UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO

FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE ESTADÍSTICA E INFORMÁTICA

"Desarrollo e Implementación de Software Multimedia para la Generación de Ambientes 3D a partir de Figuras 2D Aplicando el Método de Triangulación de Delaunay - Puno 2012"

TESIS

PRESENTADA POR:

Bach. CHOQUE LARICO, HENRY JOSE

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ESTADÍSTICO E INFORMÁTICO

PUNO – PERÚ

2013

UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO
FACUTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA
ESCUELA PROFESIONAL DE ESTADÍSTICA E INFORMÁTICA

TESIS

"DESARROLLO E IMPLEMENTACIÓN DE SOFTWARE MULTIMEDIA PARA LA GENERACIÓN DE AMBIENTES 3D A PARTIR DE FIGURAS 2D APLICANDO EL MÉTODO DE TRIANGULACIÓN DE DELAUNAY – PUNO 2012"

Presentada por:

Bach. CHOQUE LARICO, HENRY JOSÉ

A la Coordinación de Investigación de la Facultad de Ingeniería Estadística e Informática de la Universidad Nacional del Altiplano - Puno, para optar el Título Profesional de :

INGENIERO ESTADÍSTICO E INFORMÁTICO

APROBADA POR:

PRESIDENTE	:	M.Sc. Edgar Eloy Carpio Vargas
PRIMER MIEMBRO	:	M.Sc. Alejandro Apaza Tarqui
SEGUNDO MIEMBRO	:	M.Sc. Percy Huata Panca
DIRECTOR	:	M.C. César Augusto Lluén Vallejos
ASESOR	:	Ing. Alcides Ramos Calcina

DEDICATORIAS

Por el gran esfuerzo y apoyo incondicional para la culminación de este proyecto es que dedico este trabajo:

A mi madre Dominga Larico Vda. de Choque, por su esfuerzo realizado para que termine mi educación superior ha permitido que me desarrolle intelectualmente y personalmente, a mis hermanas Esther, Karina y Mitzy quienes siempre me apoyaron en cada momento.

A los docentes de la Facultad de Ingeniería Estadística e informática, por apoyarme en mi formación profesional y humanística.

AGRADECIMIENTOS

Quiero manifestar mi agradecimiento a la Escuela Profesional de Estadística e Informática por su impecable labor de continuas enseñanzas y apoyo durante mis estudios de pregrado.

A mis docentes quienes me aportaron con el conocimiento que pudieron obtener y transmitirlo con mucho empeño y profesionalismo.

Finalmente, pero no por ello menos importante, agradezco a mi familia, por comprenderme, apoyarme y motivarme en cada instante de mi vida.

A todos, Muchísimas Gracias.

Henry José

ÍNDICE GENERAL

DEDICATORIAS i

AGRADECIMIENTOS ii

RESUMEN ix

ABSTRACT xi

INTRODUCCIÓN xiii

CAPITULO I PLAN DE INVESTIGACION

1.1. Planteamiento del Problema 1

1.1.1. Definición del Problema 1

1.1.2. Justificación de la Investigación 2

1.2. Hipótesis 3

1.3. Objetivos 3

1.3.1. Objetivo General 3

1.3.2. Objetivos Específicos 4

CAPITULO II MARCO TEORICO

2.1. Antecedentes de la Investigación 5

2.2. Marco Teórico 7

2.2.1. Pixel 7

2.2.2. Generación de Primitivas Geométricas 10

2.2.3. Renderización 10

2.2.4. Triangulación de Delaunay 11

2.2.5. Imagen 14

2.2.6. Geometría Tridimensional 15

2.2.7. Proyección sobre la Pantalla del Computador 15

2.2.8. Transformaciones Tridimensionales 18

2.2.9. Entorno 22

2.2.10. Realidad Virtual 23

2.2.11. Librerías Graficas 25

2.2.12. Metodología 29

2.3. Marco Conceptual 34

2.3.1. Entorno 34

2.3.2. Imagen 34

2.3.3. Mallas 35

2.3.4. Pixel 35

2.3.5. Realidad Virtual 35

2.3.6. Renderización 36

2.3.7. Teselación 36

2.3.8. Triangulación de Delaunay 36

CAPITULO III MATERIALES Y MÉTODOS

3.1. Ubicación 37

3.2. Población 37

3.3. Muestra 37

3.4. Método de recolección de datos 37

3.5. Metodología de desarrollo 38

3.5.1. Programación Extrema 38

3.5.2. Herramientas Case 40

3.6. UML Lenguaje Unificado de Modelado 41

3.6.1. Diagramas de Casos de Uso 41

3.6.2. Diagramas de Clases 42

3.6.3. Diagramas de Secuencias 43

3.6.4. Diagramas de Actividades 44

3.6.5. Diagramas de Flujo 45

3.7. Material Experimental 48

3.7.1. Hardware 48

3.7.2. Software 48

3.8. Métricas de Calidad McCall 49

3.9. Factores de Calidad de McCall 50

3.9.1. Corrección 50

3.9.2. Fiabilidad 50

3.9.3. Eficiencia 50

3.9.4. Integridad 50

3.9.5. Flexibilidad 50

3.9.6. Capacidad de Pruebas 51

3.9.7. Reusabilidad 51

3.9.8. Usabilidad 51

CAPITULO IV RESULTADOS Y DISCUSIÓN

4.1. Diseño 52

4.2. Interfaz Grafica de Usuario 53

4.2.1. Elementos Utilizados para el Desarrollo 53

4.2.2. Elementos Externos 54

4.3. Implementación 54

4.4. Prueba del Sistema 55

4.5. Resultados 56

4.5.1. Metas 56

4.5.2. Proceso de Refinamiento de Delaunay 59

4.6. Métricas de calidad de McCall 59

4.6.1. Factores de Calidad de McCall..... 59

4.6.2. Corrección 61

4.6.3. Fiabilidad 62

4.6.4. Eficiencia 62

4.6.5. Integridad 63

4.6.6. Flexibilidad..... 64

4.6.7. Capacidad de Pruebas 64

4.6.8. Reusabilidad 65

4.6.9. Usabilidad 66

4.6.10. Calculo del Factor de Calidad de McCall 67

CONCLUSIONES 68

RECOMENDACIONES 69

BIBLIOGRAFÍA 70

ANEXOS 72

Anexo A

Cuestionario para la Evaluación de las Métricas de Calidad 73

Anexo B

Cuestionario para la Evaluación de los Factores de Calidad 74

Anexo C

C.1. Ventana de Renderizado con OpenGL 75

C.2. Ventana de Dibujo y Carga de Elementos 2D 77

C.3. Código para el Proyecto con CodeGear 2009 83

C.4. Estructura Dinámica de Datos Basados en Templetes 88

C.5. Algoritmo de Rotación de Imágenes 95

C.6. Refinamiento de Mallas de Delaunay 96

Anexo D

D.1. Manual del Usuario 119

ÍNDICE DE FIGURAS

2.1. Perspectiva de un Pixel y un conjunto de ellos formando una figura ... 8

2.2. Profundidad de color mediante RGB de 24-bits y escala de grises 8-bits 9

2.3. Primitivas geométricas renderizadas con OpenGL 10

2.4. Ejemplo de Renderización de vectores a sólidos 11

2.5. Ejemplo de generación de mallas por triangulación de Delaunay 12

2.6. Triangulación con todas las circunferencias circunscritas y sus centros 13

2.7. Conectando los centros de las circunferencias circunscritas se produce el diagrama de Voronoi 13

2.8. Vértices A,B,C del triángulo ABC misma distancia al circuncentro 14

2.9. Geometría tridimensional para la generación de ambientes simulados. 15

2.10. Proyección de figuras 3D en monitor 2D, con el punto de la perspectiva del observador 16

2.11. Efecto obtenido al multiplicar las coordenadas un punto por K 17

2.12. Efecto al dividir las coordenadas de un punto por su coordenada Z 17

2.13. Rotación de Imágenes de imágenes desde el punto X,Y y Z 20

2.14. Reposición de un objeto desplazándolo a las nuevas coordenadas 20

2.15. Ejemplo de un entorno 3D generado por computador 23

2.16. Ambiente virtual generado como posibilidad de plano 3D 24

2.17. Proceso de OpenGL de rotación de Imágenes 26

2.18. Esquema de trabajo multicapa de OpenGL 27

2.19. Logo de DirectX Renderizado en el mismo motor grafico 28

2.20. Fases de un proyecto de Extreme Programming 31

3.1. Fases de la metodología XP 38

3.2. Gráfico de Grafico de Casos de Uso 41

3.3. Gráfico de Diagramas de Clases 42

3.4. Gráfico de Diagramas de Secuencia 43

3.5. Gráfico de Diagramas de Actividades 44

3.6. Gráfico de Diagramas de Flujo 47

3.7. Modelado de compilación en C 49

4.1. Entorno de desarrollo CodeGear RAD Studio 2009 52

4.2. Interfaz gráfica de Media3D 53

4.3 Modulo principal de la aplicación de Media3D 57

4.4. Módulo Graficador en 2 dimensiones en Media3D 58

4.5. Mallas resultantes en Media3D 58

4.6. Entorno Renderizado en Media3D 59

4.7. Proceso de Refinamiento de Delaunay 59

C.1. Vista del Proyecto de Media3D en tiempo de desarrollo en CodeGear 2009 76

C.2. Vista del Proyecto de Media3D en CodeGear 2009 mostrando la interfaz grafica 83

C.3. Refinamiento de mallas por triangulación de Delaunay 97

ÍNDICE DE CUADROS

2.1. Diagramas de UML 33

4.1. Cuadro de los Contenidos de los Controles de Media3D 53

6.1. Refinamiento de mallas por triangulación de Delaunay 97

RESUMEN

El problema que se presentaba en la generación de ambientes 3D partir de figuras 2D con las aplicaciones comerciales es que no permitía generar ambientes 3D complejos por la lentitud de los procesos de generación de los puntos de intersección de las mallas, las cuales no admitían dar un refinamiento adecuado a los ambientes 3D en las superficies y volúmenes de los objetos. En este sentido la presente tesis se ha desarrollado con el objetivo de "Desarrollar un software eficiente en la generación de ambientes 3D a partir de figuras 2D aplicando el método de Triangulación de Delaunay", con el fin de que esta herramienta facilite el trabajo de generación de ambientes 3D complejos, haciéndolos más precisos, adaptables y de más rápida realización.

Para cumplir con el objetivo trazado se utilizó el modelo de desarrollo de programación extrema, ya que es el más destacado de los procesos ágiles de desarrollo de software. Para hallar la eficiencia del sistema se utilizaron las métricas de calidad de McCall, donde se aplicaron dos cuestionarios a cuatro especialistas que trabajan con diseños 3D, quienes hicieron las pruebas de uso del sistema. Se usó el muestreo no probabilístico, ya que nos permitió tomar una muestra a criterio del investigador, las comprobaciones cuantitativas implementadas se han basado en las métricas de calidad de McCall, con ella se ha comprobado numéricamente la eficiencia del sistema que fue 8.19 en una escala de 0 a 10, la cual nos indicó que el software tiene un alto nivel de calidad.

Por tanto, el Software Multimedia para la generación de ambientes 3D a partir de figuras 2D aplicando el método de triangulación de Delaunay en la

renderización de imágenes es óptima dando lugar a un aspecto sumamente realista en sentido visual, con esta investigación concluimos que este nos permitirá incluso generar mallas de ambientes a un más complejas.

