma de los hechos como por ejemplo un viaje virtual a otro planeta.

El proceso de dar movimiento a los objetos en un espacio tridimensional se forma con una serie consecutiva de imágenes es decir como una fotografía por cada instante en que se produce el movimiento.

La animación también puede ser realizada por la incrustación de esqueletos o huesos en los objetos que ayudar a tener un movimiento con mucho más detalle que las simples transformaciones lineales, éste método de animación llega a afectar la forma del modelo a tal punto que es sumamente necesario la unión del objeto al hueso. En caso de que el acoplamiento del modelo al esqueleto no esté correctamente realizado la animación solo estará aplicada al esqueleto dejando el modelo deformado.

6.3. Renderizado

“El renderizado es un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen 2D a partir de una escena 3D. Así podría decirse que en el proceso de renderización, la computadora "interpreta" la escena 3D y la plasma en una imagen 2D.

La renderización se aplica a los gráficos por ordenador, más comúnmente a la infografía. En infografía este proceso se desarrolla con el fin de imitar un espacio 3D formado por estructuras poligonales, comportamiento de luces, texturas, materiales, animación, simulando ambientes y estructuras físicas verosímiles, etc.

Cuando se trabaja en un programa de diseño 3D por computadora, no es posible visualizar en tiempo real el acabado final deseado de una escena 3D compleja ya que esto requiere una potencia de cálculo demasiado elevada. Por lo que se opta por crear el entorno 3D con una forma de visualización más simple y técnica y luego generar el lento proceso de renderización para conseguir los resultados finales deseados.” [41]

CAPÍTULO VII

DESARROLLO DEL SISTEMA

7.1. Bases matemáticas y geométricas

La geometría que describe el proceso de reconstrucción 3D basado en Iluminación Estructurada está formado por un esquema inicial en el que se basa nuestro trabajo de gradocomo muestra la siguiente figura:

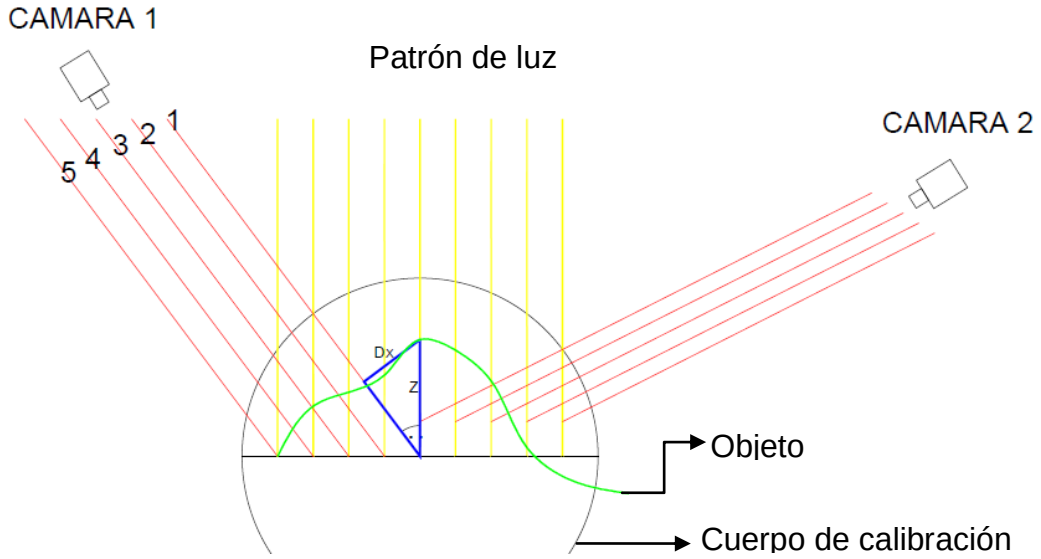


Figura 13: Boceto inicial cálculos matemáticos

El círculo descrito en la imagen anterior constituye el objeto de dimensiones conocidos con el cual se va realizar el proceso de calibración de cámaras y en lo posterior nos referiremos de este objeto como cuerpo de calibración.

Las cámaras indicadas en las esquinas del esquema están separadas a una distancia equidistante del eje central de referencia, las mismas proyectan líneas paralelas orientadas al centro del círculo. La representación del rostro en la escena está dada por la línea verde inscrita en la circunferencia. El patrón de color que proyecta nuestra fuente de luz constituye las líneas amarillas de la Figura13; con lo antes mencionado procedemos a la descripción de la geometría.

Para el proceso de reconstrucción es necesario tomar cuatro fotografías por cada lado de la cara:

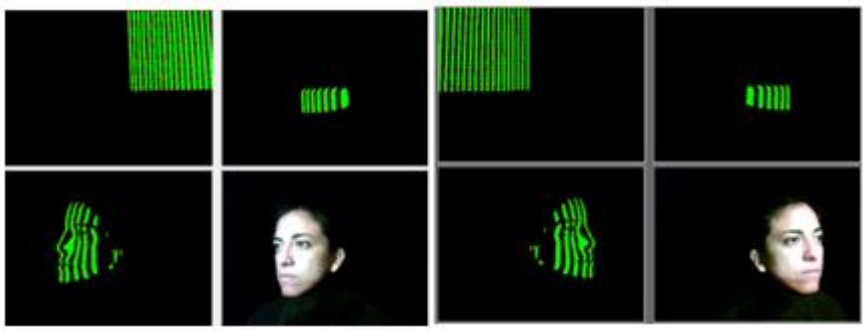


Figura 14: Fotografías de cada lado del rostro

- Foto1: El patrón de color proyectado sobre el fondo.
- Foto2: El patrón de color proyectado sobre el cuerpo de calibración.
- Foto3:El patrón de color proyectado sobre el rostro de la persona
- Foto4. Imagen de la persona con luz natural.

La intersección entre las líneas de proyección de las cámaras, el cuerpo de calibración, el rostro humano y el patrón de color establecen la geometría del proceso de reconstrucción.

A continuación mostramos el triangulo rectángulo con los datos necesarios.

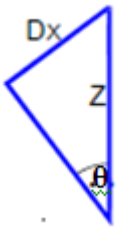


Figura 15: Triángulo Rectángulo

Donde,

Con **z** conocido siendo éste el diámetro de nuestro cuerpo de calibración,

Dx quees la diferencia en la coordenada x de las líneas proyectadas en fondo con las líneas proyectadas en el cuerpo.

Donde n, es el número de puntos encontrados.

Una vez conocido el parámetro K aplicamos la misma fórmula a cada punto de las líneas proyectadas sobre la cara.

Cálculo de zi (profundidad por punto de la cara)

—

Dx que es la diferencia en la coordenada x de las líneas proyectadas en fondo con las líneas proyectadas en el rostro.

7.2. Infraestructura básica del sistema de iluminación estructurada

Nuestro sistema de Iluminación estructurada está formado por los siguientes equipos:

- Proyector [Ver Anexo2]
- Dos cámaras web
- Extensiones para conexión USB de las cámaras.
- Computadora
- Silla
- El modelo (Rostro Humano)
- Cuerpo de Calibración
- Armadura recubierta con tela negra (simulación cuarto oscuro)

Las condiciones necesarias para tener un ambiente óptimo en la captura de imágenes con nuestra estructura son las que mencionamos a continuación:

- Para limitar el área a reconstruir es necesario que la persona lleve puesto una tela negra que tape desde el cuello hacia abajo.
- Luz ambiente debe ser oscura.
- La armadura que cumple el papel de cuarto oscuro debe estar cubierta por una tela negra gruesa, la misma que debe cumplir la propiedad absorbente, además debe ser estable para evitar complicaciones en las capturas de imágenes posteriormente.
- Las cámaras deben estar fijas y previamente configuradas con el centro de enfoque orientado al mismo punto.
- El proyector debe estar ubicado frente al cuerpo de tal manera que las líneas de color se reflejen verticalmente evitando deformaciones o inclinaciones en las líneas proyectadas.
- El fondo de la armadura con la proyección, el cuerpo de calibración y el rostro de la persona deben situarse a la misma altura.
- Las líneas proyectadas deben cubrir toda la zona de la cara que se va a reconstruir.

7.3. Patrón de color

7.3.1. Modo de proyección

En el software se incluyó la forma de crear el patrón en una ventana adicional que permite controlar desde el mismo computador los colores que se proyectan sobre el rostro. [Ver ANEXO 8 y ANEXO 9]

En la ventana **PATRÓN** muestra las siguientes opciones que permiten modificar las características de las líneas a proyectar.

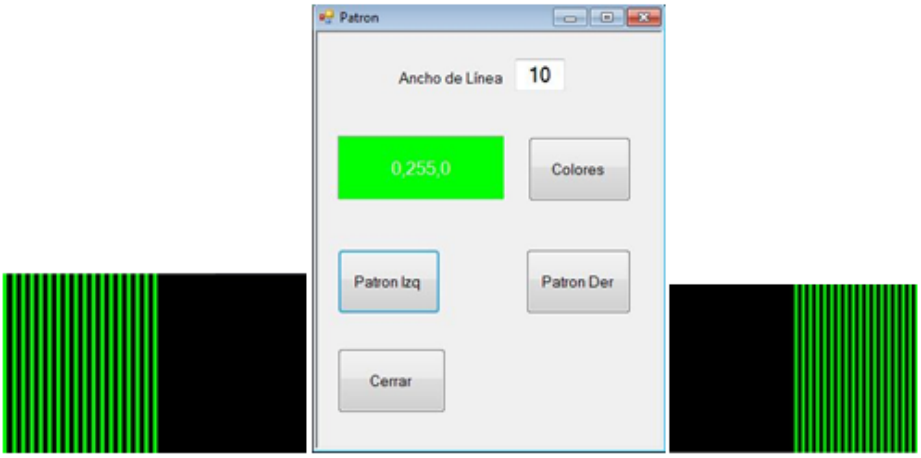


Figura 16: Ventana para controlar las carácterísticas de las líneas

Ancho de línea: Es la distancia entre líneas y ancho de línea.

Colores: Selección del color a proyectar.

Patron Der: Líneas proyectadas en el lado derecho y color negro para la otra mitad.

Patron Izq:Líneas proyectadas en el lado izquierdo y color negro para la otra mitad.

7.3.2.Elección del patrón de color

Debemos tomar en cuenta los conceptos básicos de la modificación del color (visto en Capítulo III), de tal manera que podamos extraer las propiedades de color RGB en cada pixel. [Ver ANEXO 8 y ANEXO 9]

Para determinar el color con el cual se va realizar el proceso de cálculo de datos se realizaron las siguientes pruebas.

7.3.2.1. Color rojo

La proyección del patrón de color rojo sobre el rostro provoca una gran cantidad de ruido y se pierde definición en las líneas ya que la componente roja del tono de piel es más elevada que la de los otros

componentes, lo que hace que el cálculo y discriminación del fondo de la imagen sea más complejo de procesar.

Figura 17: Patrón rojo sobre el rostro

7.3.2.2. Color azul

La proyección del patrón de color azul en el rostro de la persona, provoca una gran cantidad de brillo y ruido, además es mucho más notorio las líneas azules en el fondo de la imagen por tanto el proceso de cálculo se complica a tal punto que devuelve discontinuidades en las rebanadas de color proyectadas en el rostro.

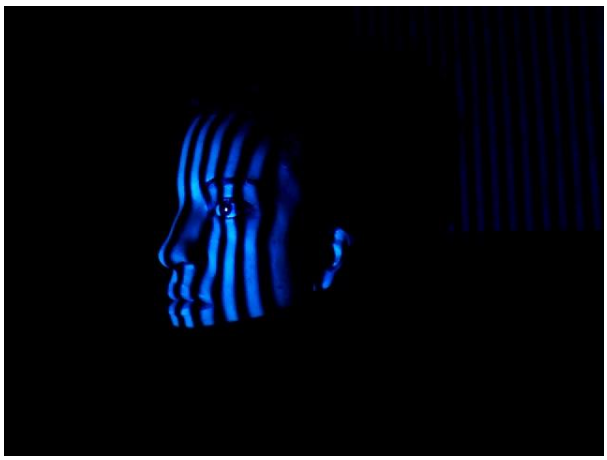


Figura 18: Patrón azul sobre el rostro

7.3.2.3. Color verde

La proyección del patrón de color verde en el rostro de la persona ayuda con la discriminación del fondo, causa menos brillo y ruido lo que contribuye con la obtención de los puntos que forman las líneas proyectadas. Con éste color delimitamos de forma específica las características en la fisionomía del rostro.

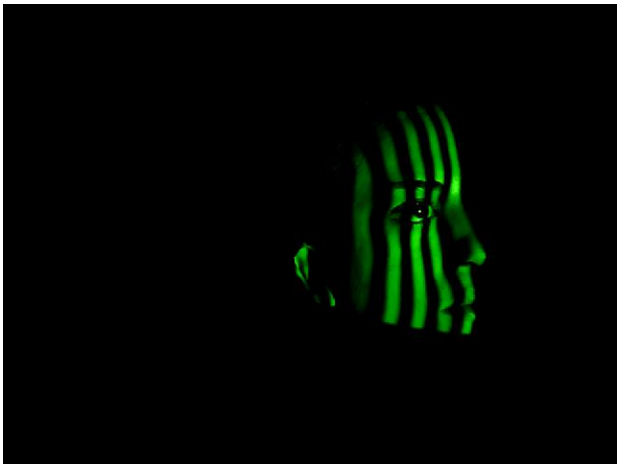


Figura 19: Patrón verde sobre el rostro

7.3.2.4. Patrón de varios colores

La proyección de un patrón de varios colores sobre el rostro de la persona ocasiona una serie de inconvenientes tanto de cálculo como de visualización ya que las fotografías contienen mucho ruido, brillo y la dircriminacion del fondo es un proceso mas complejo. Las pruebas iniciales las realizamos con este modelo de luz, lo que nos llevó a la búsqueda de un patrón de luz estructurada más óptima que sea fácil de procesar para poder cumplir con los objetivos de nuestro trabajo de grado.

Figura 20: Patrón de colores sobre el rostro

Con todas las pruebas realizadas llegamos a la conclusión que el mejor color para la proyección de la luz estructurada es el verde, pues nos da las mejores condiciones para el cálculo de la geometría de la reconstrucción tridimensional.

7.4. Proceso de captura de imágenes

Con todos los elementos en la escena procedemos a la captura de las fotografías necesarias para ello empezamos la utilización de nuestro software. [Ver ANEXO 5]

Las imágenes que vamos a obtener de esta captura están en formato JPG, las mismas que en lo posterior convertiremos a formato BMP, para el procesamiento digital de la imagen y una vez lograda la reconstrucción, lograríamos incluso generar una animación básica renderizada en formato GIF.

Iniciamos el video de una cámara a la vez y luego procedemos a la captura de las 4 fotografías por cada lado del rostro.

Figura 21: Captura de Imágenes de los dos lados de la cara

Para el procesamiento y la captura de las imágenes utilizamos un Framework llamado AForge.

"AForge.NET es un Framework desarrollado C#, el cual fue desarrollado para el procesamiento digital de imágenes, en el campo de la Visión Computacional e Inteligencia Artificial."[44]

Es importante destacar que el AForge, funciona de manera correcta con el .NET Framework 2.0, ya que en versiones más recientes genera conflictos y con este estamos trabajando en el proyecto.

NOTA IMPORTANTE

En la captura de imágenes pudimos observar que es importante que el patrón proyectado sobre el fondo llegue hasta el marco superior de la imagen ya que nos basamos en ésta estructura para el cálculo de la constante K y Z.

Una vez capturadas las imágenes éstas se guardan en forma secuencial en una carpeta que se crea automáticamente con la fecha actual. La ruta de las fotografías es C:\FotosEscaneo3D\dd-mm-yyy\

[44]http://www.aforgenet.com/

Como para el proceso de toma de fotos usamos cámaras web; la resolución de las imágenes con las que estamos trabajando en el proceso de reconstrucción es VGA (640x480px).

7.5. Limpieza de la imagen y obtención de datos

En la ventana de **FILTROS** muestra las siguientes opciones para procesamiento digital de las imágenes con lo que logramos tener líneas más definidas. [Ver ANEXO 6 y ANEXO 7]

Para el proceso de eliminación de ruido y purificación del color descomponemos cada una de las imágenes proyectadas en sus respectivas componentes de color en el sistema RGB y tomamos el componente G como el color predominante.