Palabras claves: Ambientes, Método, Triangulación, Mallas, Renderización

ABSTRACT

The problem that is presented in the generation of 3D environments from 2D pictures with business applications is not allowed to generate complex 3D environments by the slowness of the generation of the points of intersection of the mesh, which did not admit give suitable refinement to 3D environments in the area and volume of objects. In this sense this thesis has been developed with the aim to "develop an efficient software in generating 3D environments from 2D pictures using the Delaunay triangulation method", so that this tool facilitates the work of generation complex 3D environments, making them more precise, adaptable and faster completion.

To meet the stated objective of the development model of extreme programming was used, as it is the highlight of the agile software development processes. To find the efficiency of the quality metrics of McCall, where two questionnaires were applied to four specialists who work with 3D designs, who made use of the test system used. Non-probability sampling was used because it allowed us to take a sample at the discretion of the investigator, quantitative tests have been implemented based on quality metrics McCall with it is numerically verified the efficiency of the system was 8.19 on a scale from 0 to 10, which indicated to us that the software has a high level of quality.

Therefore, the Multimedia Software for generating 3D environments from 2D pictures using the method of Delaunay triangulation in rendering of images is optimal giving rise to a highly realistic appearance in visual sense, with this research we conclude that this will allow us even further generate grids complex environments.

Keywords: Environments, Methods, Triangulation, Meshes, Rendering.

INTRODUCCIÓN

Las investigaciones de carácter computacional o informática en nuestro país han sido orientadas erróneamente a la ejecución de sistemas de información, lo que nos ha llevado a ser tipificados como usuarios e incluso técnicos, pero lo que buscamos con una carrera profesional de educación superior es aplicar conocimiento basado en investigación, en la generación de nuevo y mejor conocimiento partiendo de investigaciones previas agregando nuestro aporte, este pensamiento nos llevara a disminuir los parámetros con lo que han tipificado nuestra muy valorada profesión, mostrando que estamos y debemos enfocarnos en la matemática abstracta para generar nuevos y mejores esquemas computacionales de resolución a los diversos problemas inherentes.

Una de las principales herramientas de la geometría computacional la constituyen las triangulaciones, tanto de nube de puntos como polígonos. En este tipo de problemas es donde aparece la triangulación de Delaunay, para los vectores que generemos describiremos la mejor forma de implementarlo en un programa generador de ambientes 3D.

La triangulación de Delaunay es uno de los métodos más rápidos y relativamente fáciles de implementar para contar con algoritmos bastante eficientes para el cálculo. En este proyecto se pretende elaborar un algoritmo y método que permita realizar este tipo de triangulaciones en espacios de varios puntos. Por lo tanto el mayor problema es manejar la nube de puntos de figuras 2D y por consiguiente crear una triangulación tan considerable. La estrategia es dividir en regiones, para luego calcular triangulaciones en sub espacios por separado, de esa manera disminuir la complejidad del problema.

El presente trabajo de tesis consiste en el desarrollo e implementación de un software multimedia para la generación de ambientes 3D a partir de figuras 2D aplicando el método de triangulación de Delaunay.

La estructura de la presente investigación es la siguiente :

En el Capítulo I se realizó la identificación del problema, planteamiento de objetivos e hipótesis de la investigación.

En el Capítulo II se desarrolló la búsqueda de la información disponible que nos ayude en la resolución del problema, la teoría disponible de los métodos para desarrollar el Software.

En el Capítulo III se aplica los materiales y métodos para el desarrollo del Software, para la obtención de resultados según los objetivos planteados

En el Capítulo IV se realiza la discusión de los resultados de la implementación de Software Media 3D.

Finalmente, se tratan las conclusiones de la investigación, recomendaciones, bibliografías y los anexos de Investigación.

CAPÍTULO I

PLAN DE LA INVESTIGACIÓN

1.1 Planteamiento del Problema

1.1.1 Definición del Problema

Los ambientes de 3 dimensiones (3D) ofrecen una visión de la realidad simulada por computador, ofreciendo todas las características visuales, aunque carezcan de la sensación integra de movimiento esto puede fácilmente asimilarse por el cerebro completándose a través de la vista. Diseñar ambientes tridimensionales se convierte en una labor bastante complicada, por el hecho de que debe completarse las paredes y el color de ellas, triplicando el esfuerzo de construir cualquier tipo de ambiente.

El proceso normal es la obtención de los puntos en 3D, también las tonalidades de luz, sombra, color y textura como podría apreciarse en cualquier archivo de datos exportado por programas como 3D Max Studio, Maya entre otros, el objetivo es generar de forma automática y haciendo mayor hincapié en el proceso de generación de los puntos 3D a partir de puntos 2D, para la generación de la malla que más adelante se podrá dar color y textura para generar un ambiente.

El proceso de obtención de los puntos y la generación de mallas será aportado por los conceptos de la **Triangulación de Delaunay**, de esta

forma dar solución al problema de investigación, se requiere desarrollar el software que muestre este proceso y detallar en la investigación, describiendo los algoritmos usados para el proceso de transformaciones geométricas que aplicaremos sobre los objetos generados.

Nuestra Investigación se centra en creación de una aplicación que permita generar ambientes 3D facilitando a los usuarios la creación a partir de figuras 2D, con ello nos hacemos la pregunta: **¿La implementación de software multimedia mejorara la generación de ambientes 3D a partir de figuras 2D aplicando el método de triangulación de Delaunay?**

1.1.2 Justificación de la Investigación

El sistema se compone de dos módulos principales. El primero realiza el procesamiento de imagen, cuyo objetivo es determinar el mapa de profundidad en un par de vistas, donde cada par de vistas sucesivas sigue una secuencia de fases.

Primero la detección de puntos de interés, correspondencia de puntos y reconstrucción de puntos; en el proceso de reconstrucción se determinan los parámetros que describen el movimiento (matriz de rotación R y el vector de traslación T) entre las dos vistas. Esta secuencia de pasos se repite para todos los pares de vistas sucesivas del conjunto.

El segundo módulo tiene como objetivo crear el modelo 3D del objeto, para lo cual debe determinar el mapa total de todos los puntos 3D

generados; en cada iteración del módulo anterior, una vez obtenido el mapa de profundidad total, genera la malla 3D, aplicando el método de Triangulación de Delaunay. Los resultados obtenidos del proceso de reconstrucción son modelados usando las librerías gráficas de OpenGL[1] y usando el lenguaje de programación C++ bajo los sistemas operativos Linux y Windows para de este modo obtener una visualización más realista del objeto.

1.2 Hipótesis

El proceso de generación de ambientes 3D a partir de figuras 2D, mejorara después de aplicar el método de Triangulación de Delaunay.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar e Implementar un software multimedia para la generación de ambientes 3D a partir de Figuras 2D aplicando la Triangulación de Delaunay.

1.3.2 Objetivos Específicos

- Plantear los algoritmos para la transformación y rotación de imágenes.
- Implementar el algoritmo de generación de mallas, basado en Triangulación de Delaunay para renderizar el ambiente 2D y generar una perspectiva de 3D.

[1] OpenGL : Librería gráfica desarrollada en Silicon Graphics

- Desarrollar un software que permita transformar figuras 2D en ambientes de 3 Dimensiones.

CAPÍTULO II

MARCO TEORICO

2.1. Antecedentes de la Investigación

- En la Tesis Doctoral **(Tejada , 2008)**, se usan técnicas de generación de mallas para la creación de superficies aplicables en campos como la Topografía e Ingeniería Civil, hemos tomado los conceptos vertidos de la Triangulación de Delaunay y los diagramas de Voronoi aplicados en la generación de vectores con relación geométrica. Otras investigaciones de este autor son: Redes ART como Motores de Inferencia (ART Networks as Inference Engines). Engineer Thesis, University San Agustin de Arequipa, Peru, 2005. y agrupamiento visual en grandes conjuntos de dados multidimensionales (Visual clustering in large multi-dimensional datasets). Master thesis, University of Sao Paulo (ICMC), Sao Carlos, Brazil, 2003.

- En la Tesis **(Alex Cuadros, 2001)** presenta dos nuevas maneras de atacar el problema de generación de modelos a partir de imágenes. La primera, denominada Beta-Connection aborda el problema por medio de secciones planares, o sea regiones contenidas en cortes paralelos de un objeto son conectadas para producir un modelo tridimensional. A diferencia de otras técnicas de la literatura, que en general adoptan heurísticas para definir la correspondencia entre contornos, Beta-

Connection posibilita generar una familia de modelos a partir de un mismo conjunto de contornos. Esta familia es definida en función del parámetro Beta, es decir, alterando este parámetro se establece diferentes conexiones entre contornos adyacentes, permitiendo tratar el problema de la correspondencia de una forma mucho más flexible.

- La segunda estrategia presentada en este trabajo, denominada Imesh, adopta una idea basada en volúmenes o sea los modelos 3D son generados directamente a partir de imágenes volumétricas. La técnica Imesh se distingue de las demás estrategias de generación de modelos a partir de volúmenes por incorporar mecanismos de segmentación en el proceso de reconstrucción. El proceso es guiado por una métrica de error que puede ser fácilmente alterada, permitiendo que descomposiciones simples con características específicas sean generadas. Ejemplos de utilización de las dos técnicas en diferentes tipos de aplicaciones serán presentados y discutidos.
- En la Tesis **(Oliver Vilca, 2009)** Para el refinamiento de mallas, se ha diseñado diferentes estrategias, por ejemplo, las que se basan en la inserción de vértices en el circuncírculo de los triángulos que se requieren refinar.
- El estudio del refinamiento de mallas es un tema teórico de interés. Sin embargo, existen inconvenientes para su análisis, se desconoce el comportamiento exacto del proceso de refinamiento de mallas geométricas. El resultado varía y depende de los algoritmos de construcción y refinamiento de mallas. En distintas aplicaciones se

necesita resolver el siguiente problema: Dada una triangulación inicial Ti de calidad aceptable3 en particular de triángulos rectángulos isósceles, una región de refinamiento R de Ti y un parámetro de longitud de arista más larga requerida; refinar los triángulos que intersectan la región R (por ejemplo el área definida por un cuadrado) construyendo una nueva triangulación Tf.

Para mejorar los conceptos que adquirimos de estos textos también se consultó en la revista digital indexada **SciELO** la obra **(Natividad Grandon, 2007),** también hemos aprovechado el convenio de la Universidad Nacional del Altiplano con el sitio WEB especializado en investigaciones científicas **eBrary** [2]

2.2. Marco Teórico

2.2.1. Pixel

Es la mínima unidad de medida gráfica, no representa necesariamente un punto en la pantalla. Pixel, abreviatura de Picture Element, es un único punto en una imagen gráfica.

Los monitores gráficos muestran imágenes dividiendo la pantalla en miles (o millones) de pixeles, dispuestos en filas y columnas. Los pixeles están tan juntos que parece que estén conectados **(Rafael, 1996).**

El número de bits usados para representar cada pixel determina cuántos colores o gamas de gris pueden ser mostrados. Por ejemplo, en modo color de 8-bits, el monitor en color utiliza 8 bits para cada pixel,

[2] eBrary : www.eBrary.com

permitiendo mostrar 2 elevado a 8 (256) colores diferentes o gamas de gris.

En monitores de color, cada pixel se compone realmente de tres puntos rojo, verde y azul. Idealmente, los tres puntos convergen en el mismo punto, pero todos los monitores tienen cierto error de convergencia que puede hacer que el color de los pixeles aparezca borroso.

Figura 2.1 : Perspectiva de un pixel y un conjunto de ellos formando imágenes.

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 o en un Framebuffer[3].

Debido a la naturaleza del sistema binario de numeración, una profundidad de bits de ***n*** implica que cada píxel de la imagen puede tener ***n*** posibles valores y por lo tanto, representar ***2n*** colores distintos. Debido a la aceptación prácticamente universal de los octetos de 8 bits como unidades básicas de información en los dispositivos de

[3] **Framebuffer** : Bloque de bytes a ser enviados al controlador de vídeo, Esp. VESA 3.0

almacenamiento, los valores de profundidad de color suelen ser divisores o múltiplos de 8, a saber 1, 2, 4, 8, 16, 24 y 32, con la excepción de la profundidad de color de 10 o 15, usada por ciertos dispositivos gráficos.

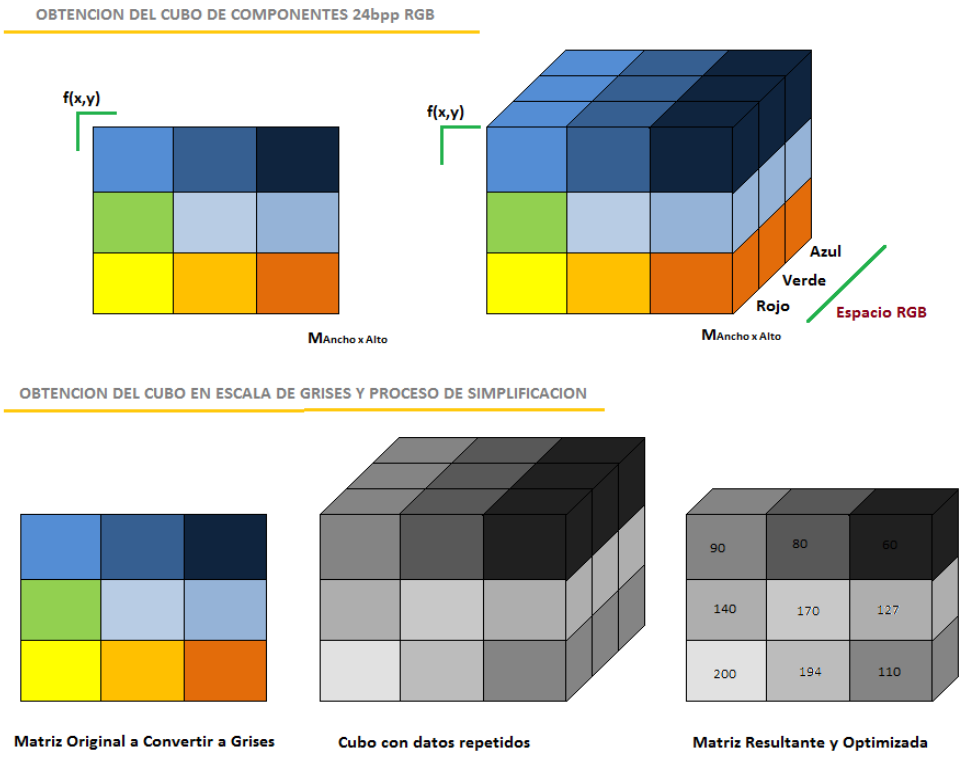


Figura 2.2 : Profundidad de color mediante RGB de 24-bits y escala de grises 8-bits.

2.2.2 Generación de Primitivas Geométricas

Nos basamos en las reglas del eje cartesiano de 2 dimensiones para el trazado y la creación de los puntos que a su vez generaran las primitivas geométricas, con la distinción de que para graficar los puntos sin tener que preocuparnos por la generación mental de los puntos en 2D a 3D usaremos un software desarrollado para la presente investigación. Los puntos y el proceso de generado de 2D y la triangulación para 3D, pueden verse en la Figura 3.

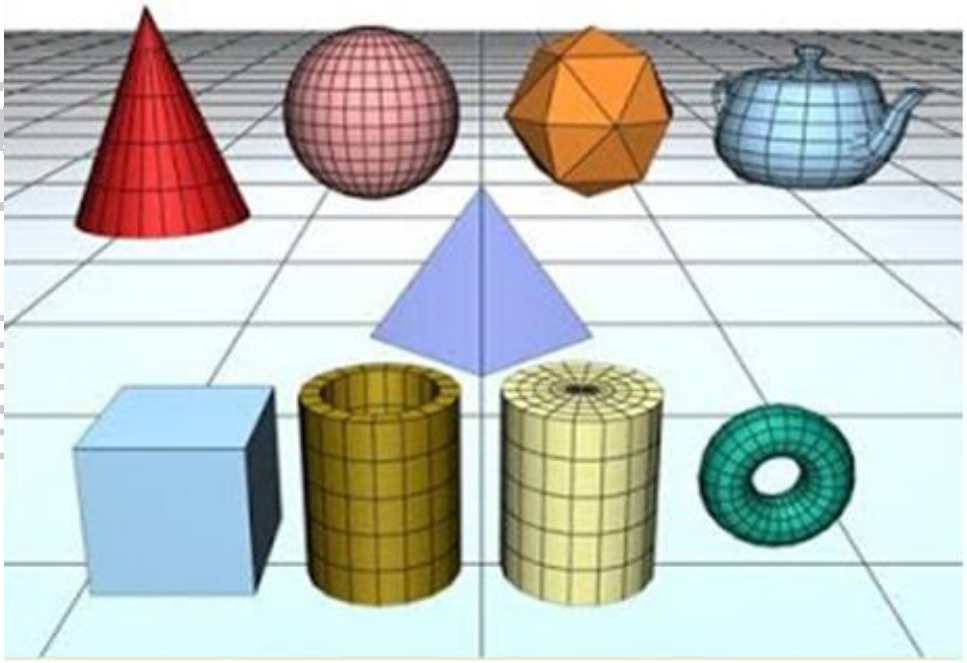


Figura 2.3 : Primitivas geométricas renderizados con OpenGL .

2.2.3 Renderización

La palabra renderización es una adaptación al castellano del vocablo inglés "rendering" y que define un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen o secuencia de imágenes.

Habitualmente se utiliza esta nomenclatura para definir el proceso por el cual se pretende imitar un entorno tridimensional, formado por estructuras poligonales, luces, texturas y materiales, simulando ambientes y estructuras físicas verosímiles. También se suele utilizar para procesos 2D que requieren cálculos complejos como la edición de vídeo, la animación o el desarrollo de efectos visuales basados en proyecciones vectoriales **(Schuller, 2011)**.

Es el proceso de creación de una superficie plana o curva a través del proceso de llenado de los polígonos en una colección de triángulos encadenados, los cuales tienen los puntos básicos de la forma o figura que esta mapeándose (**M0RPhEuS, 2008**).

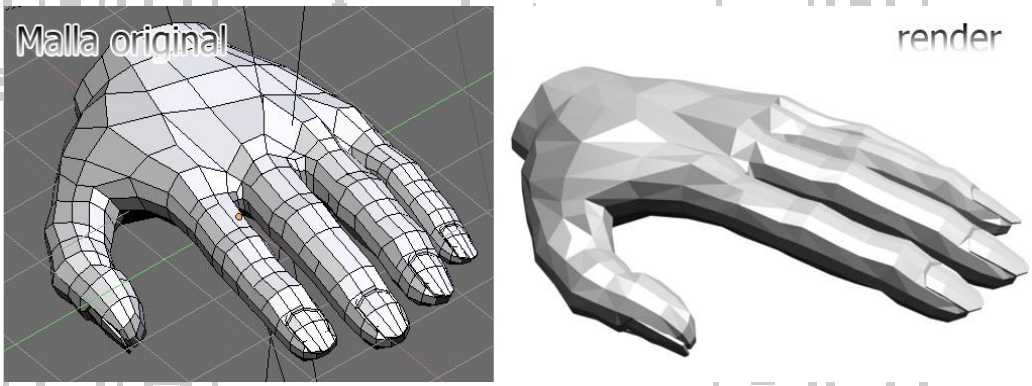


Figura 2.4 : Ejemplos de renderización de vectores a sólidos.

2.2.4 Triangulación de Delaunay

Es una red de triángulos que cumple la condición de Delaunay. Esta condición dice que la circunferencia circunscrita de cada triángulo de la red no debe contener ningún vértice de otro triángulo. Se usan triangulaciones de Delaunay en geometría por ordenador.

La circunferencia circunscrita de un triángulo es la circunferencia que

contiene los tres vértices del triángulo. Según la definición de Delaunay la circunferencia circunscrita es vacía, si no contiene otros vértices aparte de los tres que la definen.

La condición de Delaunay dice que una red de triángulos es una triangulación de Delaunay si todas las circunferencias circunscritas de todos los triángulos de la red son vacías. Esa es la definición original para espacios bidimensionales. Es posible ampliarla para espacios tridimensionales usando la esfera circunscrita en vez de la circunferencia circunscrita. También es posible ampliarla para espacios con más dimensiones pero no se usa en la práctica.

Esa condición asegura que los ángulos del interior de los triángulos son lo más grandes posible. Es decir, maximiza la extensión del ángulo más pequeño de la red optimizando el número total generado (**Natividad Grandon, 2007**), podemos ver una descripción gráfica en la Figura 5.

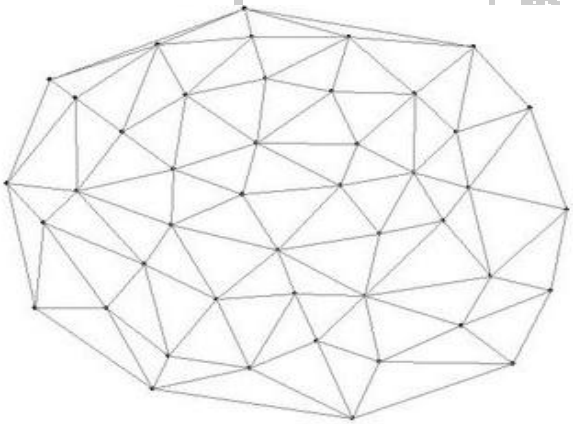


Figura 2.5 : Ejemplo de generación de mallas por triangulación de Delaunay .

Propiedades

- La triangulación forma la envolvente convexa del conjunto de puntos.

- El ángulo mínimo dentro de todos los triángulos está maximizado.
- La triangulación es unívoca si en ningún borde de circunferencia circunscrita hay más que tres vértices.

Relación con los diagramas de voronoi

La triangulación de Delaunay con todos los circuncentros es el grafo dual del diagrama de Voronoi: los circuncentros son los vértices de los segmentos del diagrama:

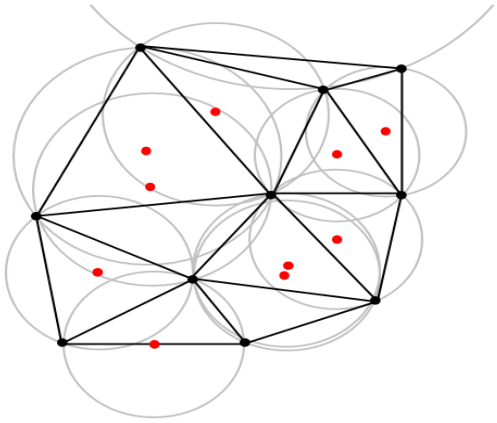


Figura 2.6 : La triangulación con todas las circunferencias circunscritas y sus centros.