Una vez separada la componente G (verde) de las imágenes, purificamos el color basándonos en umbrales, los cuales nos sirven para discriminar cualquier residuo del fondo o ruido generado. El valor del umbral es 70 (color.G>= 70). De esta manera todos los valores mayores al umbral los consideramos puros en su componente G (color.G = 255). Los valores que están por debajo del umbral definido son discriminados y los consideramos color negro.

A continuación, definimos el alto del barrido que va a recorrer y el ancho de la línea proyectada para hacer un barrido por las imágenes y encontrar las coordenadas X e Y que se generaron con la luz estructurada sobre los cuerpos.

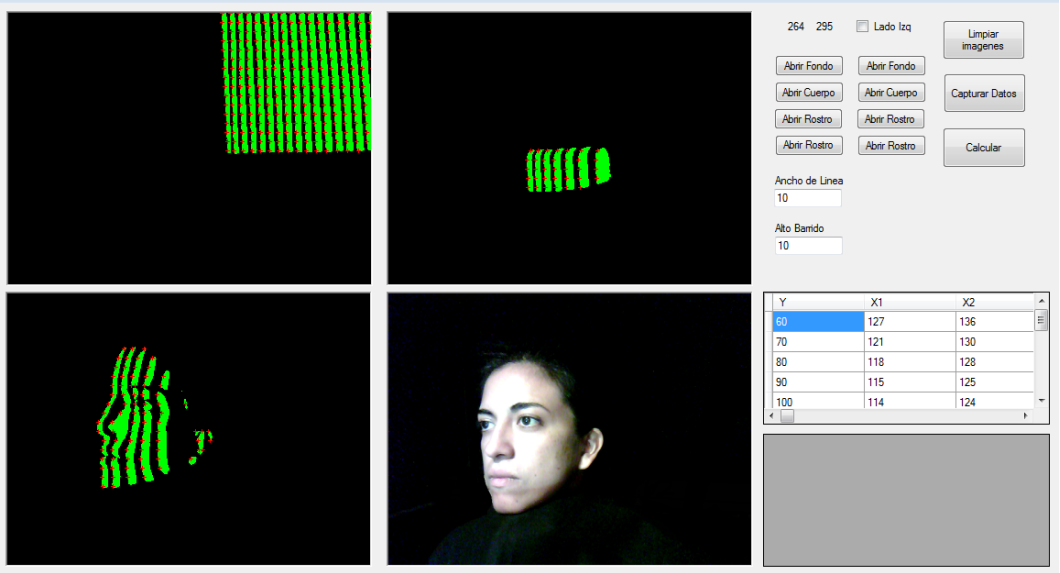


Figura 22: Limpieza y barrido de imagen

Mostramos los datos capturados en la siguiente ventana

En la ventana de **CAPTURAR PUNTOS** se muestra un detalle de todos los datos que se han obtenido durante el proceso de barrido.

Los cuadros están separados según la imagen donde se realizó el barrido definiéndolos como:

- Coordenadas obtenidas del fondo
- Coordenadas obtenidas del cuerpo de calibración
- Coordenadas obtenidas del rostro de la persona
- Cálculo de la coordenada z de cada uno de los puntos de la cara.

Donde las líneas están definidas como X1, X2, X3… etc., según el orden en el que fueron encontradas como muestra la siguiente imagen.

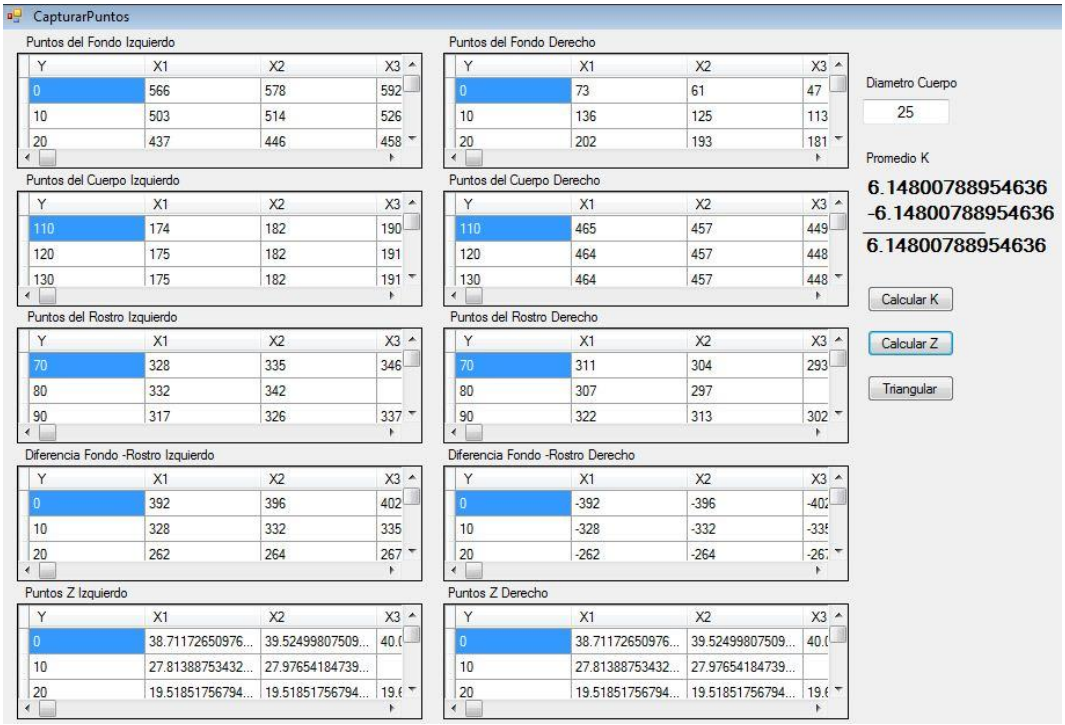


Figura 23: Obtención de coordenadas para cada una de las imágenes

Como podemos observar en el lado derecho de la imagen anterior se ingresa los datos del diámetro del cuerpo de calibración lo cual nos permite calcular la constante K de nuestro proceso y de ésta manera poder obtener la coordenada z en cada punto encontrado del rostro.

Una vez terminado el proceso de cálculo de la coordenada Z, obtenemos la reconstrucción de cada lado del rostro. Usamos la correspondencia de puntos y métodos de escalamiento y traslación para unificar los 2 lados del rostro, conociendo que el parámetro común entre los 2 lados es la línea central que esta proyectada sobre la nariz.

Este proceso nos da como resultado una nube de puntos que corresponden a las coordenadas de cada franja de color proyectado sobre el rostro.

7.7. Triangulación y mallado

En la ventana de **TRIANGULACIÓN** se puede observar la imagen en 2D de los lados del rostro ya unificados y representados por coordenadas. [Ver ANEXO 10]

Los listados incluidos en ésta pantalla muestran todos los datos obtenidos a manera de secuencia tanto los datos de coordenadas del rostro como los colores en RGB asociados a dichas coordenadas.

El proceso de mallado se lo realizó con la Triangulación de Delaunay que ya se explicó en el CAPITULO IV.

En el software incluimos también el listado de los triángulos generados por éste proceso matemático, éstos datos son los que se dibujan en la generación de la malla como muestra la siguiente imagen como líneas azules sobre los puntos 2D.

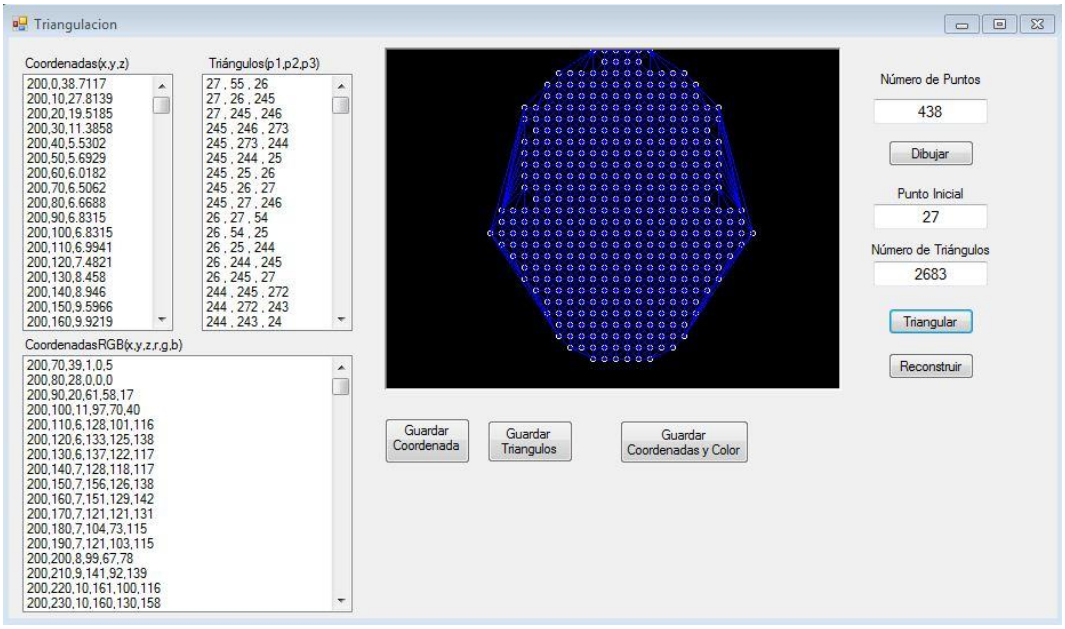


Figura 24: Triangulación y Mallado

Adicionalmente en la pantalla anterior se puede observar que tenemos el número total de puntos obtenidos y el número de triángulos generados.

7.8. Textura

En la Figura 24 como ya se mencionó tenemos un listado de las coordenadas de color de cada uno de los puntos obtenidos para la reconstrucción, Los datos de color están en RGB y son capturados de la cuarta fotografía tomada al modelo con luz natural. En siguiente paso podemos ya tener una idea más clara de todos los cálculos y procesos dichos anteriormente.

7.9. Reconstrucción

En la ventana de **RECONSTRUCCIÓN** se puede observar la imagen en 3D con detalles sumamente realistas. . [Ver ANEXO 11]

Para lograr la visualización en el espacio 3D utilizamos OpenGL.

OpenGL: "es una especificación estándar que define una APImultilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por SiliconGraphics Inc. (**SGI**) en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows."[45]

[45] http://es.wikipedia.org/wiki/OpenGL

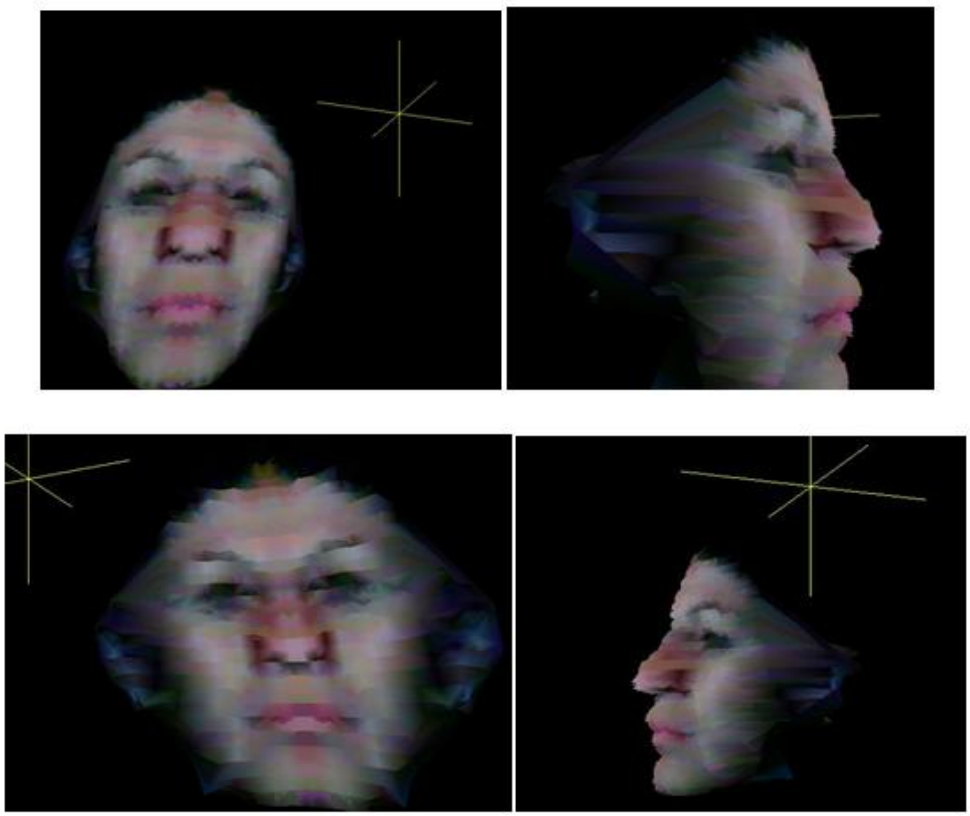


Figura 25: Reconstrucción

Como vemos en la Figura 25, la reconstrucción muestra una imagen sumamente realista que se asemeja en la mayoría de aspectos a la realidad, pues las facciones de la cara más pronunciadas como la nariz, los labios, las cejas, los ojos de pueden apreciar notablemente.

Todas las clases utilizadas para la programación están en los ANEXOS 12, 13, 14, 15.

7.10. Animación

Para el proceso de animación es necesario exportar los puntos generados de la malla y su respectivo color a un programa de modelamiento en 3D.

Para esta exportación usamos un archivo PLY(Python Lex-Yacc), que es un formato de archivo informático conocido como el formato de archivo de Polígono.

“El formato fue diseñado principalmente para almacenar datos tridimensionales desde escáneres 3D. Es compatible con una descripción relativamente sencilla de un solo objeto como una lista de polígonos nominalmente planas. Una variedad de propiedades se pueden almacenar incluyendo: color y transparencia, normal.”[46]

Dentro de este archivo esta incluido una cabecera, donde identificamos los parámetros de entrada, el número de puntos, dirección de la normal de cada punto y el número de triángulos generados en el mallado. Seguido de este incluimos la lista de coordenadas encontradas y a continuación la lista de los triángulos generados. Como se muestra a continuación [Ver ANEXO15]:

ply
format ascii 1.0
elementvertex NúmeroPuntos
propertyfloat x
property float y
property float z
property float nx
property float ny
property float nz
propertyuchar red
propertyuchar green
propertyuchar blue
element face NúmeroTriángulos
property list ucharuintvertex_indices
end_header
x y z nxnynz r g b
lados t1 t2 t3

[46]http://en.wikipedia.org/wiki/PLY_%28file_format%29

Con el archivo PLY listo, procedemos a la importación en un programa de modelamiento y animación 3D.

Para este trabajo de grado usaremos Blender.

“Blender es un programa informático multiplataforma, dedicadoespecialmente al modelado, animación y creación de gráficos tridimensionales.

El programa fue inicialmente distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX.

Tiene una muy peculiar interfaz gráfica de usuario, que es criticada como poco intuitiva, pues no se basa en el sistema clásico de ventanas; pero tiene a su vez ventajas importantes sobre éstas, como la configuración personalizada de la distribución de los menús y vistas de cámara.”[47]

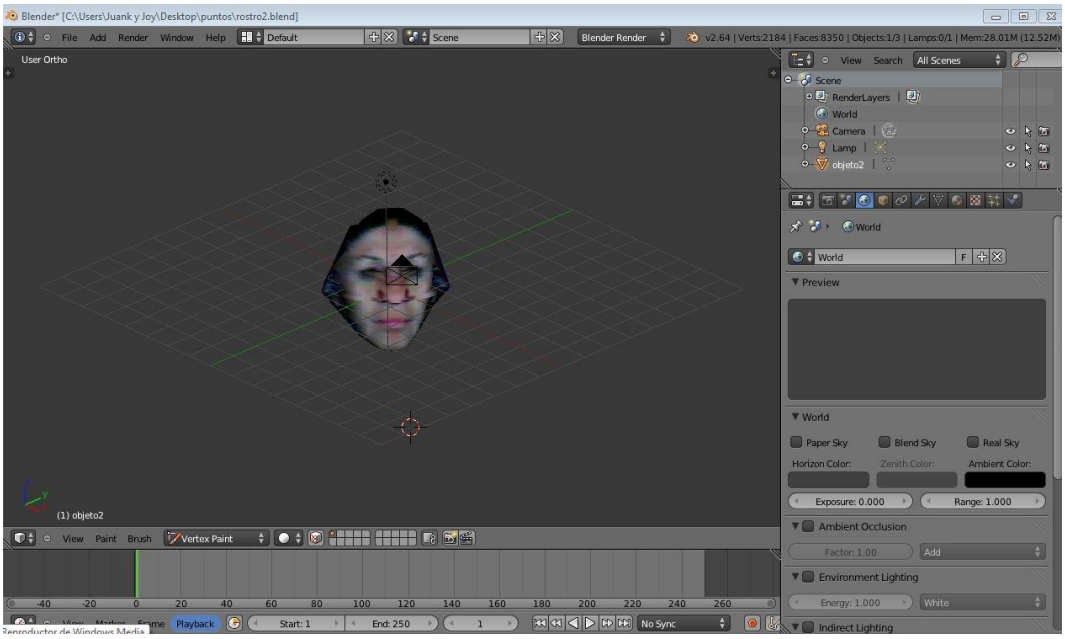


Figura 26: Ambiente y objeto de animación con datos importados.

[47]http://es.wikipedia.org/wiki/Blender

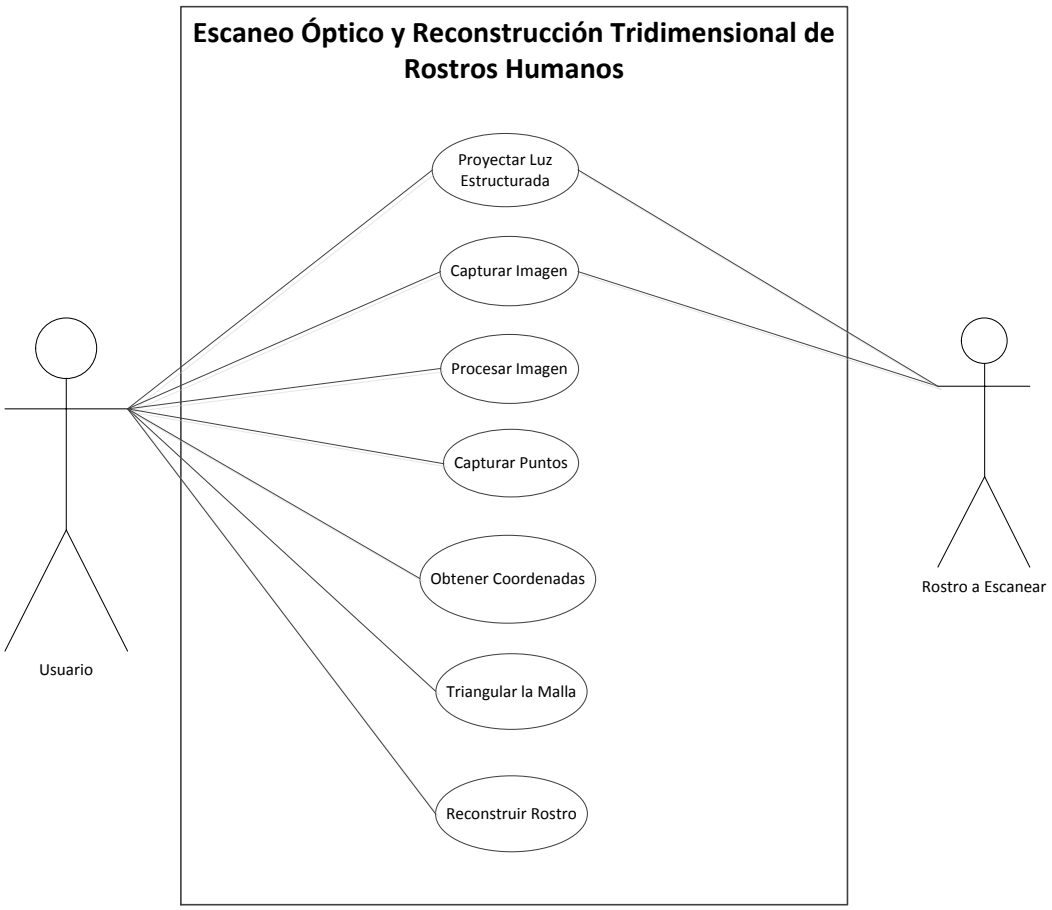
Una vez importados los datos al modelador la animación queda a elección libre del diseñador según las necesidades de la escena a renderizar

7.10. Diagrama uml

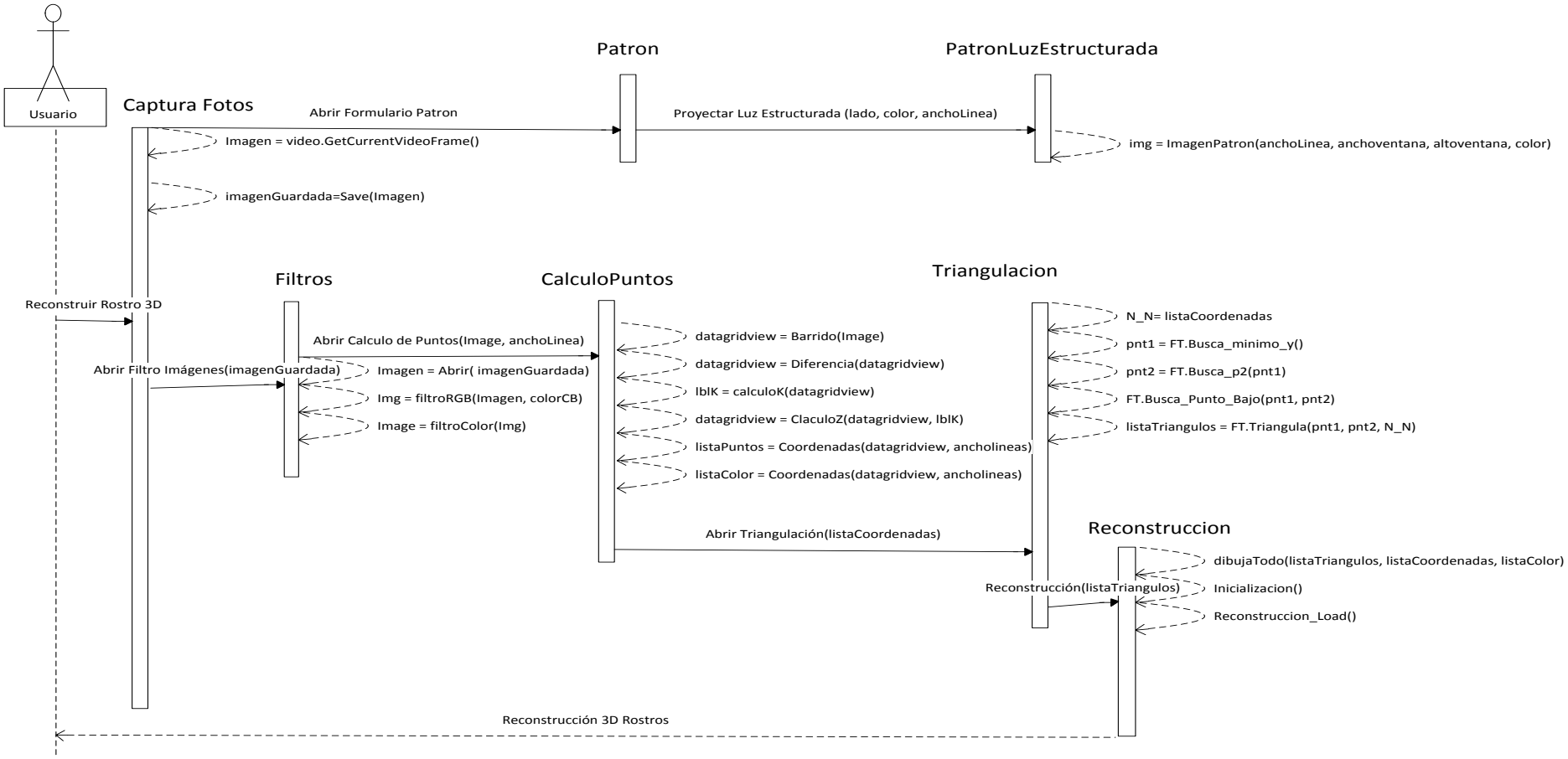
Nuestro software no está basado en un diagrama de clases específico ya que no tenemos herencias entre clases ni realizamos llamadas a la base de datos, todos los procesos que se realizan son el forma secuencial donde las funciones reciben datos y devuelven el resultado. Por tanto no tenemos un diagrama UML que interprete nuestro código fuente.

Sin embargo realizamos el diagrama de los casos de uso y una secuencia UML de procesos.

7.10.1. Diagrama de casos de uso



7.10.2. Secuencia uml de procesos



CAPÍTULO VIII

ANÁLISIS DE RESULTADOS

- En el modelo tridimensional del rostro humano que se obtuvo al final del proceso se pueden diferenciar claramente sus facciones, adicionalmente la inclusión de la textura en la imagen tridimensional nos da un modelo semejante a la realidad.
- El color verde proyectado por el patrón de color sobre el rostro humano favoreció la calidad de las fotografías en comparación con los patrones de color azul y rojo que proporcionaban muchos brillos y ruido en el rostro.
- El proceso de mallado mediante la triangulación de Delaunay proporcionó la reconstrucción tridimensional esperada y los resultados obtenidos son satisfactorios, pues se llegó a la restructuración de la cara en el espacio 3D con un buen detalle en las facciones que permitió su visualización y su correspondiente semejanza con la realidad.
- La infraestructura que utilizamos nos ayudó significativamente durante todo el proyecto, sin embargo la armadura no resultó ser estable ya que al momento del cambio del modelo producía una desconfiguración de las cámaras. El traslado de la infraestructura es relativamente fácil lo único que hay que tomar en cuenta es la forma de armar nuestra capsula para simular el cuarto oscuro.
- La toma de fotografías en un principio resultó un inconveniente pues la webcams utilizadas en ese entonces no tenían una resolución y compatibilidad adecuada, por lo que realizamos muchas pruebas que no tuvieron excelentes resultados.
- Para una toma óptima de fotografías fue necesario tener en cuenta que el obturador de las cámaras visualizan mejor una imagen al momento que empieza a correr el video, es decir tomamos las fotos

durante el proceso de abertura del obturador con lo cual evitamos brillos excesivos en la foto.

- El proyector a utilizar fue un limitante ya que la intensidad de luz que éste proyectaba era demasiado fuerte, lo que opacaba las líneas más finas del rostro.
- El objeto de calibración lo realizamos de una forma conocida y sobre todo de un material traslucido, para que éste nos ayude a definir mejor las líneas que se van a proyectar sobre el rostro y poder con éstos datos calcular la constate K y la coordenada Z.
- El cuerpo de calibración que se utilizó en un principio reflejaba mucho la luz del proyector lo que impedía tener una definición clara de los colores en la fotografía de éste objeto. Posteriormente se decidió pintar el objeto de color negro lo que tampoco ayudó a solucionar el problema de los brillos ya la pintura en si generaba más brillos aun, por tanto el material que nos favoreció fue el uso de fomix que no generó brillos con un color parecido al tono de piel.
- La proyección de un patrón de distintos colores con el blanco en el centro no nos proporcionó los resultados esperados, pues el color amarillo y azul que se reflejaba en la cara provocaba una cantidad excesiva de brillos que al momento de realizar la limpieza de ruido de la imagen éstos colores se transformaba en blanco lo cual producía errores y alto tiempo de cálculo en la captura de los datos de la imagen.
- La compatibilidad entre programas para la exportación del mallado depende mucho del uso que se le quiera dar a éste. Los programas modeladores de objetos 3D tienen un formato específico para los archivos de importación, es por ello que generamos un archivo genérico (formato ply) adecuado que sea legible para Blender, 3D MAX, MESHLAB, AUTOCIVIL.

CAPÍTULO IX

CONCLUSIONES Y RECOMENDACIONES

9.1. Conclusiones

- Con los resultados obtenidos hemos podido cumplir los objetivos que nos plantemos, la reconstrucción del rostro humano a partir de dos fotografíasdel mismo y la implementación de iluminación estructurada es todo un éxito, ya que conseguimos con un sistema de bajo costo y un software específico poder visualizar las facciones del rostro en tres dimensiones.
- El hardware y el software que se utilizó para la captura de fotografías es una estructura fácil de manejar ya que se utilizó simplemente dos cámaras, un proyector, una armadura que se asemeja a un cuanto oscuro y una computadora cuyo costo no es elevado.
- El patrón de color verde utilizado sobre la cara de una persona ayudó a optimizar el tiempo de cálculo en el procesamiento digital de imágenes, lo cual contribuyó con la obtención de las coordenadas precisas y necesarias para los posteriores cálculos matemáticos.
- La triangulación de Delaunay es el método que se utilizó para la creación de toda la malla del rostro, el mismo cumple con todas las especificaciones matemáticas que su definición conlleva, lo cual arrojó todos los datos esperados y que hicieron posible la visualización del modelo en tres dimensiones.
- Para mejorar la visualización del mallado del rostro se utilizó la librería gráfica OpenGL(API) cuyo código fuente se basa en funciones específicamente creadas para gráficos en 2d y 3d que ayudan a la construcción de escenas tridimensionales. Mediante la inclusión de ésta librería en el software se logró ver el rostro en un eje de coordenadas (X,Y,Z) con su respectiva textura y rotación, sin

tener en éste punto la necesidad de exportar las coordenadas obtenidas a un programa más especializado en moldeamiento 3D para poder ver en detalle la reconstrucción.

- El sistema está desarrollado en una arquitectura de código secuencial, existen clases pero no hay herencia de las mismas, es decir no hay una comunicación entre ellas; por lo tanto la diagramación UML no representa la estructura del código fuente de nuestro proyecto.
- Este trabajo de tesis es la pauta inicial para la inclusión de nuestra universidad en la industria de videojuegos, películas animadas, estudios médicos etc. Lastimosamente la especialidad no es conocida a nivel nacional, es por ello que el presente trabajo es una ventana para mostrar toda la potencialidad que tiene la Computación Gráfica y sus posibles aplicaciones en el futuro dentro de nuestro país.

9.2. Recomendaciones

Como recomendaciones se sugiere lo siguiente:

- El sistema de iluminación estructurada debe estar formado por una armadura que sea estable ya que los continuos movimientos al cambiar de modelo provocaban una desconfiguración en el enfoque de las cámaras.
- Las cámaras deben estar separadas de la armadura en una posición fija preferiblemente que se conecten al ordenador vía Wifi que impidan su desconfiguración para la obtención de una mejor calidad de datos.
- El valor definido para el umbral en el proceso de limpieza de la imagen debe ser un dato que pueda ser modificado con facilidad ya que éste parámetro determina la cantidad de verdes que se van a redibujar en las fotografías tomadas y la cantidad de negro que pertenece al fondo de la imagen.

- Actualmente se utilizan otras técnicas para el mallado como Marching Triangles y Ball-Pivoting que se basan en puntos pivotantes de una nueve de puntos puede ayudar a reducir el tiempo de procesamiento y la calidad dela malla.
- Procurar que la toma dela mayor cantidad de imágenes sea en una habitación totalmente oscura y que no tenga ninguna entrada de luz natural para resaltar con la el patrón de luz las facciones del rostro con mayor cantidad de detalle posible.
- Incentivar a los actuales cursantes de la carrea en Computación Gráfica a realizar proyectos orientados a modelamientos 3d, lo cual va a contribuir con la correcta difusión y presentación de la Computación Gráfica como especialidad en la Universidad Central del Ecuador.
- Incrementar el número de libros de Computación Gráfica en la biblioteca de nuestra facultad ya que actualmente no existen muchas referencias bibliográficas para ésta especialización, por lo que resulta complicado encontrar mayor información de temas referentes a nuestro proyecto y en general a la carrera.
- Fomentar la difusión y promoción enfocada a los nuevos alumnos de la Facultad de Ingeniería Ciencias Fsicas y Matematicas a optar por la carrera de Computación Grafica, ya que en el ámbito laboral se requiere este perfil profesional, tomando en cuenta que no se dispone de profesionales capacitados en esta área.