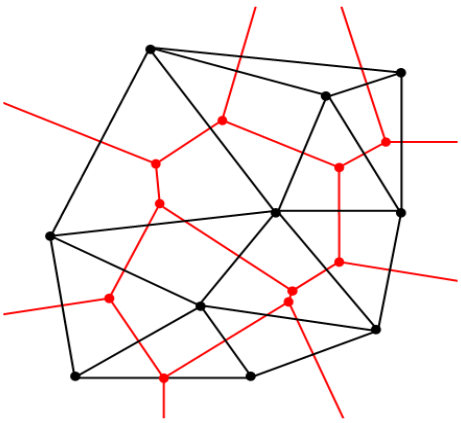
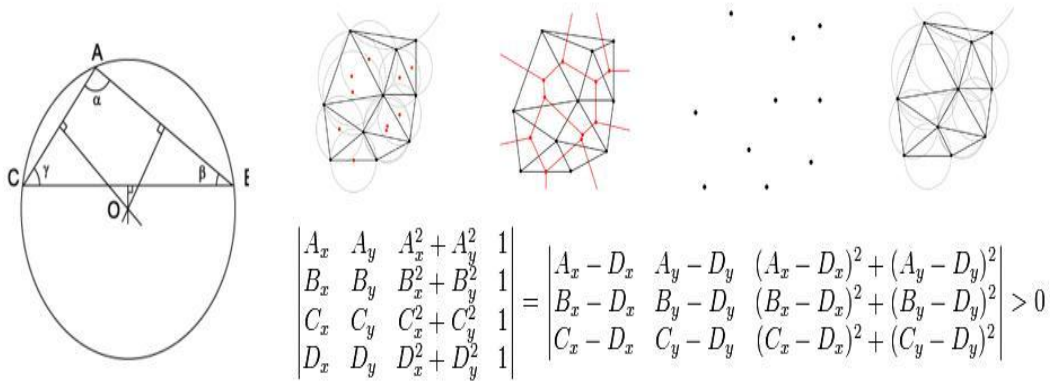


Figura 2.7 : Conectando los centros de las circunferencias circunscritas se produce el diagrama de Voronoi.



$$\begin{vmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{vmatrix} = \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x-D_x)^2+(A_y-D_y)^2 \\ B_x-D_x & B_y-D_y & (B_x-D_x)^2+(B_y-D_y)^2 \\ C_x-D_x & C_y-D_y & (C_x-D_x)^2+(C_y-D_y)^2 \end{vmatrix} > 0$$

Figura 2.8: Vértices A, B, C del triangulo ABC misma distancia al circuncentro O .

2.2.5 Imagen

Es la unidad o primitiva en el desarrollo de entornos, puede tomarse como tal a un vector en el espacio 2D, 3D hasta una malla de polígonos que trazan la superficie de un rostro, en el caso de una fotografía se debe tomar como una imagen de mapa de bits es decir una colección matricial de puntos.

El término imagen se refiere a una función bidimensional de la luz y la intensidad, a la que indicamos por ***f (x,y)*** da la intensidad (iluminación) de la imagen en este punto puesto que la luz es una forma de energía, ***f (x,y)*** debe ser estrictamente mayor que cero y finita es decir una cantidad positiva de puntos para los elementos de la colección **(Rafael, 1996)**.

2.2.6 Geometría Tridimensional

Consiste en algunos aspectos de los tratamientos que se deben hacer sobre los objetos en tres dimensiones, para poder introducirlos y tratarlos dentro de un lenguaje de programación, para esto primero se hablara de cómo proyectar algo ubicado en el mundo tridimensional, sobre la pantalla bidimensional de un computador y luego sobre como operar sobre los puntos en tres dimensiones utilizando la técnica matricial **(Ruiz, 2002).**

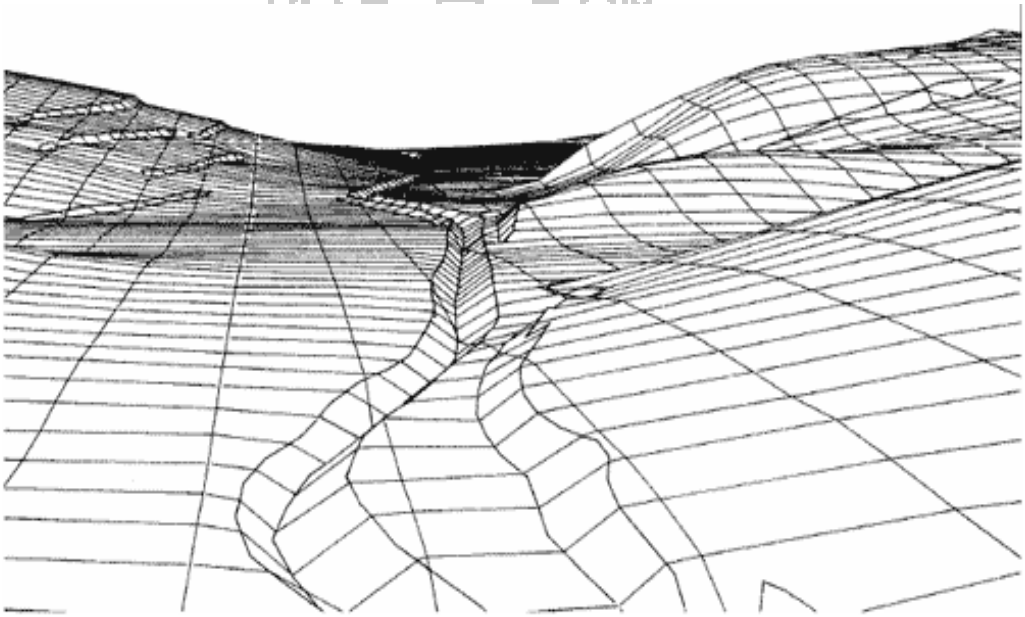


Figura 2.9 : Geometría tridimensional para generación de ambientes simulados.

2.2.7 Proyección sobre la Pantalla del Computador

Lo primero que se mirará, en el desarrollo de un programa para representar objetos sólidos en la pantalla de un computador, es buscar cómo se puede representar un punto que posee 3 dimensiones, dentro de la pantalla que tiene 2 dimensiones.

Para esto habría que hallar una forma para hacer que las coordenadas **X** y **Y** en la pantalla dependieran de la coordenada **Z** del punto, para ello considérese el caso de un punto en tres dimensiones multiplicado por una constante : **k(x; y; z) = (kx; ky; kz)**

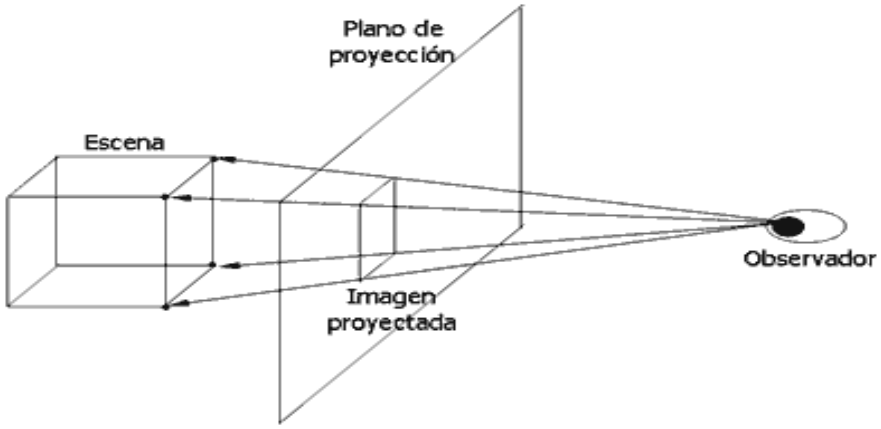


Figura 2.10 : Proyección de figuras 3D en monitor 2D, con el punto de la perspectiva del observador.

Con esto se obtiene otro punto que esta sobre la misma línea que une el punto original con el origen de coordenadas, como se ve en la Figura 2.11

Ahora, si se cambia la constante **k** de la formula anterior por **1/z:**

Se convierte en :

1/z(x, y, z) = (x/z, y/z, 1)

Se tendrá un punto que yace sobre el plano z = 1, que para nuestro caso vendrá a ser la pantalla del computador.

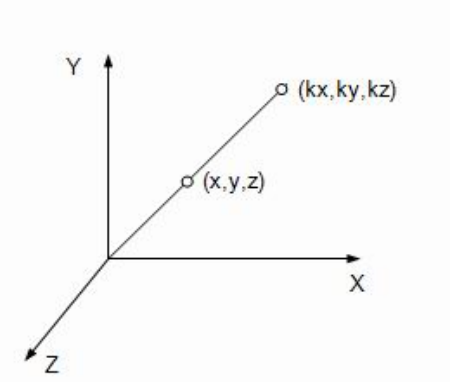


Figura 2.11 : Efecto obtenido al multiplicar las coordenadas un punto por k

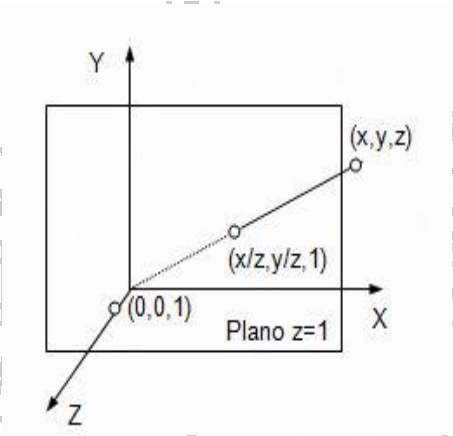


Figura 2.12 : Efecto al dividir las coordenadas de un punto por su coordenada Z.

Si se hace lo mismo con todos los puntos del sólido que se quiere representar, se tendrá una proyección en perspectiva de él sobre el plano ***z = 1,*** teniendo como punto de visión al origen como se puede ver en la figura. Si se aleja o acerca del origen el plano de proyección, se obtendrá como resultado un cambio en la perspectiva del objeto proyectado, para hacer esto se debe cambiar el ***1/z*** de la formula anterior por ***a/z***, donde a indica la distancia hasta el origen del plano de proyección **(Rafael,96)**., también indica la perspectiva de la proyección. La fórmula entonces quedaría así: ***a/z(x, y, z) = (a.x/z, a.y/z, a)***

Entre más grande sea está , la proyección obtenida se acercara a la proyección paralela. Esto ocurre en la realidad, por ejemplo: entre más

cerca se está de una construcción se verá en perspectiva, en cambio si se está muy lejos de ella, tanto como para tener que verla con un telescopio, se verá casi en proyección paralela.

Puntos situados detrás del plano de proyección ***(z = a)***, y delante del **plano *x - y (z = 0)***, es decir con su coordenada ***z*** entre los valores de ***0*** y a son proyectados correctamente, como ocurría con los situados delante del plano de proyección. No ocurre lo mismo con los puntos que están en el plano ***x - y (z =0)*** ni con los que están detrás de él. Los primeros, al ser proyectados, recordando la fórmula de proyección, debido a que su coordenada z es cero, producen una división por cero; y los otros son proyectados al revés, sus coordenadas ***x*** *y* ***y*** son cambiadas de signo al ser divididas por la coordenada ***z*** negativa del punto. Este problema se soluciona fácilmente teniendo en cuenta que todos estos puntos están detrás o al mismo nivel del punto de visión, lo que quiere decir que son puntos no visibles y por lo tanto no se dibujaran. Para resolver esto en el programa se compara la coordenada **z** de cada punto con ***0***, si es mayor, el punto (o mejor, la cara) se dibuja, de otra forma no.