GLOSARIO DE TÉRMINOS

Bitmap: Representación de una imagen digital con color en la memoria del computador.

Brillo: Cantidad de flujo luminoso emitido.

Color: Es una percepción visual que se genera en el cerebro de los humanos y animales al interpretar las señales nerviosas que le envían los foto receptores en la retina del ojo, que a su vez interpretan y distinguen las distintas longitudes de onda que captan de la parte visible del espectro electromagnético (la luz).

Contraste: Es la diferencia relativa en intensidad entre un punto de una imagen y sus alrededores.

Coordenada: Es una pareja de objetos matemáticos, en la que se distingue un primer elemento y un segundo elemento. El par ordenado cuyo primer elemento es a y cuyo segundo elemento es b se denota como (a, b).

Eje Coordenado: Son dos ejes perpendiculares entre sí, que se cortan en el origen.

Fotografía: Es el arte y la técnica para obtener imágenes duraderas debidas a la acción de la luz.

Fuente de Luz: Es el medio mediante el cual se produce la luz.

Histograma: Es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. En el eje vertical se representan las frecuencias, y en el eje horizontal los valores de las variables,

normalmente señalando las marcas de clase, es decir, la mitad del intervalo en el que están agrupados los datos.

Iluminación: Es el conjunto de luces que se instala en un determinado lugar con la intención de iluminarlo.

Imagen: Representación figurativa de una cosa.

Malla: Tejido parecido a una red.

Modelamiento: Es el proceso de creación de una representación o imagen (el modelo) de un objeto real.

Pixel: Es la menor unidad homogénea en color que forma parte de una imagen.

Renderización: Es el termino usado para referirse al proceso de generar una imagen desde un modelo. Éste termino técnico es utilizado por los animadores o productores audiovisuales y en programas de diseño en 3D.

Resolución: Cantidad de pixeles por unidad de longitud, comúnmente pixeles por pulgada (ppp).

Triangulación: Es el uso de la trigonometría de triángulos para determinar posiciones de puntos, medidas de distancias o áreas de figuras.

Umbral: Valor simple de un pixel cuyo valor mínimo de gris está por debajo y el valor máximo de intensidad está por encima.

Webcams: Son dispositivos de captura digital de video de baja resolución que pueden tomar imágenes y transmitirlas a través de Internet.

BIBLIOGRAFÍA

Libros

1. MERY, Domingo, Visión Artificial, 1era Edición, Universidad Santiago de Chile, 2002.
2. ESCALERA HUESO, Arturo, Visión por Computador Fundamentos y Métodos, 1era Edición, Universidad Carlos III de Madrid, 2001.
3. ZISSERMAN, Andrew, HARTLEY, Richard, Multiple View Geometry in Computer Vision, 2da Edición, Australian National University, Camberra, 2003.
4. BURGE, Wilhelm, J. BURGE, Marck, Digital Image Processing and Algorithmic, 3ra Edición 564p, Austria/Springer,2008.
5. DAVID, Ebert, MUSGARVE, F. Kenton, Texturing and Modeling ,4ta Edición 687p, Amsterdam/Morgan, Kaufmann Publishers, 2003.

Artículos y publicaciones

1. NARVÁEZ, Astrid, RAMÍREZ, Esmitt, Un Escáner Simple Basado en Visión Pasiva para Reconstrucción Geométrica. Universidad Central de Venezuela, Publicada en Revista IEEE América Latina, 2012, http://www.ewh.ieee.org/reg/9/etrans/ieee/issues/vol10/vol10issue5Sept2012/10TLA5_15Narvaez.pdf
2. PÉREZ ÁLVARES, Juan Antonio, Apuntes de Fotometría II, Centro Universitario de Mérida, http://es.scribd.com/doc/36263964/102/Geometria-epipolar.
3. Cd Informativo adjunto con cámaras a utilizarse
4. Segmentación de Imágenes, http://alojamientos.us.es/gtocoma/pid/tema4.pdf
5. C. PAZ, Genaro, Procesamiento Digital de Imágenes, Textos Universitarios / Serie Docencia, http://www2.uacj.mx/Publicaciones/GeneracionImagenes/imagenesCap8.pdf.

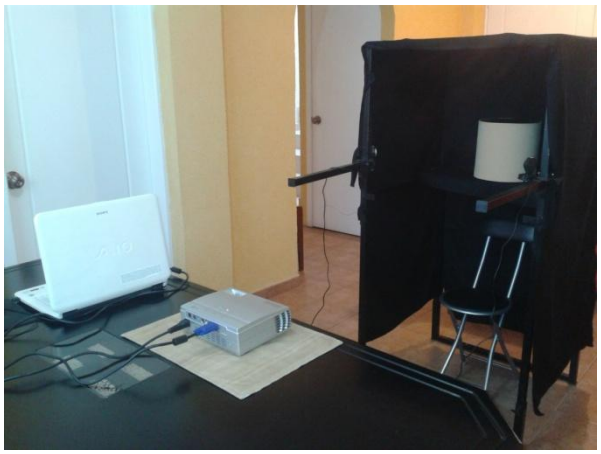
6. Código compartido por compañeros del año lectivo 2010-2011 de la materia de Visión Computacional
7. CREATIVE COMMONS, Histograma y Ajustes de Color, España, http://www.santalices.net/cuadernos/programas_cb/a_2.htm
8. Problema de Estimación y Emparejamiento de Puntos, http://iie.fing.edu.uy/investigacion/grupos/gti/cursos/egvc/material/tema-5.pdf
9. SUÁREZ BRAVO, Álvaro, Análisis de Métodos de Procesamiento de Imágenes Estereoscópicas Forestales, Universidad Complutense de Madrid, España, 2008-2009 http://eprints.ucm.es/9875/1/An%C3%A1lisis_de_M%C3%A9todos_de_Procesamiento_de_Im%C3%A1genes_Estereosc%C3%B3picas_Forestales_-_%C3%81lvaro_Su%C3%A1rez_Bravo.pdf
10. RODRÍGUEZ, Pedro A., Mallas Geométricas, sus Aplicaciones y la Paralelización de Algoritmos de Refinamiento de Mallas, 2011, http://www.ubiobio.cl/miweb/webfile/media/182/resumen/resumenpedrorodriguez.pdf
11. ORTEGA ALVARADO, Lidia, El Diagrama de Voronoi, Geometría Computacional , 1er de Ingeniería Informática, Plan 2004, http://wwwdi.ujaen.es/asignaturas/gc/tema5.pdf
12. Texturas y Superficies http://www.eui.upm.es/~fmartin/Docencia/A3D/Teoria/Tema4/Presentacion.ppt
13. MENCHACA, Neo G., Tono y Textura, 2010, http://www.slideshare.net/NoeGMenchaca/tono-y-textura
14. Modelado 3, Clase 2 Conceptos Fundamentales de la Modelación 3d, http://abc.mitreum.net/wp-content/uploads/clase2-parte1-teoria.pdf
15. RODRÍGUEZ PÉREZ, Francisco José, ALMUEDA, Vicente Tocino, Modelado en 3D y Composición de Objetos, http://gsii.usal.es/~igrafica/descargas/temas/Tema09.pdf

Páginas web

1. Computación Gráfica, http://humbertomazuera.freehosting.net/ComputacionGrafica.htm.
2. Definición de Luz , http://definicion.de/luz/
3. INFORMÁTICA MODERNA, La Cámara Web – Webcam , http://www.informaticamoderna.com/Camara_web.htm ,
4. GXZONE, Forma y Figura, http://foros.gxzone.com/110105-forma_y_figura.html
5. Triangulaciones, http://personal.us.es/almar/docencia/practicas/triangulaciones/tema4.html
6. DESARROLLOMULTIMEDIA.ES, Cómo Iluminar una escena en fotografía y Video, http://www.desarrollomultimedia.es/articulos/iluminacion.html
7. AForge.NET, http://www.aforgenet.com/
8. http://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
9. http://es.wikipedia.org/wiki/Computaci%C3%B3n_gr%C3%A1fica
10. http://es.wikipedia.org/wiki/Segmentaci%C3%B3n_%28procesamiento_de_im%C3%A1genes%29
11. http://es.wikipedia.org/wiki/Interpolaci%C3%B3n
12. http://es.wikipedia.org/wiki/OpenGL
13. http://en.wikipedia.org/wiki/PLY_%28file_format%29

ANEXOS

Anexo1: Sistema de Iluminación Estructurada



Anexo2: Fuente de luz (proyector)

Anexo3: Webcam

Anexo 4: Código de umbralización con punteros

```
“public unsafe CImagenBin(CImagenGrisIg)
{intI,U=125;
ImaBin = new Bitmap(Ig.ToBitmap());
int ancho = Ig.Ancho, alto = Ig.Alto;
this.Ib = new byte[ancho + 1, alto + 1];
Rectangle rec = new Rectangle(0, 0, ImaBin.Width, ImaBin.Height);
BitmapDatabmpD = ImaBin.LockBits(rec, ImageLockMode.ReadWrite,
ImaBin.PixelFormat);
IntPtrptr = bmpD.Scan0;
byte* RealPointer1 = (byte*)ptr.ToPointer();
int bytes = bmpD.Stride * ImaBin.Height;

for (inti = 3; i< bytes; i += 4)
        {
byteNewValue = 0;
            I = (int)(*(RealPointer1 + i - 2));
if (I < U)
            {
NewValue = 255;
RealPointer1[i - 1] = RealPointer1[i - 2] = RealPointer1[i - 3] = NewValue;
            }
else
            {
NewValue = 0;
RealPointer1[i - 1] = RealPointer1[i - 2] = RealPointer1[i - 3] = NewValue;
}
        }
ImaBin.UnlockBits(bmpD);
        }”[48]
```

Anexo 5: Código fuente captura de fotos

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

[48]Código compartido por compañeros del año lectivo 2010-2011 de la materia de Visión Computacional

```
using System.IO;
using System.Diagnostics;

using AForge.Video;
using AForge.Video.DirectShow;

namespace Tesis_Escaneo_3D
{
   public partial class CapturaFotos : Form
   {
//Lista de camaras
FilterInfoCollection videoDevices;
private Stopwatch pararVideo = null;

       // Imagenes caapturadas
       public System.Drawing.Bitmap ImagenIzq;
       public System.Drawing.Bitmap ImagenDer;

// Contador de imagenes por sesion
       int contador = 0;

       public CapturaFotos()
       {
          InitializeComponent();

          //Mostramos Lista de Camaras en los Combos
          try
          {
             // Emunerar los dispositivos de video
videoDevices = new FilterInfoCollection(FilterCategory.VideoInputDevice);

             if (videoDevices.Count == 0)
             {
                throw new Exception();
             }

             for (int i = 1, n = videoDevices.Count; i <= n; i++)
             {
                string cameraName = i + " : " + videoDevices[i - 1].Name;
cbCamaraIzq.Items.Add(cameraName);
                cbCamaraDer.Items.Add(cameraName);
             }

             // comprobando el numero de camaras
if (videoDevices.Count == 1)
             {
                cbCamaraDer.Items.Clear();
cbCamaraDer.Items.Add("Solo se encontró solo una camara");
cbCamaraDer.SelectedIndex = 0;
                cbCamaraDer.Enabled = false;
             }
             else
             {
```

```
                cbCamaraDer.SelectedIndex = 1;
            }
            cbCamaraIzq.SelectedIndex = 0;
        }
        catch
        {
cbCamaraIzq.Items.Add("No se encontraron camaras");
            cbCamaraDer.Items.Add("No se encontraron camaras");
            cbCamaraIzq.SelectedIndex = 0;
            cbCamaraDer.SelectedIndex = 0;
            cbCamaraIzq.Enabled = false;
            cbCamaraDer.Enabled = false;
        }
    }

    private void StartCamarasIzq()
    {
        // Creamos el primer video source
VideoCaptureDevice videoSource1 = new
VideoCaptureDevice(videoDevices[cbCamaraIzq.SelectedIndex].MonikerString);
videoSource1.DesiredFrameRate = 10;
        videoIzq.VideoSource = videoSource1;
        videoIzq.Start();
        parar Video = null;
    }

    private void StartCamarasDer()
    {
        // Creamos en Segundo video source
if (cbCamaraDer.Enabled == true)          {
            System.Threading.Thread.Sleep(500);

            VideoCaptureDevice videoSource2 = new
VideoCaptureDevice(videoDevices[cbCamaraDer.SelectedIndex].MonikerString);
            videoSource2.DesiredFrameRate = 10;
            videoDer.VideoSource = videoSource2;
videoDer.Start();
        }
        // Paramos el video
        parar Video = null;
    }

    private void StopCamaraIzq()
    {
        // Detiene el video de la cámara Izquierda
videoIzq.SignalToStop();
        videoIzq.WaitForStop();
    }
    private void StopCamaraDer()
{
        // Detiene el video de la cámara Derecha
videoDer.SignalToStop();
        videoDer.WaitForStop();
```

```
        }

        private void CapturaFotos_FormClosing(object sender,
FormClosingEventArgs e)
{
            // Al momento de cerra la ventana detenemos el video de las cámaras
            StopCamaraIzq();
            StopCamaraDer();
        }

        private void btnIniciarVideo_Click(object sender, EventArgs e)
        {
           // Inicia el video de la cámara Izquierda y oculta la captura(foto) anterior
            StartCamarasIzq();
            pbImagenIzq.Visible = false;
        }
        private void btnIniciarVideoDer_Click(object sender, EventArgs e)
        {
            // Inicia el video de la cámara Derecha y oculta la captura(foto) anterior
            StartCamarasDer();
            pbImagenDer.Visible = false;
        }

        private void btnPararVideo_Click(object sender, EventArgs e)
        {
            // Al momento de cerra la ventana detenemos el video de las cámaras
            StopCamaraIzq();
            StopCamaraDer();
        }

        private void btnCapturarGuardar_Click(object sender, EventArgs e)
        {
            // Guardar la imagen capturada en forma secuencial en carpeta con fecha
actual
try
            {
                DateTime Hoy = DateTime.Today;
string fecha_actual = Hoy.ToString("dd-MM-yyyy");
                contador++;

                // Captura frame actual del video de la cámara izquierda como derecha
                ImagenIzq = videoIzq.GetCurrentVideoFrame();
                ImagenDer = videoDer.GetCurrentVideoFrame();

                StopCamaraIzq();
                StopCamaraDer();

                //Guarda imagen Izquierda
                ImagenIzq.Save(@"C:\FotosTesis\" + fecha_actual + "/"+
contador.ToString() +"ImgIzq.jpg");
pbImagenIzq.Visible = true;
                pbImagenIzq.Image = ImagenIzq;
                pbImagenIzq.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
//Guarda imagen Derecha
            ImagenDer.Save(@"C:\FotosTesis\" + fecha_actual + "/" +
contador.ToString() + "ImgDer.jpg");
pbImagenDer.Visible = true;
            pbImagenDer.Image = ImagenDer;
            pbImagenDer.SizeMode = PictureBoxSizeMode.StretchImage;
         }
         catch (Exception error)
         {
            MessageBox.Show(error.Message);
         }
      }

      private void CapturaFotos_Load(object sender, EventArgs e)
{
         // Referencia al directorio de carpetas que almacenan las imágenes
capturadas
         string activeDir = @"C:\FotosTesis";

         //Fecha Actual
         DateTime Hoy = DateTime.Today;
         string fecha_actual = Hoy.ToString("dd-MM-yyyy");

         //Genera archivos de salida en la ruta y nombre de carpeta específicado
string newPath = System.IO.Path.Combine(activeDir, fecha_actual);
         //Creación del directorio
         System.IO.Directory.CreateDirectory(newPath);

         string newFileName = System.IO.Path.GetRandomFileName();
         newPath = System.IO.Path.Combine(newPath, newFileName);

// Cuenta el número de imagenes existentes en el directorio y empeza el
contador para la secuencia de fotos a guardar
{
            string[] dirs = Directory.GetFiles(@"C:\FotosTesis\"+fecha_actual,
"*.jpg");

int cantidad = dirs.Length;
            contador = Convert.ToInt32(cantidad);
            lblTotalFotos.Text = cantidad.ToString();
}
      }

      private void recortarImagenesToolStripMenuItem_Click(object sender,
EventArgs e)
{
         //RecortarImagenes recortar = new RecortarImagenes();
         //recortar.Show();
}

      private void luzEstructuradaToolStripMenuItem_Click(object sender,
EventArgs e)
```

```
        {
            Patron patron = new Patron();
            patron.Show();
        }

        private void filtroDeImagenesToolStripMenuItem_Click(object sender,
EventArgs e)
{
            //Abrir la ventana de filtros de imagen
            Filtros filtro = new Filtros();
            filtro.Show();
        }

        private void btnCapturarIzq_Click(object sender, EventArgs e)
        {
            // Guardar la imagen capturada en forma secuencial en carpeta con fecha
actual
try
            {
                DateTime Hoy = DateTime.Today;
string fecha_actual = Hoy.ToString("dd-MM-yyyy");
                contador++;

                // Captura frame actual del video de la cámara Izquierda
                ImagenIzq = videoIzq.GetCurrentVideoFrame();
                StopCamaraIzq();

                //Guarda imagen Izquierda
                ImagenIzq.Save(@"C:\FotosTesis\" + fecha_actual + "/" +
contador.ToString() + "ImgIzq.jpg");
pbImagenIzq.Visible = true;
                pbImagenIzq.Image = ImagenIzq;
                pbImagenIzq.SizeMode = PictureBoxSizeMode.StretchImage;
            }
            catch (Exception error)
            {
                MessageBox.Show(error.Message);
            }
        }

        private void btnCapturarDer_Click(object sender, EventArgs e)
{
            // Guardar la imagen capturada en forma secuencial en carpeta con fecha
actual
try
            {
                DateTime Hoy = DateTime.Today;
string fecha_actual = Hoy.ToString("dd-MM-yyyy");
                contador++;

                // Captura frame actual del video de la cámara Derecha
                ImagenDer = videoDer.GetCurrentVideoFrame();
                StopCamaraDer();
```

```
            //Guarda imagen Derecha
            ImagenDer.Save(@"C:\FotosTesis\" + fecha_actual + "/" +
contador.ToString() + "ImgDer.jpg");
pbImagenDer.Visible = true;
            pbImagenDer.Image = ImagenDer;
            pbImagenDer.SizeMode = PictureBoxSizeMode.StretchImage;
        }
        catch (Exception error)
        {
            MessageBox.Show(error.Message);
        }
    }

    private void btnIniciaVideo_Click(object sender, EventArgs e)
{
        // Inicializa video cámara izquierda
        StartCamarasIzq();
        pbImagenIzq.Visible = false;
        // Inicializa video cámara derecha
StartCamarasDer();
        pbImagenDer.Visible = false;
    }

    private void capturarPuntosToolStripMenuItem_Click(object sender,
EventArgs e)
{
        //Abrir la ventana de captura puntos
        CapturarPuntos captura = new CapturarPuntos();
        captura.Show();
}

    private void cerrarToolStripMenuItem_Click(object sender, EventArgs e)
{
        Patron patron = new Patron();
        patron.Hide();
    }
  }
}
```

Anexo 6: Código fuente Filtros

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Text;
```

```
using System.Windows.Forms;

using AForge;
using AForge.Imaging;
using AForge.Imaging.Filters;

namespace Tesis_Escaneo_3D
{
   public partial class Filtros : Form
   {
      // Declaración Variables
      public static Bitmap ImagenFDer, ImagenCDer, ImagenRDer,
ImagenRCDer;
      public static Bitmap ImagenFIzq, ImagenCIzq, ImagenRIzq, ImagenRCIzq;
      Funciones fun = new Funciones();

      public Filtros()
      {
         InitializeComponent();
      }

      #region Abrir Imagenes

      private void Abrir(System.Windows.Forms.PictureBox picture)
      {
         try
         {
            // show file open dialog
            if (abrir1.ShowDialog() == DialogResult.OK)
            {
               // load image
               Bitmap I = (Bitmap)Bitmap.FromFile(abrir1.FileName);

               // check pixel format
               if ((I.PixelFormat == PixelFormat.Format16bppGrayScale) ||
                    (Bitmap.GetPixelFormatSize(I.PixelFormat) > 32))
               {
                  MessageBox.Show("Solo soporta imagenes.", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                  // free image
                  I.Dispose();
                  I = null;
}

               /// escalo la imagen
               double r_ancho;
double r_alto;
               int ancho;
               int alto;

               r_alto = (double)picture.Height / I.Height;

               if (r_alto * I.Width < picture.Width)
```

```
                {
                    alto = picture.Height;
                    ancho = (int)(I.Width * r_alto);
                }
                else
                {
                    r_ancho = (double)picture.Width / I.Width;

                    alto = (int)(I.Height * r_ancho);
                    ancho = picture.Width;
                }

                picture.Image = I.GetThumbnailImage(ancho, alto, null, IntPtr.Zero);
            }
        }
        catch
        {
            MessageBox.Show("Fallo la carga de imagen", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void btnAbrir1_Click(object sender, EventArgs e)
{
        Abrir(pbImagenFondo);
    }

    private void btnAbrir2_Click(object sender, EventArgs e)
    {
        Abrir(pbImagenCuerpo);
}

    private void button1_Click(object sender, EventArgs e)
{
        Abrir(pbImagenRostro);
    }
    private void btnAbrirRostroColor_Click(object sender, EventArgs e)
    {
        Abrir(pbImagenRostroColor);
}

    private void btnAbrirFondo2_Click(object sender, EventArgs e)
{
        Abrir(pbImagenFondoIzq);
    }

    private void btnAbrirCuerpo2_Click(object sender, EventArgs e)
    {
        Abrir(pbImagenCuerpoIzq);
}

    private void btnAbrirRostro2_Click(object sender, EventArgs e)
{
```

```
            Abrir(pbImagenRostroIzq);
        }

        private void btnAbrirRostroColor2_Click(object sender, EventArgs e)
        {
            Abrir(pbImagenRostroColorIzq);
        }

        #endregion

        private void btnLimpiarImagen_Click(object sender, EventArgs e)
        {
          //Declaración de variables de color componentes RGB
            Color rojo = Color.FromArgb(255, 0, 0);
            Color verde = Color.FromArgb(0, 255, 0);
            Color azul = Color.FromArgb(0, 0, 255);

            Color colorCB = Color.FromArgb(0,0,0);

         //Selecc ón de color con el que se va a filtrar
if(cbRojo.Checked)
            {
                colorCB = rojo;
            }
            if (cbVerde.Checked)
            {
                colorCB = verde;
            }
            if (cbAzul.Checked)
            {
                colorCB = azul;
}

            // Imágenes del lado Izquierdo
            // Limpieza de la imágen para discriminar ruido y colores del fondo

Bitmap IfondoIzq = new Bitmap(pbImagenFondoIzq.Image);
            IfondoIzq = fun.filtroRGB(IfondoIzq, colorCB);
            pbImagenFondoIzq.Image = fun.filtroColor(IfondoIzq);

            Bitmap IcuerpoIzq = new Bitmap(pbImagenCuerpoIzq.Image);
IcuerpoIzq = fun.filtroRGB(IcuerpoIzq, colorCB);
            pbImagenCuerpoIzq.Image = fun.filtroColor(IcuerpoIzq);

Bitmap IrostroIzq = new Bitmap(pbImagenRostroIzq.Image);
IrostroIzq = fun.filtroRGB(IrostroIzq, colorCB);
            pbImagenRostroIzq.Image = fun.filtroColor(IrostroIzq);

            // Imágenes del lado Derecho
            // Limpieza de la imágen para discriminar ruido y colores del fondo

Bitmap IfondoDer = new Bitmap(pbImagenFondo.Image);
            IfondoDer = fun.filtroRGB(IfondoDer, colorCB);
```

```
            pbImagenFondo.Image = fun.filtroColor(IfondoDer);

            Bitmap IcuerpoDer = new Bitmap(pbImagenCuerpo.Image);
IcuerpoDer = fun.filtroRGB(IcuerpoDer, colorCB);
            pbImagenCuerpo.Image = fun.filtroColor(IcuerpoDer);

Bitmap IrostroDer = new Bitmap(pbImagenRostro.Image);
IrostroDer = fun.filtroRGB(IrostroDer, colorCB);
            pbImagenRostro.Image = fun.filtroColor(IrostroDer);
        }

        //Declaración de las variables de los parámetros necesarios para la
recolección de las coordenadas de color
public static int altoBarrido;
        public static int anchoLinea;
        public static int anchoDLinea;

        private void btnDatos_Click(object sender, EventArgs e)
{
            //Obtención de los datos de entrada
            altoBarrido = Convert.ToInt32(txtAltoBarrido.Text);
anchoLinea = Convert.ToInt32(txtAnchoLinea.Text);
            anchoDLinea = Convert.ToInt32(txtAnchoDLinea.Text);

// Imágenes del lado Derecho
            // Obtención de las imagenes de los PixtureBox correspondientes a las
imágenes del lado derecho
Bitmap IcuerpoDer = new Bitmap(pbImagenCuerpo.Image);
            Bitmap IfondoDer = new Bitmap(pbImagenFondo.Image);
            Bitmap IrostroDer = new Bitmap(pbImagenRostro.Image);

//Creación del datagridview que está asociado con la captura de datos del lado
derecho
            dgDatosDer.DataSource = fun.crearDataGridView();

            //Identificación de las coordenadas en pixeles segun el color selecionado
            fun.barridoDer(IfondoDer, altoBarrido, pbImagenFondo.CreateGraphics(),
anchoDLinea);
            fun.barridoDer(IcuerpoDer, altoBarrido,
pbImagenCuerpo.CreateGraphics(), anchoDLinea);
            fun.barridoDer(IrostroDer, altoBarrido, dgDatosDer,
pbImagenRostro.CreateGraphics(), anchoDLinea);

            fun.eliminarVacias(dgDatosDer);

            // Imágenes del lado Izquierdo
            // Obtención de las imagenes de los PixtureBox correspondientes a las
imágenes del lado izquierdo
            Bitmap IcuerpoIzq = new Bitmap(pbImagenCuerpoIzq.Image);
Bitmap IfondoIzq = new Bitmap(pbImagenFondoIzq.Image);
            Bitmap IrostroIzq = new Bitmap(pbImagenRostroIzq.Image);
```

```
//Creación del datagridview que está asociado con la captura de datos del lado
izquierdo
            dgDatosIzq.DataSource = fun.crearDataGridView();

            //Identificación de las coordenadas en pixeles segun el color selecionado
            fun.barridoIzq(IfondoIzq, altoBarrido,
pbImagenFondoIzq.CreateGraphics(), anchoDLinea);
            fun.barridoIzq(IcuerpoIzq, altoBarrido,
pbImagenCuerpoIzq.CreateGraphics(), anchoDLinea);
            fun.barridoIzq(IrostroIzq, altoBarrido, dgDatosIzq,
pbImagenRostroIzq.CreateGraphics(), anchoDLinea);

            //Eliminación de celdas vacías en el datagridview
            fun.eliminarVacias(dgDatosIzq);
        }

        private void pbImagenCuerpo_MouseMove(object sender, MouseEventArgs
e)
        {
            // Mostrar en etiquetas las posiciones de los pixeles
try
            {
                Point punto;
                Color color;
                Bitmap I = new Bitmap(pbImagenCuerpo.Image);
punto = pbImagenCuerpo.PointToClient(Cursor.Position);
                color = I.GetPixel(punto.X, punto.Y);

                lblX.Text = punto.X.ToString();
                lblY.Text = punto.Y.ToString();


            }
            catch //(Exception error)
            {
                //MessageBox.Show("Mensaje: " + error);
            }
        }


        private void btnCapturaPuntos_Click(object sender, EventArgs e)
        {
            // Transferencia de imágenes y valores al cálculo de Calibración y
obtención de la coordenada Z
            altoBarrido = Convert.ToInt32(txtAltoBarrido.Text);
anchoLinea = Convert.ToInt32(txtAnchoLinea.Text);
            anchoDLinea = Convert.ToInt32(txtAnchoDLinea.Text);

            ImagenFDer = (Bitmap)pbImagenFondo.Image;
            ImagenCDer = (Bitmap)pbImagenCuerpo.Image;
            ImagenRDer = (Bitmap)pbImagenRostro.Image;
            ImagenRCDer = (Bitmap)pbImagenRostroColor.Image;
```

```
            ImagenFIzq = (Bitmap)pbImagenFondoIzq.Image;
            ImagenCIzq = (Bitmap)pbImagenCuerpoIzq.Image;
            ImagenRIzq = (Bitmap)pbImagenRostroIzq.Image;
            ImagenRCIzq = (Bitmap)pbImagenRostroColorIzq.Image;

// Form que realiza todos los calculos de Calibración y Cálculo de la coordenada
Z
            CapturarPuntos captura = new CapturarPuntos();
            captura.Show();
}

        private void pbImagenRostro_MouseMove(object sender, MouseEventArgs
e)
{
            // Mostrar en etiquetas las posiciones de los pixeles
try
            {
                Point punto;
                Color color;
                Bitmap I = new Bitmap(pbImagenRostro.Image);
punto = pbImagenRostro.PointToClient(Cursor.Position);
                color = I.GetPixel(punto.X, punto.Y);

                lblX.Text = punto.X.ToString();
                lblY.Text = punto.Y.ToString();


}
            catch //(Exception error)
            {
                //MessageBox.Show("Mensaje: " + error);
            }
        }