2.2.8 Transformaciones Tridimensionales

El modo más eficiente y fácil de entender de hacer manipulaciones tridimensionales es utilizar matrices de transformaciones.

Se pueden tener varias matrices que contengan por ejemplo, rotaciones sencillas para luego multiplicarlas hasta obtener una sola matriz, esta

matriz representa la combinación de las rotaciones de cada una de las otras matrices.

Pero al igual que en las rotaciones normales, la posición final del punto o conjunto de puntos depende del orden en que se hayan efectuado las multiplicaciones de las matrices.

Rotación

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

$$Rz(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

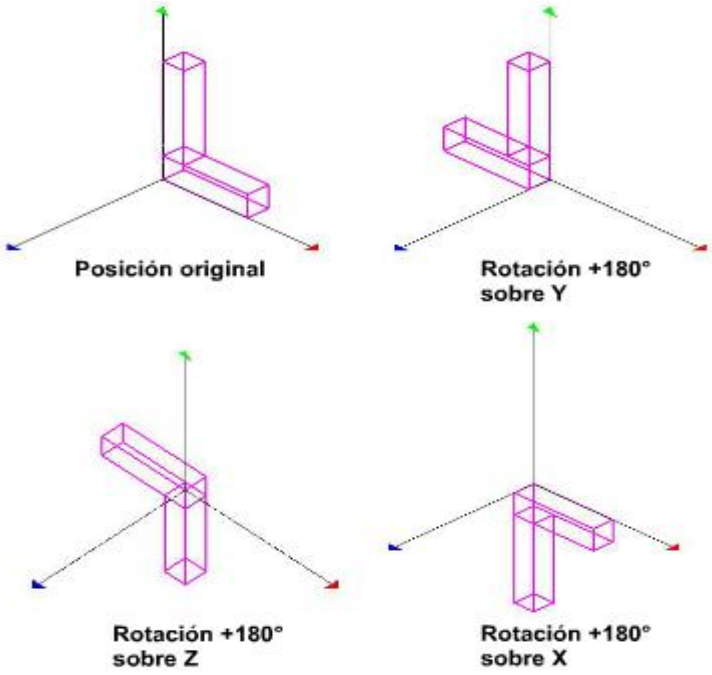


Figura 2.13: Rotación de imágenes desde el puno X, Y y Z .

Traslación

Supóngase que se trata de trasladar un punto de coordenadas ***(X,Y,Z)*** a un nuevo emplazamiento, empleando unos desplazamientos ***(Xo,Yo,Zo)***. La traslación se realiza fácilmente empleando las ecuaciones.

$$X' = X + Xo$$
$$Y' = Y + Yo$$
$$Z' = Z + Zo$$

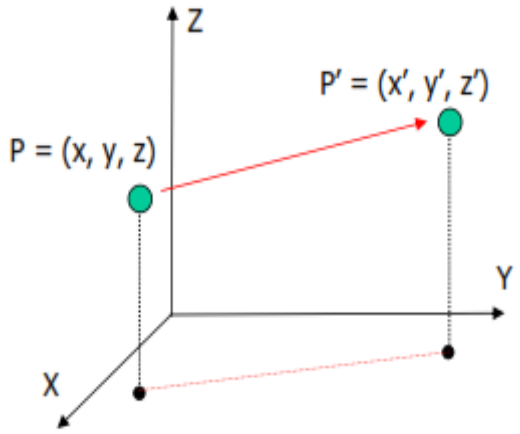


Figura 2.14 : Reposición de un objeto desplazándolo a las nuevas coordenadas.

Escalado o Re-dimensionamiento

El cambio de escala empleando factores ***Sx, Sy*** y ***Sz*** a lo largo de los ejes ***X*** **,** ***Y*** y ***Z***, viene dando por la matriz de transformación:

$$S = \begin{bmatrix} Sx & 0 & 0 & X_0 \\ 0 & Sy & 0 & Y_0 \\ 0 & 0 & Sz & Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La Técnica Matricial

El método anterior en donde se utilizan matrices para realizar operaciones tridimensionales es denominado técnica matricial. En esta técnica, la orientación de un objeto viene dada por la matriz de transformaciones, en ella se indica la posición en el espacio del objeto y la orientación del mismo. En realidad es una matriz 4x4 y no 3x3 como las mostradas anteriormente, pero para las operaciones de rotación, sólo se opera con esta sub matriz.

Cuando se van a hacer rotaciones y/o traslaciones sobre objetos 3D, no se efectúa cada una de estas operaciones por separado sobre el objeto, sino que todas estas operaciones, representadas por matrices, se aplican a la matriz de transformación y una vez se obtiene el conjunto de todas las operaciones a realizar sobre el objeto, esto es una única matriz, se procede a transformar cada uno de los puntos del objeto con esta matriz.

De esta manera para lograr una orientación dada en el espacio, no se hace cada operación sobre todos los puntos, sino sobre una matriz, para luego si aplicar la matriz obtenida a todos los puntos.

Como se dijo antes, la matriz de transformación, representa una posición y orientación del objeto 3D en el espacio, entonces, se analizarán sus elementos:

Posee 3 vectores que representan los 3 ejes coordenados del objeto, el primer vector representa el eje x del objeto, y está formado por los 3 primeros números de la primera fila, el eje y, los 3 primeros números de la segunda fila y el eje z los 3 primeros números de la tercera fila. La matriz se muestra a continuación:

$$\begin{array}{ll} \text{Eje } x \text{ del objeto} \rightarrow & \\ \text{Eje } y \text{ del objeto} \rightarrow & \\ \text{Eje } z \text{ del objeto} \rightarrow & \\ \text{Centro del objeto} \rightarrow & \end{array} \begin{bmatrix} X_x & X_y & X_z & 0 \\ Y_x & Y_y & Y_z & 0 \\ Z_x & Z_y & Z_z & 0 \\ C_x & C_y & C_z & 1 \end{bmatrix}$$

2.2.9 Entorno

Se define a un conjunto de elementos gráficos renderizados bajo un ambiente computacional, es la base de la realidad virtual sin tener que agregar toda la complejidad que esta envuelve, podemos ver un ejemplo de entorno 3D en la Figura 2.15.

Pero la creación de entornos virtuales no obedece específicamente a un fin sino que sirve como base para el desarrollo de proyectos de terceros, generando así un aporte importante.

Figura 2.15: Ejemplo de un entorno 3D generado por computador.

2.2.10 Realidad Virtual

Es un sistema tecnológico, basado en el empleo de ordenadores y otros dispositivos, cuyo fin es producir una apariencia de realidad que permita al usuario tener la sensación de estar presente en ella. Se consigue mediante la generación por ordenador de un conjunto de imágenes que son contempladas por el usuario a través de un casco provisto de un visor especial. Algunos equipos se completan con trajes y guantes equipados con sensores diseñados para simular la percepción de diferentes estímulos, que intensifican la sensación de realidad.

Su aplicación, aunque centrada inicialmente en el terreno de los videojuegos, se ha extendido a otros muchos campos, como la medicina o las simulaciones de vuelo.

La visión por computadora es el campo de la Inteligencia Artificial que estudia los sistemas dotados con la capacidad de ver el entorno que los rodea. Este campo es muy extenso y abarca desde las técnicas generales hasta las más especializadas, cubriendo una gran gama de aplicaciones, que incluyen el reconocimiento de caracteres, la interpretación de fotografías, la identificación de huellas dactilares y el control de robots y autómatas independientes del entorno que los rodea (Nilson, 2001).

Figura 2.16 : Ambiente virtual generado como posibilidad de plano 3D.

2.2.11 Librerías Graficas

Es una API[4] de funciones desarrollada para aprovechar las prestaciones gráficas de los actuales tarjetas gráficas con aceleración 3D y una Unidad de Procesamiento de Gráficos (GPU), dejando al usuario la labor de trabajar con el hardware y poder concentrarse en el resultado.

A continuación describimos las librerías más importantes, usados entre los desarrolladores de juegos y aplicaciones gráficas:

BGI - Borland Graphics Interface

Desarrollada en 1992 para el compilador Turbo C++ 3.0, y modificada para el compilador Borland C++ 3.1, esta librería integra modos de video en resoluciones de 320x200 pixeles, 640x480 en modos de 16 colores, además contiene una API de funciones para crear figuras y formas básicas, así como el desarrollo de nuevos controladores gráficos como Súper VGA (VESA).

OpenGL

Desarrollada en Silicón Graphics para la renderización de primitivas geométricas, cuenta con múltiples opciones de transformación de ambientes enteros, un ejemplo bastante claro puede verse en la Figura.

[4] **API : Interfaz Programable de Aplicaciones**

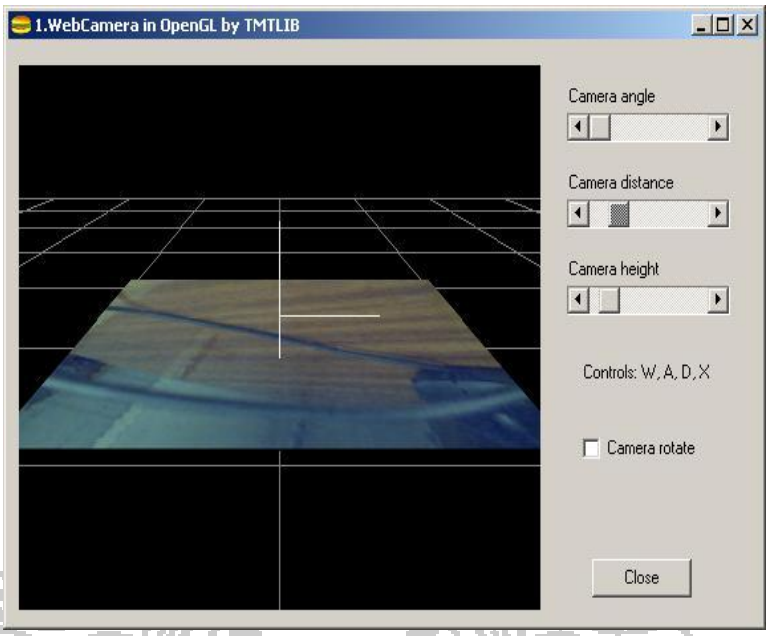


Figura 2.17 : Proceso de OpenGL de rotación de imágenes

Cada computador tiene hardware gráfico especial que controla lo que se visualizar en cada computador que controla el quehacer, la Figura muestra como OpenGL es usado por los juegos sobre los computadores, u otra pieza de software usa sus comandos para suplir la carencia de acceso directo al hardware gráfico **(schullerl, 2011).**

Esta librería es una de las más antiguas y reconocidas, de las más populares que los programadores de juegos han tenido. Fue desarrollada en 1992 por Silicón Graphics Inc. (SGI) pero fue realmente interesante para los programadores de juegos después de haber sido usado for GLQuake (Doom3D) en 1997. The GameCube, Wii, PlayStation 3, y el iPhone basaron sus librerías graficas en OpenGL.