        private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
           // Muestra y Oculta el grupo de imágenes segun el lado (Derecha o
Izquierda)
if (checkBox1.Checked)
                panel1.Visible = true;
else
                panel1.Visible = false;
        }
    }
}
```

Anexo 7: Código fuente captura puntos

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Tesis_Escaneo_3D
{
   public partial class CapturarPuntos : Form
   {
//Inicializo las variables tomando los valores obtenidos en el formulario de Filtros
      Funciones fun = new Funciones();
      Bitmap IFondoDer = Filtros.ImagenFDer;
      Bitmap ICuerpoDer = Filtros.ImagenCDer;
      Bitmap IRostroDer = Filtros.ImagenRDer;
      Bitmap IRostroCDer = Filtros.ImagenRCDer;

      Bitmap IFondoIzq = Filtros.ImagenFIzq;
      Bitmap ICuerpoIzq = Filtros.ImagenCIzq;
      Bitmap IRostroIzq = Filtros.ImagenRIzq;
      Bitmap IRostroCIzq = Filtros.ImagenRCIzq;

      int altoBarrido = Filtros.altoBarrido;
      int anchoLinea = Filtros.anchoLinea;
      int anchoDLinea = Filtros.anchoDLinea;

public static List<String> listaPuntos = new List<String>();
      public static List<String> colorPuntos = new List<String>();

public CapturarPuntos()
      {
         InitializeComponent();
      }

      private void CapturarPuntos_Load(object sender, EventArgs e)
      {
         // Imagenes lado Derecho

         // Creación de Datagridview
         dgDatosFondoDer.DataSource = fun.crearDataGridView();
         dgDatosCuerpoDer.DataSource = fun.crearDataGridView();
         dgDatosRostroDer.DataSource = fun.crearDataGridView();
         dgDatosKDer.DataSource = fun.crearDataGridView();
         dgDatosZDer.DataSource = fun.crearDataGridView();

         //Barrido de la imágen para encontrar los puntos que cumplan con los
parámetros establecidos
         fun.barridoDer(IFondoDer, altoBarrido, dgDatosFondoDer, anchoDLinea);
         fun.barridoDer(ICuerpoDer, altoBarrido, dgDatosCuerpoDer,
anchoDLinea);
         fun.barridoDer(IRostroDer, altoBarrido, dgDatosRostroDer,
anchoDLinea);

         //Eliminacion de celdas vacías de los Datagridview
```

```
        fun.eliminarVacias(dgDatosFondoDer);
        fun.eliminarVacias(dgDatosCuerpoDer);
        fun.eliminarVacias(dgDatosRostroDer);

        //Calculo de la resta de coordenadas entre el fondo y el cuerpo de
calibración
        fun.diferencia(dgDatosFondoDer, dgDatosCuerpoDer, dgDatosKDer);

        // Imagenes lado Izquierdo

        // Creación de Datagridview
        dgDatosFondoIzq.DataSource = fun.crearDataGridView();
        dgDatosCuerpoIzq.DataSource = fun.crearDataGridView();
        dgDatosRostroIzq.DataSource = fun.crearDataGridView();
        dgDatosKIzq.DataSource = fun.crearDataGridView();
        dgDatosZIzq.DataSource = fun.crearDataGridView();

        //Barrido de la imágen para encontrar los puntos que cumplan con los
parámetros establecidos
        fun.barridoIzq(IFondoIzq, altoBarrido, dgDatosFondoIzq, anchoDLinea);
        fun.barridoIzq(ICuerpoIzq, altoBarrido, dgDatosCuerpoIzq, anchoDLinea);
        fun.barridoIzq(IRostroIzq, altoBarrido, dgDatosRostroIzq, anchoDLinea);

        //Eliminacion de celdas vacías de los Datagridview
        fun.eliminarVacias(dgDatosFondoIzq);
        fun.eliminarVacias(dgDatosCuerpoIzq);
        fun.eliminarVacias(dgDatosRostroIzq);

        //Calculo de la resta de coordenadas entre el fondo y el cuerpo de
calibración
        fun.diferencia(dgDatosFondoIzq, dgDatosCuerpoIzq, dgDatosKIzq);
}

      private void btnCalcularK_Click(object sender, EventArgs e)
{
        // Imagenes lado Derecho

        //Cálculo de la constante  de calibración "K" para la obtención de Z
        lblKDer.Text = fun.calculoK(dgDatosKDer,
Convert.ToDouble(txtZCuerpo.Text)).ToString();
        fun.eliminarVacias(dgDatosKDer);

        // Imagenes lado Izquierdo

        //Cálculo de la constante  de calibración "K" para la obtención de Z
        lblKIzq.Text = fun.calculoK(dgDatosKIzq,
Convert.ToDouble(txtZCuerpo.Text)).ToString();
        fun.eliminarVacias(dgDatosKIzq);

        Double kDer = Convert.ToDouble(lblKDer.Text);
        Double kIzq = Convert.ToDouble(lblKIzq.Text);
Double K = (Math.Abs(kDer) + Math.Abs(kIzq))/2;
```

```
            lblK.Text = K.ToString();
        }

        private void btnCalcularZ_Click(object sender, EventArgs e)
{
            // Imagenes lado Derecho

            //Calculo de la resta de coordenadas entre el fondo y el rostro de
calibración
            fun.diferencia(dgDatosFondoDer, dgDatosRostroDer, dgDatosZDer);
            //Eliminacion de celdas vacías de los Datagridview
            fun.eliminarVacias(dgDatosZDer);
            //Cálculo de la coordenada Z
            fun.calculoZ(dgDatosZDer, Convert.ToDouble(lblKDer.Text));


            // Imagenes lado Izquierdo

            //Calculo de la resta de coordenadas entre el fondo y el rostro de
calibración
            fun.diferencia(dgDatosFondoIzq, dgDatosRostroIzq, dgDatosZIzq);
            //Eliminacion de celdas vacías de los Datagridview
            fun.eliminarVacias(dgDatosZIzq);
            //Cálculo de la coordenada Z
            fun.calculoZ(dgDatosZIzq, Convert.ToDouble(lblKIzq.Text));
}

        private void btnTriangular_Click(object sender, EventArgs e)
{
            //Envío de datos al form de Triangulación
            listaPuntos = fun.coordenadas(dgDatosZIzq, dgDatosZDer, anchoLinea);
            colorPuntos = fun.colorCoordenadas(dgDatosRostroIzq, IRostroCIzq,
dgDatosRostroDer, IRostroCDer, dgDatosZIzq, dgDatosZDer, anchoLinea);
            Triangulacion triangulacion = new Triangulacion();
            triangulacion.Show();
        }
    }
}
```

Anexo 8: Código fuente patrón

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

```
namespace Tesis_Escaneo_3D
{
   public partial class Patron : Form
{
       // Llamamos a las clases
       LuzEstructurada luz = new LuzEstructurada();
private PatronLuzEstructurada form = null;
       int anchoLinea = 0;
       String lado;
       Color color;

       private PatronLuzEstructurada FormInstance
       {
          get
          {
             if (form == null)
             {
                form = new PatronLuzEstructurada(lado, color, anchoLinea);
                form.Disposed += new EventHandler(form_Disposed);
             }

             return form;
          }
       }

       void form_Disposed(object sender, EventArgs e)
       {
          form = null;
       }


       public Patron()
       {
          InitializeComponent();
       }

       private void btnPatronIzq_Click(object sender, EventArgs e)
{
          lado = "Izquierda";
          color = lblColor.BackColor;
anchoLinea = Convert.ToInt32(txtAnchoLinea.Text);
PatronLuzEstructurada frm = this.FormInstance;
          frm.Text = "Luz Estructurada Izquierda";
frm.Show();
          frm.BringToFront();
       }

       private void btnPatronDer_Click(object sender, EventArgs e)
{
          lado = "Derecha";
          color = lblColor.BackColor;
anchoLinea = Convert.ToInt32(txtAnchoLinea.Text);
PatronLuzEstructurada frm = this.FormInstance;
```

```
            frm.Text = "Luz Estructurada Derecha";
frm.Show();
            frm.BringToFront();
        }

        private void Patron_Load(object sender, EventArgs e)
        {
            this.BackColor = System.Drawing.Color.Black;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnColores_Click(object sender, EventArgs e)
        {
            ColorDialog colorDlg = new ColorDialog();

            if (colorDlg.ShowDialog() == DialogResult.OK)
            {
                lblColor.BackColor = colorDlg.Color;
                lblColor.Text = colorDlg.Color.R +","+ colorDlg.Color.G + "," +
colorDlg.Color.B;
            }
        }

        private void button1_Click_1(object sender, EventArgs e)
{
            //PatronLuzEstructurada frm = new
PatronLuzEstructurada(lado,color,anchoLinea);
form.pbPatron.BackgroundImage = luz.ImagenPatronIzq(anchoLinea,
form.Width, form.Height, color);
}
    }
}
```

Anexo 9: Código fuente patrón luz estructurada

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Tesis_Escaneo_3D
{
    public partial class PatronLuzEstructurada : Form
    {
```

```
Boolean borde = false;
        LuzEstructurada luz = new LuzEstructurada();
int anchoLinea = 0;
        String lado;
        Color color;

        public PatronLuzEstructurada(String lado, Color color, int anchoLinea)
        {
            InitializeComponent();
            this.anchoLinea = anchoLinea;
            this.lado = lado;
            this.color = color;
        }

        private void PatronColor_Click(object sender, EventArgs e)
        {
            if (borde)
            {
                this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.None;
                borde = false;
            }
            else
            {
                this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.Sizable;
                borde = true;
            }
        }

        private void PatronColor_DoubleClick(object sender, EventArgs e)
        {
            this.Close();
        }

        private void PatronColor_Load(object sender, EventArgs e)
        {
            int anchoventana = Convert.ToInt32(pbPatron.Width);
            int altoventana = Convert.ToInt32(pbPatron.Height);

if(lado == "Derecha")
                pbPatron.BackgroundImage = luz.ImagenPatronDer(anchoLinea,
anchoventana, altoventana, color);

            if(lado == "Izquierda")
                pbPatron.BackgroundImage = luz.ImagenPatronIzq(anchoLinea,
anchoventana, altoventana, color);
}

        private void pbPatron_Click(object sender, EventArgs e)
        {
            if (borde)
            {
```

```
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.None;
            borde = false;
         }
         else
         {
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.Sizable;
            borde = true;
         }
      }

      private void pbPatron_DoubleClick(object sender, EventArgs e)
      {
         this.Close();
      }

      private void PatronColor_ResizeEnd(object sender, EventArgs e)
      {
         int anchoventana = Convert.ToInt32(pbPatron.Width);
         int altoventana = Convert.ToInt32(pbPatron.Height);

if (lado == "Derecha")
            pbPatron.BackgroundImage = luz.ImagenPatronDer(anchoLinea,
anchoventana, altoventana, color);

         if (lado == "Izquierda")
            pbPatron.BackgroundImage = luz.ImagenPatronIzq(anchoLinea,
anchoventana, altoventana, color);
}

      private void PatronColor_KeyPress(object sender, KeyPressEventArgs e)
      {
         int anchoventana = Convert.ToInt32(pbPatron.Width);
         int altoventana = Convert.ToInt32(pbPatron.Height);

         if (e.KeyChar.ToString() == "a")
         { pbPatron.BackgroundImage = luz.ImagenPatronIzq(anchoLinea,
anchoventana, altoventana, color); }
         if (e.KeyChar.ToString() == "d")
{ pbPatron.BackgroundImage = luz.ImagenPatronDer(anchoLinea,
anchoventana, altoventana, color); }
      }
   }
}
```

Anexo 10: Código fuente triangulación

```
using System;
using System.Collections;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Tesis_Escaneo_3D
{
   public partial class Triangulacion : Form
   {
//Declaración de las variables
       ArrayList m_alSelectedPts = new ArrayList();
       funciones_triangulacion FT;
       graficos grf = new graficos();
       int pnt1, pnt2;
public static Bitmap Imagen1;
       public static Color colorselecion;
       public static Color[] colores = new Color[100];
       public static List<String> listaTriangulos = new List<String>();

int totalPuntos = CapturarPuntos.listaPuntos.Count;

       public Triangulacion()
       {
          InitializeComponent();
          FT = new funciones_triangulacion();
       }

       private void Triangulacion_Load(object sender, EventArgs e)
       {
         //Carga datos de coordenadas almacenadas del cálculo de puntos

           String[] trianguloPuntos, colorPuntos;
           //Carga la lista de los puntos
         foreach (String puntos in CapturarPuntos.listaPuntos)
         {
            trianguloPuntos = puntos.Split(',');
            lbCoordenadas.Items.Add(puntos);

         }
         txtBuscar.Text = lbCoordenadas.Items.Count.ToString();
          // Carga la lista de los colores
         foreach (String puntos in CapturarPuntos.colorPuntos)
         {
            colorPuntos = puntos.Split(',');
lbColores.Items.Add(puntos);
         }

       }
```

```
        private void btnBuscar_Click(object sender, EventArgs e)
        {
            int i=0;
            try
            {

                FT.N_N = lbCoordenadas.Items.Count;
                String[] trianguloPuntos = new String[3];
                foreach (object lista in lbCoordenadas.Items)
                {
                    trianguloPuntos = lista.ToString().Split(',');
                    FT.N[i].x = Convert.ToDouble(trianguloPuntos[0]);
                    FT.N[i].y = Convert.ToDouble(trianguloPuntos[1]);
i++;
                }

                pnt1 = FT.Busca_minimo_y();
textBox1.Text = pnt1.ToString();
                pnt2 = FT.Busca_p2(pnt1);
textBox2.Text = pnt2.ToString();
                FT.Busca_Punto_Bajo(pnt1, pnt2);

FT.Triangula(pnt1, pnt2, FT.N_N);
                for (i = 0; i <= FT.N_T - 1; i++)
                {
                    grf.DrawLines((int)FT.N[FT.T[i].p1].x, (int)FT.N[FT.T[i].p1].y,
(int)FT.N[FT.T[i].p2].x, (int)FT.N[FT.T[i].p2].y,
pbImagenRostro.CreateGraphics());
                    grf.DrawLines((int)FT.N[FT.T[i].p1].x, (int)FT.N[FT.T[i].p1].y,
(int)FT.N[FT.T[i].p3].x, (int)FT.N[FT.T[i].p3].y,
pbImagenRostro.CreateGraphics());
                    grf.DrawLines((int)FT.N[FT.T[i].p3].x, (int)FT.N[FT.T[i].p3].y,
(int)FT.N[FT.T[i].p2].x, (int)FT.N[FT.T[i].p2].y,
pbImagenRostro.CreateGraphics());

                    lbTriangulos.Items.Add(FT.T[i].p1 + " , " + FT.T[i].p2 + " , " +
FT.T[i].p3);

                    listaTriangulos.Add(FT.T[i].p1 + " , " + FT.T[i].p2 + " , " + FT.T[i].p3);

                }

                textBox2.Text = FT.N_T.ToString();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }

        }
```

```
        private void btnGuardar_Click(object sender, EventArgs e)
        {
            if(abrir1.ShowDialog() == DialogResult.OK)
            {
                StreamWriter sw = new StreamWriter(abrir1.FileName);
                foreach (object lista in lbCoordenadas.Items)
                {
                    sw.WriteLine(lista.ToString());
                }
                sw.Close();
            }
        }

        private void pbImagenRostro_MouseUp(object sender, MouseEventArgs e)
        {
            grf.DrawSphere(e.X, e.Y, pbImagenRostro.CreateGraphics());
        }

        private void btnDibujar_Click(object sender, EventArgs e)
        {
            String[] trianguloPuntos = new String[3];
            foreach (object lista in lbCoordenadas.Items)
            {
                trianguloPuntos = lista.ToString().Split(',');
                grf.DrawSphere(Convert.ToInt32(trianguloPuntos[0]),
Convert.ToInt32(trianguloPuntos[1]), pbImagenRostro.CreateGraphics());
            }
        }

        private void btnGraficar_Click(object sender, EventArgs e)
        {
            Reconstruccion rec = new Reconstruccion();
            rec.Show();
        }

        private void btnGuardaTriangulos_Click(object sender, EventArgs e)
        {
            if (abrir1.ShowDialog() == DialogResult.OK)
            {
                StreamWriter sw = new StreamWriter(abrir1.FileName);
                foreach (object lista in lbTriangulos.Items)
                {
                    sw.WriteLine(lista.ToString());
                }
                sw.Close();
            }
        }

        private void btnGuardaCoordenadasColor_Click(object sender, EventArgs
e)
        {
            if (abrir1.ShowDialog() == DialogResult.OK)
            {
```