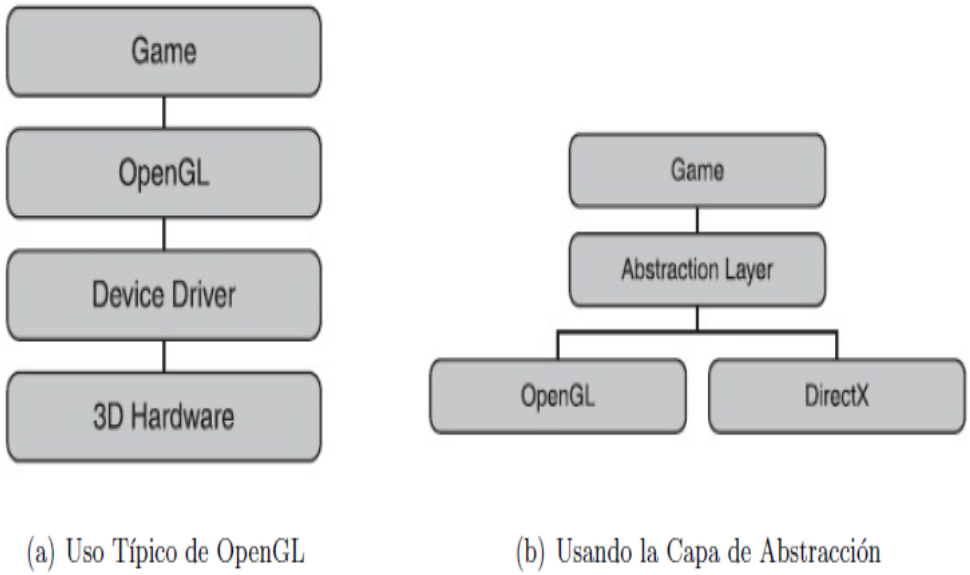


Figura 2.18: Esquema de trabajo multicapa de OpenGL

Microsoft DirectX

API de interacción gráfica entre el sistema operativo Windows y el dispositivo de vídeo, Desarrollado por Microsoft desde 1993 presentada por primera vez en el sistema operativo Windows95 incrementando la posibilidad de interacción multimedia entre usuario y el computador (Ernest,09).

La alternativa grafica de Microsoft DirectX, acompañada de un largo número de librerías, incluidas de sonido y entrada de audio, es mejorada comparada con OpenGL por la librería Direct3D la última versión DirectX 11 fue usada en Xbox 360, además solo trabaja con computadores que tengan como sistemas operativo Windows Vista, Windows 7.

Figura 2.19 : Logo de DirectX renderizado en el mismo motor grafico.

API y SDK

Aplication Programming Interface - Software Development Kit, interfaz programable de aplicaciones y paquete de librerías para desarrollo de software respectivamente.

El conjunto de características entre Direct3D and OpenGL son muy equivalentes. Los modernos motores de juegos por ejemplo están construidos en modo no real lo que significa que la capa gráfica puede cambiarse por los usuarios para alternar entre OpenGL y DirectX de modo que puedan ver el desempeño de acuerdo a las características de su computador.

Una SDK envuelve un conjunto de librerías de enlace dinámico DLL en windows y SO en Linux, los cuales se enlazan a los programas en tiempo de ejecución, esta unidades o DLL tienen dentro de ellas funciones exportadas para los programadores estas son llamadas funciones API. Para nuestra investigación nuestro SDK se llama

FreeGlut 3, y las funciones API son todas las funciones que tienen como prefijo la palabra glut, podemos apreciar un ejemplo de uso de las API de FreeGLUT.

```
// uso de funciones API FreeGLUT
//
#include <GL/freeglut.h> // Librerias API
#include <cstring>     // Libreria ANSI C++

void PreparaEntorno( const char *lpTitulo )
{
 // funciones nativas del compilador
 char *lpNuevo = new char [255];
 strcpy( lpNuevo, lpTitulo );
strcat( lpNuevo, "-Agregado" );

 // funciones API
glutCreateWindow( 100, 100 );
glutSetWindowTitle( lpNuevo );
glutMainLoop();
}
int main( int argc, char *argv[] )
{
 PreparaEntorno( "Demo de Ventana Creada con APIs" );
 return 0;
}
```

2.2.12 Metodología

Metodología XP (eXtreme Programming)

Esta metodología promueve los siguientes valores:

Comunicación

El eXtreme Programming se nutre del ancho de banda más grande que se puede obtener cuando existe algún tipo de comunicación: la comunicación directa entre personas. Es muy importante entender cuáles son las ventajas de este medio. Cuando dos (o más) personas se comunican directamente pueden no solo consumir las palabras formuladas por la otra persona, sino que también aprecian los gestos, miradas, etc. que hace su compañero. Sin embargo, en una conversación mediante el correo electrónico.

Hay muchos factores que hacen de esta una comunicación, por así decirlo, mucho menos efectiva.

Coraje

El coraje es un valor muy importante dentro de la programación extrema. Un miembro de un equipo de desarrollo extremo debe de tener el coraje de exponer sus dudas, miedos, experiencias sin "embellecer" éstas de ninguna de las maneras. Esto es muy importante ya que un equipo de desarrollo extremo se basa en la confianza para con sus miembros. Faltar a esta confianza es una falta más que grave.

Simplicidad

Dado que no se puede predecir cómo va a ser en el futuro, el software que se está desarrollando; un equipo de programación extrema intenta mantener el software lo más sencillo posible.

Esto quiere decir que no se va a invertir ningún esfuerzo en hacer un desarrollo que en un futuro pueda llegar a tener valor. En el XP frases como "en un futuro vamos a necesitar." o "Haz un sistema genérico de" no tienen ningún sentido ya que no aportan ningún valor en el momento.

Retroalimentación (feedback)

La agilidad se define (entre otras cosas) por la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino. Por este motivo uno de los valores que nos hace más ágiles es el continuo seguimiento o Retroalimentación que recibimos a la hora de desarrollar en un entorno ágil de desarrollo. Esta retroalimentación se toma del cliente, de los miembros del equipo, en cuestión de todo el entorno en el que se mueve un equipo de desarrollo ágil.

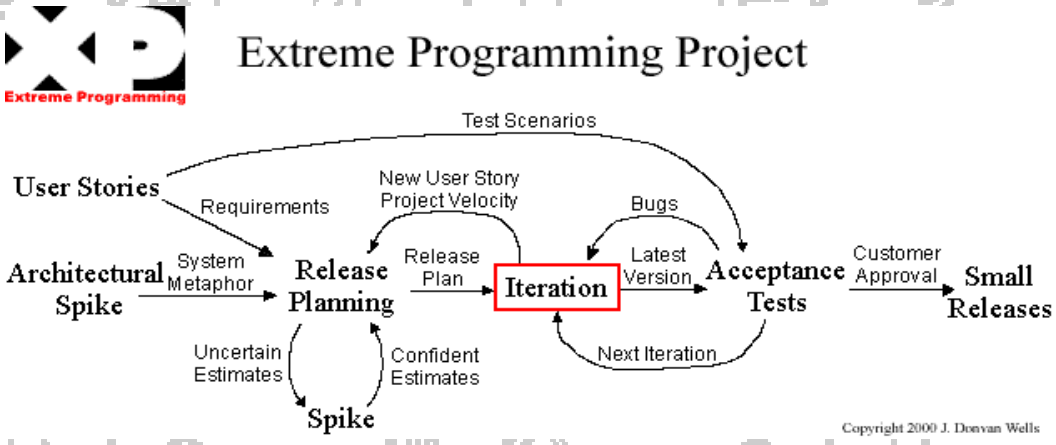


Figura 2.20 : Fases de un proyecto de Extreme Programming.

UML (Unified Modeling Language)

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres

métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la que basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (principalmente Booch, OMT y OOSE). UML ha puesto fin a las llamadas “guerras de métodos” que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. En la figura superior se puede ver cuál ha sido la evolución de UML hasta la creación de UML 1.3, en el que se basa este documento. Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación. En estos conceptos se sigue el método propuesto por Craig

Larman **(Larman, 1999).** El UML está compuesto por los siguientes diagramas:

Área	Vista	Diagramas	Conceptos Principales
Estructural	Vista Estática	Diagrama de Clases	Clase, asociación, generalización, dependencia, realización, interfaz.
	Vista de Casos de Uso	Diagramas de Casos de Uso	Caso de Uso, Actor, asociación, extensión, generalización.
	Vista de Implementación	Diagramas de Componentes	Componente, interfaz, dependencia, realización.
	Vista de Despliegue	Diagramas de Despliegue	Nodo, componente, dependencia, localización.
Dinámica	Vista de Estados de máquina	Diagramas de Estados	Estado, evento, transición, acción.
	Vista de actividad	Diagramas de Actividad	Estado, actividad, transición, determinación, división, unión.
	Vista de interacción	Diagramas de Secuencia	Interacción, objeto, mensaje, activación.
		Diagramas de Colaboración	Colaboración, interacción, rol de colaboración, mensaje.
Administración o Gestión de modelo	Vista de Gestión de modelo	Diagramas de Clases	Paquete, subsistema, modelo.
Extensión de UML	Todas	Todos	Restricción, estereotipo, valores, etiquetados.

Cuadro 2 : Diagramas de UML

2.3 Marco Conceptual

Como marco conceptual se considero importantes los siguientes conceptos:

2.3.1 Entorno

Es la proyección gráfica generada por herramientas de diseño asistido por computador, que permite el poder desplazarse y tener una perspectiva tridimensional de una colección de objetos gráficos que interpretamos como formas sólidas.

Se tiene un espacio subjetivo que formara una imagen visual formado por polígonos y puntos unidos por vértices, lo que comúnmente se conoce como mallas en el caso particular de esta investigación serán mallas generadas por triángulos, todo un conjunto de ellos unidos.

2.3.2 Imagen

Es el conjunto de pixeles que desplegadas en un monitor nos muestran una figura o conjunto de ellas que entendemos como imagen. de forma matemática se representa como una matriz de puntos de color definida por la anchura por altura, teniendo en cada punto de color un componente interno RGB con el que podremos representar hasta 24 millones de colores en un entero de 32 bits.

2.3.3 Mallas

Traducido del término inglés Mesh, usado para clasificar polígonos unidos que permiten formar una figura, las mallas pueden formarse por triángulos o por cuadrados unidos, son el elemento base pre renderización y generación de ambientes virtuales.

2.3.4 Pixel

Es el mínima unidad gráfica, no necesariamente es un punto en la pantalla pues esto dependerá de la resolución del modo de vídeo actual, pudiendo representarse el pixel en una matriz de hasta 4x4 puntos en la pantalla real para el caso de una resolución de 640x480 pixeles en un monitor con resolución de 2048x1024.

2.3.5 Realidad Virtual

Es la representación simulada de un entorno real, sobre un computador apoyado por material audiovisual para la correcta sensación de los mismos, la idea principal es dar al cerebro y los sentidos una experiencia virtual ya sea de paseo, juego o lo que actualmente ha tenido más auge, la simulación de vuelo y el de manejo de vehículos para el entrenamiento de personas nerviosas o sencillamente en escuelas de conducción.

2.3.6 Renderización

Es el proceso de cálculo matemático para generar una forma y textura basada en mallas que representan una superficie o cualquier objeto gráfico. En los programas 3D Max Studio y Maya el proceso de renderización toma bastante tiempo porque se terminan de completar las mallas faltantes, pues mientras más triángulos tenga una malla, más calidad de definición y realidad se podrá obtener.