```
            StreamWriter sw = new StreamWriter(abrir1.FileName);
            foreach (object lista in lbColores.Items)
            {
               sw.WriteLine(lista.ToString());
            }
            sw.Close();
}
      }
   }
}
```

Anexo 11: Código fuente reconstrucción

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using Tao.FreeGlut;
using Tao.OpenGl;
using Tao.Platform.Windows;

namespace Tesis_Escaneo_3D
{
   public partial class Reconstruccion : Form
   {
      //Declararo e Inicializo las variables para la reconstruccion del rostro
private IntPtr hDC;
      private IntPtr hRC;
      double angulo = 0;
      double time = 0;
      bool signo;

public Reconstruccion()
      {
         InitializeComponent();

         // Declaracion de variables para mover la malla usando las  teclas
this.KeyDown += new KeyEventHandler(Reconstruccion_KeyDown);
         this.KeyPress += new
KeyPressEventHandler(Reconstruccion_KeyPress);
}

      // FUNCIONES

      // Función generada para crear y ubicar la cámara dentro de la escena
public void Camara()
```

```
        {
            Gl.glMatrixMode(Gl.GL_MODELVIEW);
            Gl.glLoadIdentity();
Glu.gluLookAt(20, 20, 50, 0, 0, 0, 0, -1, 0);
        }

        //Función donde se inicializan los parametros de la escena, perspectiva,
profundidad.
public void Inicializacion()
        {
            Gl.glShadeModel(Gl.GL_SMOOTH);                          // Enable
Smooth Shading
            Gl.glHint(Gl.GL_PERSPECTIVE_CORRECTION_HINT, Gl.GL_NICEST);
// Really Nice Perspective Calculations
            Gl.glMatrixMode(Gl.GL_PROJECTION);
            Gl.glViewport(0, 0, Width, Height);
            Gl.glLoadIdentity();
            Glu.gluPerspective(60, (float)Width / (float)Height, 1, 500);
}

        //Función para dibujar elementos en función del tiempo
        public void EmpiezaTodo()
        {
            Camara();
            if (signo == true)
                dibujaTodo(angulo + time);
            else if (signo == false)
                dibujaTodo(angulo - time);
        }


        public void dibujaTodo(double anguloGeneral)
        {
          // Definimos el angulo de rotación
Gl.glRotated(anguloGeneral, 0, 1, 0);
            Gl.glPushMatrix();

//Empezamos a graficar las líneas y los colores de las coordenadas de los ejes
Gl.glBegin(Gl.GL_LINES);
                Gl.glColor3f(1.0f, 1.0f, 0.0f);

                Gl.glVertex3f(-10.0f, 0.0f, 0);
                Gl.glVertex3f(10.0f, 0.0f, 0);

                Gl.glVertex3f(0.0f, 10.0f, 0);
                Gl.glVertex3f(0.0f, -10.0f, 0);

                Gl.glVertex3f(0.0f, 0.0f, -10);
                Gl.glVertex3f(0.0f, 0.0f, 10);
Gl.glEnd();

            //Empezamos a grafica las líneas y los colores de las coordenadas
generadas de la triangulación
```

```
Gl.glBegin(Gl.GL_TRIANGLES);

        String[] trianguloPuntos = new String[3];
        String[] verticesPuntos = new String[3];
        String[] verticesColor = new String[3];
        foreach (String triangulo in Triangulacion.listaTriangulos)
        {
            int t1,t2,t3;
            String vertices, colores;
            trianguloPuntos = triangulo.Split(',');
            t1 = Convert.ToInt16(trianguloPuntos[0]);
            t2 = Convert.ToInt16(trianguloPuntos[1]);
            t3 = Convert.ToInt16(trianguloPuntos[2]);

            // T1 - T2
vertices = CapturarPuntos.listaPuntos[t1];
            colores = CapturarPuntos.colorPuntos[t1];
            verticesPuntos = vertices.Split(',');
verticesColor = colores.Split(',');

            Gl.glColor3f(Convert.ToInt16(verticesColor[3]) / 255f,
Convert.ToInt16(verticesColor[4]) / 255f, Convert.ToInt16(verticesColor[5]) /
255f);
Gl.glVertex3f(Convert.ToInt32(verticesPuntos[0]) / 10f,
Convert.ToInt32(verticesPuntos[1]) / 10f, Convert.ToInt32(verticesPuntos[2]) /
10f);

            vertices = CapturarPuntos.listaPuntos[t2];
            colores = CapturarPuntos.colorPuntos[t2];
            verticesPuntos = vertices.Split(',');
verticesColor = colores.Split(',');

            Gl.glColor3f(Convert.ToInt16(verticesColor[3]) / 255f,
Convert.ToInt16(verticesColor[4]) / 255f, Convert.ToInt16(verticesColor[5]) /
255f);
//Gl.glColor3f(0.0f, 0.0f, Convert.ToInt16(verticesPuntos[2]) / 255f);
            Gl.glVertex3f(Convert.ToInt16(verticesPuntos[0]) / 10,
Convert.ToInt16(verticesPuntos[1]) / 10, Convert.ToInt16(verticesPuntos[2]) / 10);

            vertices = CapturarPuntos.listaPuntos[t3];
            colores = CapturarPuntos.colorPuntos[t3];
            verticesPuntos = vertices.Split(',');
verticesColor = colores.Split(',');

            Gl.glColor3f(Convert.ToInt16(verticesColor[3]) / 255f,
Convert.ToInt16(verticesColor[4]) / 255f, Convert.ToInt16(verticesColor[5]) /
255f);
Gl.glVertex3f(Convert.ToInt16(verticesPuntos[0]) / 10,
Convert.ToInt16(verticesPuntos[1]) / 10, Convert.ToInt16(verticesPuntos[2]) / 10);

}

        Gl.glEnd();
```

```
            Gl.glPopMatrix();
        }

        private void Reconstruccion_Load(object sender, EventArgs e)
        {
            int pixelFormat;
            Gdi.PIXELFORMATDESCRIPTOR pfd = new
Gdi.PIXELFORMATDESCRIPTOR();
            pfd.nVersion = 1;
            pfd.dwFlags = Gdi.PFD_DRAW_TO_WINDOW |
Gdi.PFD_SUPPORT_OPENGL | Gdi.PFD_DOUBLEBUFFER;
            pfd.iPixelType = Gdi.PFD_TYPE_RGBA;
            pfd.iLayerType = Gdi.PFD_MAIN_PLANE;
pfd.cColorBits = 24;
            pfd.cDepthBits = 24;

            // Se crea la ventana
            hDC = User.GetDC(this.Handle);//primero se crea el hdc
            // Luego se utiliza en el choosePixelFormat
pixelFormat = Gdi.ChoosePixelFormat(hDC, ref pfd);
            Gdi.SetPixelFormat(hDC, pixelFormat, ref pfd);
            hRC = Wgl.wglCreateContext(hDC);
            Wgl.wglMakeCurrent(hDC, hRC);
            Gl.glEnable(Gl.GL_DEPTH_TEST);
// Proyeccion
            Inicializacion();
            Glut.glutInit();
            Gl.glClearColor(0, 0, 0, 0.5f);
}

        private void timer1_Tick(object sender, EventArgs e)
{
            // Controlamos el ángulo de giro en funcion del tiempo generado por el
timer
Gl.glClear(Gl.GL_COLOR_BUFFER_BIT | Gl.GL_DEPTH_BUFFER_BIT);
            Thread.Sleep(0);
            if (time < 10)
            {
                time = time + 5;
            }
            else if (time >= 10)
            {
                Thread.Sleep(0);
            }
            EmpiezaTodo();
            Gdi.SwapBuffers(hDC);
        }

        private void Reconstruccion_FormClosed(object sender,
FormClosedEventArgs e)
        {
            Wgl.wglMakeCurrent(hDC, IntPtr.Zero);
            Wgl.wglDeleteContext(hDC);
```

```
        }

        private void Reconstruccion_KeyPress(object sender, KeyPressEventArgs
e)
        {
            if (e.KeyChar == 'n')
            {
                if (time == 10)
                {
                    angulo = angulo + time;
                    time = 0;
                    signo = true;
                }
            }
            if (e.KeyChar == 'm')
            {
                if (time == 10)
                {
                    angulo = angulo - time;
                    time = 0;
                    signo = false;
                }
            }
        }

        private void Reconstruccion_KeyDown(object sender, KeyEventArgs e)
{
            // Giramos la escena usando las teclas de dirección Izquierda y Derecha
//double angulo = 0;
            if (e.KeyCode == Keys.Left)
            {
                if (time == 10)
                {
                    angulo = angulo + time;
                    time = 0;
                    signo = true;
                }
            }
            if (e.KeyCode == Keys.Right)
            {
                if (time == 10)
                {
                    angulo = angulo - time;
time = 0;
                    signo = false;
                }
            }
        }
    }
}
```

Anexo 12: Clase funciones

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.ComponentModel;
using System.Windows.Forms;

namespace Tesis_Escaneo_3D
{
   class Funciones
   {
      graficos grf = new graficos();

public Funciones()
{

      }

      public Bitmap filtroRGB(Bitmap I, Color color)
{
         // Segementar por RGB
         Color cr, cb, cg, colorPixel;
int alto = I.Height;
         int ancho = I.Width;
         Bitmap imgrgb = new Bitmap(ancho, alto);

for (int u = 0; u < ancho; u++)
{
            for (int v = 0; v < alto; v++)
{
               colorPixel = I.GetPixel(u, v);
               cr = Color.FromArgb(colorPixel.R, 0, 0);
               cg = Color.FromArgb(0, colorPixel.G, 0);
               cb = Color.FromArgb(0, 0, colorPixel.B);
if (color.R==255)
                  imgrgb.SetPixel(u, v, cr);

               else if (color.G==255)
imgrgb.SetPixel(u, v, cg);

else if (color.B==255)
                  imgrgb.SetPixel(u, v, cb);

               else
                  imgrgb.SetPixel(u, v, Color.FromArgb(0, 0, 0));
            }
```

```
            }
            return imgrgb;
        }

        public Bitmap filtroColor(Bitmap I)
        {
            Color color, cs;
            int alto = I.Height;
            int ancho = I.Width;
            Bitmap imgRGB = new Bitmap(ancho, alto);

for (int y = 0; y < alto; y ++) // alto de la imagen
            {
                for (int x = 0; x < ancho; x ++) // ancho de la imagen
                {
                    color = I.GetPixel(x, y);

                    if (color.R <= 20 && color.G <= 20 && color.B <= 20) // diferente de
negro
                    {
                        cs = Color.FromArgb(0, 0, 0);
                        imgRGB.SetPixel(x, y, cs);
                    }
                    else if (color.R >= 200 && color.G >= 200 && color.B >= 200) //
diferente de blanco
                    {
                        cs = Color.FromArgb(0, 0, 0);
                        imgRGB.SetPixel(x, y, cs);
                    }
                    else
{
                        if (color.R > color.G && color.R >= color.B)
{
                            cs = Color.FromArgb(255, 0, 0);
                            imgRGB.SetPixel(x, y, cs);
}

                        else if (color.G > color.R && color.G >= color.B && color.G >=70)
{
                            cs = Color.FromArgb(0, 255, 0);
                            imgRGB.SetPixel(x, y, cs);
}

                        else if (color.B > color.R && color.B >= color.G)
{
                            cs = Color.FromArgb(0, 0, 255);
                            imgRGB.SetPixel(x, y, cs);
                        }
                        else
                        {
                            cs = Color.FromArgb(0, 0, 0);
                            imgRGB.SetPixel(x, y, cs);
                        }
```

```
                }
            }
        }
        return imgRGB;
    }

    public void barridoDer(Bitmap I, int espacioBarrido,
System.Windows.Forms.DataGridView datos, Graphics grafico, int anchoDLinea)
{
        int contx = 0, conty = 0, distancia=0;
        int alto = I.Height;
        int ancho = I.Width;
        Boolean bandera = false;
Color color= Color.FromArgb(0, 0, 0), anterior= Color.FromArgb(0, 0, 0), negro =
Color.FromArgb(0, 0, 0);

        for (int v = 0; v < alto; v = v + espacioBarrido)
        {
            for (int u = 0; u < ancho; u ++)
            {
                color = I.GetPixel(u, v);
if (color != anterior && color != negro && distancia > anchoDLinea)
                {
                    datos.Rows[contx].Cells[0].Value = v.ToString();
                    datos.Rows[contx].Cells[conty+1].Value = u.ToString();
grf.DrawCross(u, v, grafico);
anterior = color;
                    conty++;
                    bandera = true;
                    distancia = 0;
                }
                else if (color == negro)
anterior = negro;

                distancia++;
             }
            if (bandera)
            contx++;
            conty = 0;
        }
     }

    public void barridoDer(Bitmap I, int espacioBarrido,
System.Windows.Forms.DataGridView datos, int anchoDLinea)
{
        int contx = 0, conty = 0, distancia=0;
        int alto = I.Height;
        int ancho = I.Width;
        Boolean bandera = false;
Color color = Color.FromArgb(0, 0, 0), anterior = Color.FromArgb(0, 0, 0), negro
= Color.FromArgb(0, 0, 0);

        for (int v = 0; v < alto; v = v + espacioBarrido)
```

```
        {
            for (int u = 0; u < ancho; u++)
            {
                color = I.GetPixel(u, v);
if (color != anterior && color != negro && distancia > anchoDLinea)
                {
                    datos.Rows[contx].Cells[0].Value = v.ToString();
                    datos.Rows[contx].Cells[conty + 1].Value = u.ToString();
                    anterior = color;
                    conty++;
                    bandera = true;
                    distancia = 0;
                }
                else if (color == negro)
anterior = negro;

                distancia++;
            }
            if (bandera)
contx++;
            conty = 0;
        }
    }

    public void barridoDer(Bitmap I, int espacioBarrido, Graphics grafico, int
anchoDLinea)
    {
        int contx = 0, conty = 0, distancia=0;
        int alto = I.Height;
        int ancho = I.Width;
        Boolean bandera = false;
Color color = Color.FromArgb(0, 0, 0), anterior = Color.FromArgb(0, 0, 0), negro
= Color.FromArgb(0, 0, 0);

        for (int v = 0; v < alto; v = v + espacioBarrido)
        {
            for (int u = 0; u < ancho; u++)
            {
                color = I.GetPixel(u, v);
                if (color != anterior && color != negro && distancia > anchoDLinea)
                {
                    grf.DrawCross(u, v, grafico);
anterior = color;
                    conty++;
                    bandera = true;
                    distancia = 0;
                }
                else if (color == negro)
anterior = negro;

                distancia++;
            }
            if (bandera)
```

```
                contx++;
            conty = 0;
        }
    }

    public void barridoIzq(Bitmap I, int espacioBarrido,
System.Windows.Forms.DataGridView datos, Graphics grafico, int anchoDLinea)
{
        int contx = 0, conty = 0, distancia = 0;
        int alto = I.Height;
        int ancho = I.Width;
        Boolean bandera = false;
Color color = Color.FromArgb(0, 0, 0), anterior = Color.FromArgb(0, 0, 0), negro
= Color.FromArgb(0, 0, 0);

        for (int v = 0; v < alto; v = v + espacioBarrido)
        {
            for (int u = ancho-1; u >0; u--)
            {
                color = I.GetPixel(u, v);
if (color != anterior && color != negro && distancia > anchoDLinea)
                {
                    datos.Rows[contx].Cells[0].Value = v.ToString();
                    datos.Rows[contx].Cells[conty + 1].Value = u.ToString();
grf.DrawCross(u, v, grafico);
anterior = color;
                    conty++;
                    bandera = true;
                    distancia = 0;
                }
                else if (color == negro)
anterior = negro;

                distancia++;
            }
            if (bandera)
                contx++;
            conty = 0;
        }
    }

    public void barridoIzq(Bitmap I, int espacioBarrido,
System.Windows.Forms.DataGridView datos, int anchoDLinea)
{
        int contx = 0, conty = 0, distancia = 0;
        int alto = I.Height;
        int ancho = I.Width;
        Boolean bandera = false;
Color color = Color.FromArgb(0, 0, 0), anterior = Color.FromArgb(0, 0, 0), negro
= Color.FromArgb(0, 0, 0);

        for (int v = 0; v < alto; v = v + espacioBarrido)
        {
```

```
            for (int u = ancho-1; u > 0; u--)
            {
                color = I.GetPixel(u, v);
if (color != anterior && color != negro && distancia > anchoDLinea)
                {
                    datos.Rows[contx].Cells[0].Value = v.ToString();
                    datos.Rows[contx].Cells[conty + 1].Value = u.ToString();
                    anterior = color;
                    conty++;
                    bandera = true;
                    distancia = 0;
                }
                else if (color == negro)
anterior = negro;

                    distancia++;
                }
                if (bandera)
                    contx++;
                conty = 0;
            }
        }

        public void barridoIzq(Bitmap I, int espacioBarrido, Graphics grafico, int
anchoDLinea)
        {
            int contx = 0, conty = 0, distancia = 0;
            int alto = I.Height;
int ancho = I.Width;
            Boolean bandera = false;
Color color = Color.FromArgb(0, 0, 0), anterior = Color.FromArgb(0, 0, 0), negro
= Color.FromArgb(0, 0, 0);

            for (int v = 0; v < alto; v = v + espacioBarrido)
            {
                for (int u = ancho-1; u > 0; u--)
                {
                    color = I.GetPixel(u, v);
                    if (color != anterior && color != negro && distancia > anchoDLinea)
                    {
                        grf.DrawCross(u, v, grafico);
anterior = color;
                        conty++;
                        bandera = true;
                        distancia = 0;
                    }
                    else if (color == negro)
anterior = negro;

                    distancia++;
                }
                if (bandera)
contx++;
```

```
            conty = 0;
        }
    }

    public DataTable crearDataGridView()
{
        DataTable aTbl = new DataTable("Puntos");
DataColumn aClmn;
        int ip = 200;
        aClmn = new DataColumn("Y");
        aClmn.DataType = System.Type.GetType("System.Int32");
        aTbl.Columns.Add(aClmn);

        for (int i = 1; i <= ip; i++)
        {
            aClmn = new DataColumn("X" + (i));
            aClmn.DataType = System.Type.GetType("System.Int32");
aTbl.Columns.Add(aClmn);