2.3.7 Teselación

Patrón grafico que permita vecindades de puntos, tiene la forma de un mosaico no uniforme una vez hecha la consolidación de los puntos una vez interconectados.

2.3.8 Triangulación de Delaunay

Es el cálculo matemático basado en el completado de áreas usando triángulos, para el completado de puntos y generación de mallas, es una técnica efectiva y lo es porque es sencilla, mantiene el concepto **"lo sencillo es bello".**

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1 Ubicación

El presente trabajo de investigación se desarrolló en el departamento de Puno.

3.2 Población

La población estuvo conformada por los Peritos Balísticos del Departamento de Criminalística – Puno.

3.3 Muestra

La muestra de estudio fue a cuatro especialistas que trabajan con diseño 3D del Departamento a - Puno, quienes realizaron las pruebas de sistema y el llenado de los cuestionarios.

El tipo de muestreo que se uso fue el muestreo no probabilístico, ya que permitió tomar una muestra a criterio del investigador.

3.4 Método de recolección de datos

Para la recolección de datos se hizo dos cuestionarios aplicados a cuatro especialistas que trabajan con diseño 3D , los cuales fueron:

- Métricas de Calidad de Software. (Anexo A)

- Factores de Calidad. (Anexo B)

3.5 Metodología de Desarrollo

3.5.1 Programación Extrema

Para el desarrollo del Software de compresión se aplicó la metodología de la Programación Extrema (Extremme Programming - XP) que se adapta hoy en día perfectamente al desarrollo del ciclo de vida del Software y para el modelado del Software se usó el Lenguaje Unificado de Modelamiento (Unified Modeling Languaje - UML).

Figura 3.1 : Fases de la Metodología XP.

Las fases de la programación extrema se divide en cuatro fases :

1. **Análisis:** La metodología XP plantea en análisis como un permanente diálogo entre la parte empresarial y técnica del proyecto, en la que los primeros decidirán el alcance ¿Qué es lo realmente necesario del proyecto?, la prioridad qué debe ser hecho en primer lugar, la composición de las versiones que debería incluir cada una de ellas y la flecha de las mismas en cuanto a los técnicos, son los responsables de estimar la

duración requerida para implementar las funcionalidades deseadas por el cliente, de informar sobre las consecuencias de determinadas decisiones, de organizar la cultura de trabajo y finalmente de realizar la planificación detallada dentro de cada versión. XP no solo es un método centrado en el código que lo es, sino que sobre todo es un método de gestión de proyectos software **(Grady Booch, 1999).**

2. **Diseño**: El Propósito del diseño es de crear una arquitectura para la naciente implementación, el diseño arquitectural sólo puede comenzar una vez que el equipo tenga un entendimiento razonable de los requerimientos del sistema. El diseño, como el análisis, nunca termina realmente hasta que el sistema final es entregado. Durante esta fase se alcanza un cierto diseño y al establecer políticas para diversos problemas tácticos.

   El diseño se enfoca en la estructura, estática y dinámica, su propósito principal es de crear el esqueleto concreto del sistema sobre el cual todo el resto de la implementación se basa **(Grady Booch, 1999)**

3. **Desarrollo**: Esta etapa debe reunir las siguientes características o cualidades :

   - El software está siempre disponible
   - Se debe escribir código de acuerdo a los estándares
   - Desarrollar la unidad de pruebas primero
   - Todo el código debe programarse por parejas
   - Integrar frecuentemente

- Todo el código es común a todos

4. **Prueba**: Todo el código debe ir acompañando, Los casos de prueba se escriben antes que el código. Los desarrolladores escriben pruebas unitarias y los clientes especifican pruebas funcionales.

3.5.2. Herramientas CASE

Para el Modelamiento, el análisis y diseño del software se hizo un análisis, atendiendo a las características que presenta cada una de ellas, entre dos de las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) podemos describir:

- *Rational Rose Software 2003*
- *DIA*
- *GEDIT, Dreamweaver, Core*

3.6 UML (Lenguaje Unificado de Modelado)

UML es un lenguaje estándar que sirve para escribir los planos del software, puede utilizarse para visualizar , especificar, construir y documentar todos los artefactos que componen un sistema .

3.6.1 Diagrama de Casos de Uso

En un diagrama de casos de uso consta de los siguientes elementos:

- Actor.
- Casos de Uso.

- Relaciones de Uso, Herencia y Comunicación.

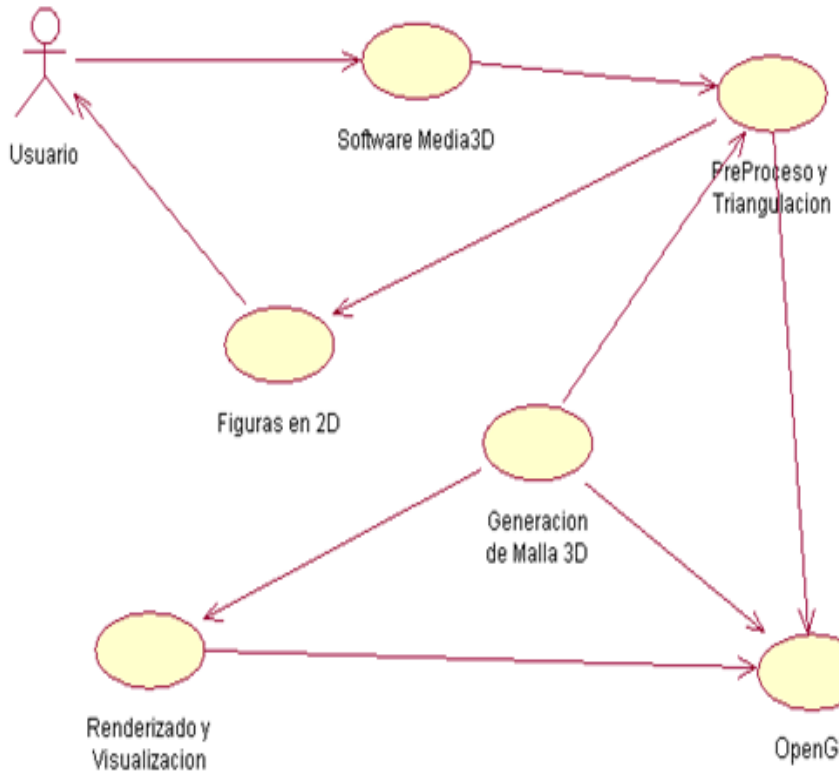


Figura 3.2 : Grafico de Casos de Uso

El usuario al ejecutar la aplicación ingresara la información de figuras en 2D, ya sea dibujando o por apertura de archivos, el software realiza un proceso de pre proceso y triangulación para la generación de la malla 3D a su vez este servirá para el proceso de renderización y visualización que será soportado para la visualización de los mismo usando OpenGL, para poder modelar y obtener perspectiva grafica de la malla y el sólido una vez renderizados, debe tenerse en cuenta que la colección de puntos varía de acuerdo a la cantidad de datos ingresados, es decir no es una estructura fija sino dinámica.

3.6.2 Diagrama de Clases

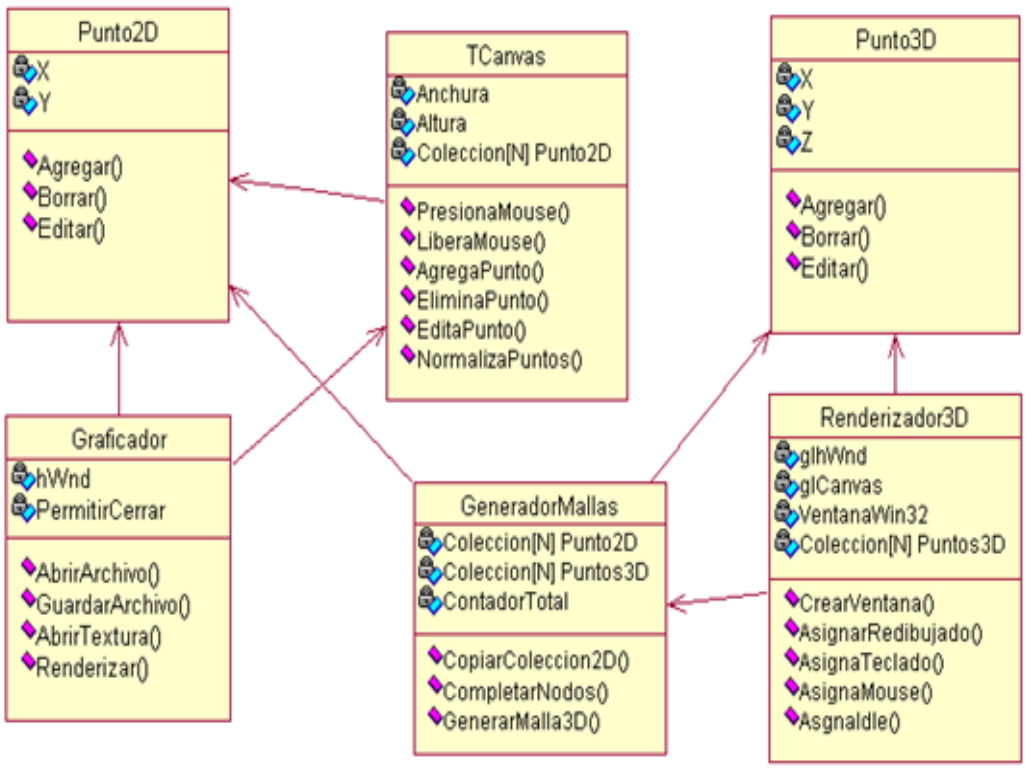


Figura 3.3 : Grafico de Diagramas de Clases

Presentamos las 6 clases que interviene en el desarrollo del software de modelamiento así como en el proceso de conversión y generación de una malla 2D a una malla3D.

- Clase Punto2D : Almacenara los elementos de la colección.
- Clase Punto3D : Almacenara los nuevos puntos generados.
- Clase TCanvas : Desplegaremos sobre ella gráficamente los puntos contenidos en la colección Punto 2D[N].
- Clase Graficador : Administra a la clase TCanvas
- Clase Generador de Mallas: Almacena el algoritmo de conversión.
- Clase Renderizador 3D: Calcula los vectores normales para graficar con OpenGL.