        }
        // LLena los datos
for (int i = 0; i <= 500; i++)
        {
            DataRow aRow;
            aRow = aTbl.NewRow();
            aTbl.Rows.Add(aRow);
        }
        return aTbl;
    }

    public void eliminarVacias(System.Windows.Forms.DataGridView datos)
    {
        // elimina filas vacias
        for (int n = datos.Rows.Count - 2; n >= 0; n += -1)
        {
            System.Windows.Forms.DataGridViewRow row = datos.Rows[n];
            if (datos.Rows[n].Cells[0].Value == DBNull.Value)
                datos.Rows.Remove(row);
        }

    }

    public void diferencia(System.Windows.Forms.DataGridView datos1,
System.Windows.Forms.DataGridView datos2,
System.Windows.Forms.DataGridView respuesta)
{
        int resta, valorF, valorC;

        for (int v = 0; v < datos1.RowCount; v++) // alto de la imagen
        {
            for (int u = 1; u < datos1.ColumnCount; u++) // ancho de la imagen
{
                if (u < datos2.ColumnCount && v < datos2.RowCount-1)
```

```
                {
                    if (datos1.Rows[v].Cells[u].Value != DBNull.Value &&
datos2.Rows[v].Cells[u].Value != DBNull.Value)
{
                        valorF = Convert.ToInt32(datos1.Rows[v].Cells[u].Value);
valorC = Convert.ToInt32(datos2.Rows[v].Cells[u].Value);
                        resta = valorF - valorC;
respuesta.Rows[v].Cells[0].Value = datos1.Rows[v].Cells[0].Value;
                        respuesta.Rows[v].Cells[u].Value = resta.ToString();
                    }
                }
            }
        }
    }

    public double calculoK(System.Windows.Forms.DataGridView datos, double
zcuerpo)
{
        double diferencia, k, suma = 0, resul;
int total = 0;

        for (int v = 0; v < datos.RowCount - 1; v++) // alto
{
            for (int u = 1; u < datos.ColumnCount - 1; u++) // ancho
{
                if (datos.Rows[v].Cells[u].Value != DBNull.Value)
                {
                    diferencia = Convert.ToDouble(datos.Rows[v].Cells[u].Value);
                    k = diferencia / zcuerpo;
                    suma = suma + k;

total++;
                }
            }
        }
        resul = suma / total;
return resul;
    }

    public void calculoZ(System.Windows.Forms.DataGridView datos, double k)
    {
        double diferencia, z;

        for (int v = 0; v < datos.RowCount - 1; v++) // alto
{
            for (int u = 1; u < datos.ColumnCount - 1; u++) // ancho
{
                if (datos.Rows[v].Cells[u].Value != DBNull.Value)
{
                    diferencia = Convert.ToDouble(datos.Rows[v].Cells[u].Value);
z = Math.Round((diferencia / k),4);
                    datos.Rows[v].Cells[0].Value = datos.Rows[v].Cells[0].Value;
                    datos.Rows[v].Cells[u].Value = z;
```

```
                }
            }
        }
    }

    public List<String> coordenadas(System.Windows.Forms.DataGridView
datosZIzq, System.Windows.Forms.DataGridView datosZDer, int anchoLinea)
    {
        List<String> listatriangulos = new List<String>();
        String x, y, z;
        int cont = 200;
for (int u = 1; u < datosZIzq.ColumnCount - 1; u++) // ancho
{
            for (int v = 0; v < datosZIzq.RowCount - 1; v++) // alto
            {
                if (datosZIzq.Rows[v].Cells[u].Value != DBNull.Value)
                {
                    x = cont.ToString();
                    y = datosZIzq.Rows[v].Cells[0].Value.ToString();
                    z = datosZIzq.Rows[v].Cells[u].Value.ToString();
listatriangulos.Add(x + "," + y + "," + z);
}
            }
            cont = cont - anchoLinea;
        }

        cont = 210;
        for (int u = 1; u < datosZDer.ColumnCount - 1; u++) // ancho
        {
            for (int v = 0; v < datosZDer.RowCount - 1; v++) // alto
            {
              if (datosZDer.Rows[v].Cells[u].Value != DBNull.Value)
              {
                  x = cont.ToString();
                  y = datosZDer.Rows[v].Cells[0].Value.ToString();
                  z = datosZDer.Rows[v].Cells[u].Value.ToString();
listatriangulos.Add(x + "," + y + "," + z);
}
           }
            cont = cont + anchoLinea;
        }
        return listatriangulos;
    }

    public List<String>
colorCoordenadas(System.Windows.Forms.DataGridView datosIzq, Bitmap IIzq,
System.Windows.Forms.DataGridView datosDer, Bitmap IDer,
System.Windows.Forms.DataGridView datosZIzq,
System.Windows.Forms.DataGridView datosZDer, int anchoLinea)
    {
        List<String> listacolor = new List<String>();
Color color;
        int x, y,z;
```

```
int cont = 200;

        for (int u = 1; u < datosIzq.ColumnCount - 1; u++) // ancho
        {
            for (int v = 0; v < datosIzq.RowCount - 1; v++) // alto
            {
                if (datosIzq.Rows[v].Cells[u].Value != DBNull.Value)
                {
                    y = Convert.ToInt32(datosIzq.Rows[v].Cells[0].Value);
                    x = Convert.ToInt32(datosIzq.Rows[v].Cells[u].Value);
color = IIzq.GetPixel(x, y);

x = cont;
                    y = Convert.ToInt32(datosIzq.Rows[v].Cells[0].Value);
                    z = Convert.ToInt32(datosZIzq.Rows[v].Cells[u].Value);

listacolor.Add(x + "," + y + "," + z + "," +color.R + "," + color.G + "," + color.B);
}
            }
            cont = cont - anchoLinea;
        }

        cont = 210;
        for (int u = 1; u < datosDer.ColumnCount - 1; u++) // ancho
        {
            for (int v = 0; v < datosDer.RowCount - 1; v++) // alto
            {
                if (datosDer.Rows[v].Cells[u].Value != DBNull.Value)
                {
                    y = Convert.ToInt32(datosDer.Rows[v].Cells[0].Value);
                    x = Convert.ToInt32(datosDer.Rows[v].Cells[u].Value);
color = IDer.GetPixel(x, y);

x = cont;
                    y = Convert.ToInt32(datosDer.Rows[v].Cells[0].Value);
                    z = Convert.ToInt32(datosZDer.Rows[v].Cells[u].Value);

listacolor.Add(x + "," + y + "," + z + "," + color.R + "," + color.G + "," + color.B);
}
            }
            cont = cont + anchoLinea;
        }
        return listacolor;
}
   }
}
```

Anexo 13: Claseluz estructurada

```
using System;
using System.Collections.Generic;
```

```
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace Tesis_Escaneo_3D
{
   class LuzEstructurada
{
       public LuzEstructurada()
       {

       }

       public Bitmap ImagenPatronIzq(int anchoLinea, int anchoVentana, int
altoVentana, Color color)
       {
          Bitmap imagen = new Bitmap(anchoVentana, altoVentana);
int u, v;
          int mitad = Convert.ToInt32(anchoVentana / 2);
          Boolean blanco = true;

          for (u = 0; u < anchoVentana; u++)
          {
             if (u <= mitad)
             {
                for (v = 0; v < altoVentana; v++)
                {
                   if (u < anchoVentana)
                      imagen.SetPixel(u, v, Color.Black);
}
             }
             else if (u > mitad && blanco)
             {
                for (int i = 1; i <= anchoLinea; i++)
{
                   for (v = 0; v < altoVentana; v++)
                   {
                      if (u < anchoVentana)
                         imagen.SetPixel(u, v, color);
                   }
                   if (u < anchoVentana)
                      u++;
                }
                blanco = false;
}
             else if (u > mitad && !blanco)
             {
                for (int i = 1; i <= anchoLinea; i++)
{
                   for (v = 0; v < altoVentana; v++)
                   {
                      if (u < anchoVentana)
```

```
                    imagen.SetPixel(u, v, Color.Black);
                }
                if (u < anchoVentana)
                    u++;
            }
            blanco = true;
}
        }
        return imagen;
    }

    public Bitmap ImagenPatronDer(int anchoLinea, int anchoVentana, int
altoVentana, Color color)
    {
        Bitmap imagen = new Bitmap(anchoVentana, altoVentana);
int u, v;
        int mitad = Convert.ToInt32(anchoVentana / 2);
        Boolean blanco = true;

        for (u = 0; u < anchoVentana; u++)
        {
            if (u >= mitad)
            {
                for (v = 0; v < altoVentana; v++)
                {
                    if (u < anchoVentana)
                        imagen.SetPixel(u, v, Color.Black);
}
            }
            else if (u < mitad && blanco)
            {
                for (int i = 1; i <= anchoLinea; i++)
{
                    for (v = 0; v < altoVentana; v++)
                    {
                        if (u < anchoVentana)
                            imagen.SetPixel(u, v, color);
                    }
                    if (u < anchoVentana)
                        u++;
                }
                blanco = false;
}
            else if (u < mitad && !blanco)
            {
                for (int i = 1; i <= anchoLinea; i++)
{
                    for (v = 0; v < altoVentana; v++)
                    {
                        if (u < anchoVentana)
                            imagen.SetPixel(u, v, Color.Black);
                    }
                    if (u < anchoVentana)
```

```
                    u++;
                }
                blanco = true;
}
            }
            return imagen;
        }
    }
}
```

Anexo 14: Clase gráficos

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace Tesis_Escaneo_3D
{
    public class graficos
    {
      public graficos()
{

        }

        public void DrawCross(int x, int y, Graphics gfx)
        {
            Point startPt = new Point(x - 2, y);
            Point endPt = new Point(x + 2, y);
            gfx.DrawLine(Pens.Red, startPt, endPt);
            startPt = new Point(x, y - 2);
            endPt = new Point(x, y + 2);
            gfx.DrawLine(Pens.Red, startPt, endPt);
        }
        public void DrawRectangle(int x, int y, int ancho, int alto, Graphics gfx)
{
          gfx.DrawRectangle(Pens.Red, x, y, ancho, alto);
}

        public void DrawSphere(int x, int y, Graphics gfx)
        {
            gfx.DrawEllipse(Pens.White, x - 2, y - 2, 4, 4);
        }

        public void DrawLines(int x1, int y1, int x2, int y2, Graphics gfx)
        {
            Point startPt = new Point(x1, y1);
```

```
            Point endPt = new Point(x2, y2);
            gfx.DrawLine(Pens.Blue, startPt, endPt);
}
   }
}
```

Anexo 15: Clase Funciones Triangulación

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;


namespace Tesis_Escaneo_3D
{
   public class funciones_triangulacion
   {
      public funciones_triangulacion()
{

      }

      public struct nodo
      {
         public double x;
         public double y;
      }

      public struct triangulo
      {
         public int p1;
         public int p2;
         public int p3;

      }
      public struct arista
      {
         public int l1;
         public int l2;

      }

      public nodo[] N = new nodo[1000000];
```

```
        public triangulo [] T = new triangulo[1000000];
        public arista [] A = new arista[1000000];

        public int N_N, N_I, N_T;

        public double Det(int p1, int p2, int p3) {
double determinante = 0;
            determinante = N[p2].x * N[p3].y +
                           N[p3].x * N[p1].y +
                           N[p1].x * N[p2].y -
                           N[p3].x * N[p2].y -
                           N[p1].x * N[p3].y -
N[p2].x * N[p1].y;
            return determinante;
        }
        public double Angulo(int p1, int p2, int q) {
            nodo v1, v2;
            double a, b, anguloX;
            v1.x = N[p1].x - N[q].x;
            v2.x = N[p2].x - N[q].x;
v1.y = N[p1].y - N[q].y;
            v2.y = N[p2].y - N[q].y;
a = Math.Pow(Math.Pow(v1.x, 2) + Math.Pow(v1.y, 2), 0.5);
            b = Math.Pow(Math.Pow(v2.x, 2) + Math.Pow(v2.y, 2), 0.5);
anguloX = Math.Acos((v1.x * v2.x + v1.y * v2.y) / (a*b));
return anguloX;
        }

        public Boolean Busca_A(int p1, int p2) {
int k = 0;
            Boolean encuentra = false;
k = 0;
            while(encuentra==false && k<=N_I)
            {
                if (p1 == A[k].l1 && p2 == A[k].l2)
                {
                    encuentra = true;
                }
                if (p1 == A[k].l1 && p2 == A[k].l1)
{
                    encuentra = true;
                }
                k = k + 1;
            }
            return encuentra;
}
        public int Encuentra_punto(int p1, int p2, int p) {
            double angl, ang, aux;
            int i, k;
            aux = Det(p1, p2, p);
            k = -1;
            if (Busca_A(p1, p2) == false)
            {
```

```
            ang = 0;
            for( i=0; i<=N_N-1; i++)
            {
               if (i != p1 && i != p2 && i != p)
               {
                  if ((aux * Det(p1, p2, i)) < 0)
                  {
                     angl = Angulo(p1,p2,i);
                     if (angl > ang)
                     {
                        ang = angl;
                        k = i;
                     }
                  }
               }
            }
         }
         if(k != -1)
         {
            A[N_I].l1= p1;
            A[N_I].l2= p2;
            N_I= N_I + 1;
         }
         return k;
      }

      public void Triangula(int p1,int p2,int r)
{
         int p3=0;
         p3 = Encuentra_punto(p1,p2,r);
if(p3 != -1)
         {
            T[N_T].p1 = p1;
            T[N_T].p2 = p2;
            T[N_T].p3 = p3;
            N_T = N_T + 1;
            Triangula(p1,p3,p2);
Triangula(p2,p3,p1);
         }
      }
      public int Busca_minimo_y()
{
         double min = 0;
         int k, inp;
         //min = 0;
         k = -1;
         for (inp = 0; inp <= N_N - 1; inp++)
         {
            if (N[inp].y > min)
            {
               min = N[inp].y;
               k = inp;
            }
```

```
    }
    return k;
}

public int Busca_p2(int p1)
{
    int inp = 0, k = 0, aux1;
    Boolean encuen = false;
    while (inp < N_N && encuen == false)
    {
        if (p1 != inp)
        {
            encuen = true;
            k = 0;
            while (k < N_N && encuen == true)
            {
                if (k < N_N && k != p1 && k != inp)
                {
                    aux1 = k + 1;
                    while (Det(p1, inp, aux1) == 0 && aux1 < N_N)
                    {
                        aux1 = aux1 + 1;
                        if ((Det(p1, inp, k) * Det(p1, inp, aux1)) <= 0)
                        {
                            encuen = false;
                        }
                    }
                }
                k = k + 1;
            }
        }
        inp = inp + 1;
    }
    return inp - 1;
}

public void Busca_Punto_Bajo(int p1, int p2)
{
    Boolean encuentra = true;
    int i = 0;
    nodo v, m1, m2;
    double h;
    while (encuentra == true)
    {
        if (p1 != i && p2 != i)
        {
            encuentra = false;
        }
        i++;
    }
    i = i - 1;
    h = Det(p1, p2, i);
    v.x = N[p1].y - N[p2].y;
```

```
            v.y = - N[p1].x + N[p2].x;
            m1.x = v.x + (N[p1].x + N[p2].x)/2;
m1.y = v.y + (N[p1].y + N[p2].y)/2;
            m2.x = -v.x + (N[p1].x + N[p2].x)/2;
            m2.y = -v.y + (N[p1].y + N[p2].y)/2;

            N[N_N].x = m1.x;
            N[N_N].y = m1.y;
if (h * Det(p1,p2,N_N) > 0)
{
                N[N_N].x = m2.x;
                N[N_N].y = m2.y;
            }
        }
    }
}
```

Anexo 16: Formato de Archivo para exportación de puntos

```
ply
format ascii 1.0
comment Created by Blender 2.64 (sub 0) - www.blender.org, source file: ''
element vertex 4
property float x
property float y
property float z
property float nx
property float ny
property float nz
element face 1
property list uchar uint vertex_indices
end_header
-1.000000 -1.000000 0.000000 0.000000 0.000000 1.000000
1.000000 -1.000000 0.000000 0.000000 0.000000 1.000000
1.000000 1.000000 0.000000 0.000000 0.000000 1.000000
-1.000000 1.000000 0.000000 0.000000 0.000000 1.000000
4 0 1 2 3
```