3.6.3 Diagrama de Secuencias

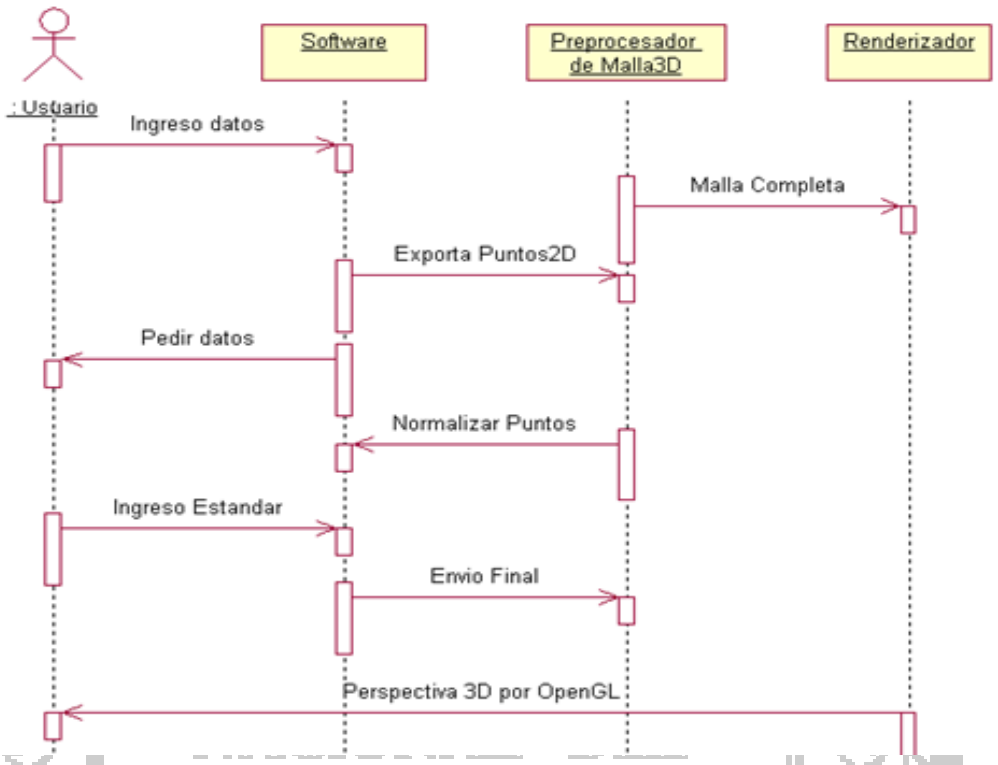


Figura 3.4 : Grafico de Diagramas de Secuencia

La secuencia que el usuario al ejecutar la aplicación ingresara la información de figuras en 2D, dibujando o por apertura de archivos, el software realiza un proceso de pre proceso y triangulación para la generación de la malla 3D a su vez este servirá para el proceso de renderización y visualización que será soportado para la visualización de los mismo usando OpenGL para poder modelar y obtener perspectiva grafica de la malla y el sólido una vez calculador, debe tenerse en cuenta que la colección de puntos varía de acuerdo a la cantidad de datos ingresados, es decir no es una estructura fija sino dinámica.

3.6.4 Diagrama de Actividades

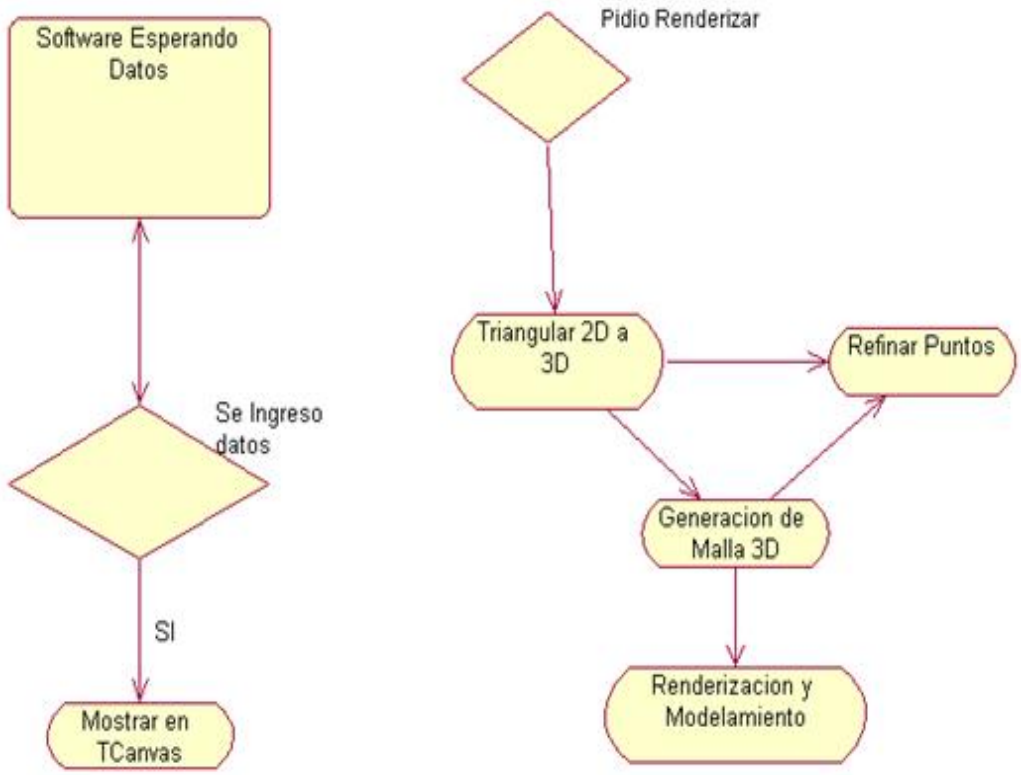


Figura 3.5 : Grafico de Diagramas de Actividades

La secuencia de actividades permite ver de forma clara y sencilla el proceso de ingreso de datos y la posterior generación de las mallas y por supuesto el renderizado.

Cabe destacar que se simplifica el proceso para comprender de forma rápida e intuitiva el completo desarrollo del software sin perder la perspectiva la complejidad que se requirió para desarrollar los algoritmos.

3.6.5 Diagrama de flujo

Un Diagrama de Flujo representa la esquematización gráfica de un algoritmo , el cual muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema . Su correcta construcción es

sumamente importante porque a partir del mismo se escribe un programa en algún Lenguaje de Programación. Si el Diagrama de Flujo está completo y correcto, el paso del mismo a un Lenguaje de Programación es relativamente simple y directo.

Un diagrama de flujo siempre tiene un único punto de inicio y un único punto de término. Además, todo camino de ejecución debe permitir llegar desde el inicio hasta el término.

Las siguientes son acciones previas a la realización del diagrama de flujo:

- Identificar las ideas principales a ser incluidas en el diagrama de flujo.
- Deben estar presentes el dueño o responsable del proceso, los dueños o responsables del proceso anterior y posterior y de otros procesos interrelacionados.
- Definir que se espera obtener del diagrama de flujo.
- Identificar quien lo empleara y como.
- Establecer el nivel de detalle requerido.
- Determinar los límites del proceso a describir.

Podemos expresar el proceso de obtención de los diversos elementos geométricos, desde la aplicación base hasta la generación del polinomio de **(x; y; z)** una vez triangulados.

- Cargar Módulos 2D y 3D.
- Extraer características de polígonos y vecindad de puntos.

- Triangulación y Completado del polígono 2D para la obtención del polígono 3D.
- Determinación de la proyección y proceso de renderizado.
- Mostrar el ambiente completado y renderizado.

Ventajas de los diagramas de flujo

- Favorecen la comprensión del proceso al mostrarlo como un dibujo. El cerebro humano reconoce muy fácilmente los dibujos. Un buen diagrama de flujo reemplaza varias páginas de texto.
- Permiten identificar los problemas y las oportunidades de mejora del proceso. Se identifican los pasos, los flujos de los reproceso, los conflictos de autoridad, las responsabilidades, los cuellos de botella y los puntos de decisión.
- Muestran las interfaces cliente-proveedor y las transacciones que en ellas se realizan, facilitando a los empleados el análisis de las mismas.
- Son una excelente herramienta para capacitar a los nuevos empleados y también a los que desarrollan la tarea, cuando se realizan mejoras en el proceso.

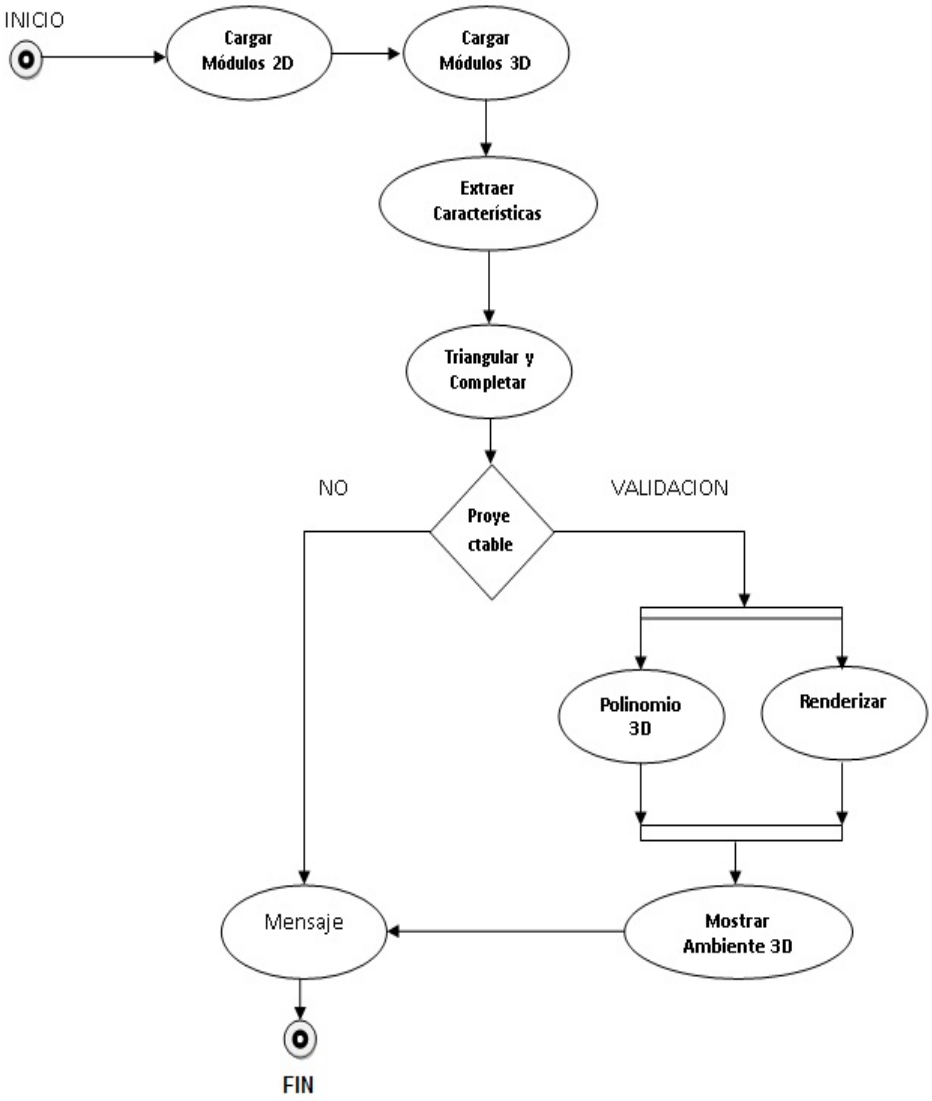


Figura 3.6 : Grafico de Diagramas de Flujo

3.7 Material Experimental

Una vez terminado el análisis del sistema el siguiente paso fue definir el diseño del sistema , precisando claramente el marco conceptual teórico del cual se partirá para el desarrollo del mismo.

3.7.1 Hardware

- Computador con Micro-Procesador Intel Core i5 de 2.8GHz, RAM de 4GB.
- Disco duro externo Samsung de 500-GB (Backups de avances)

3.7.2 Software

En la investigación se usó software de compilador C/C++, C# para poder obtener un programa portable a otros sistemas operativos, además de un conjunto de librerías gráficas para realizar la simulación en 3 dimensiones.

- Software Nero Burner para grabar y obtener imágenes ISO.
- Software DIA – editor de diagramas
- Software de modelamiento Rational Rose 2003
- CodeGear RAD Studio 2009

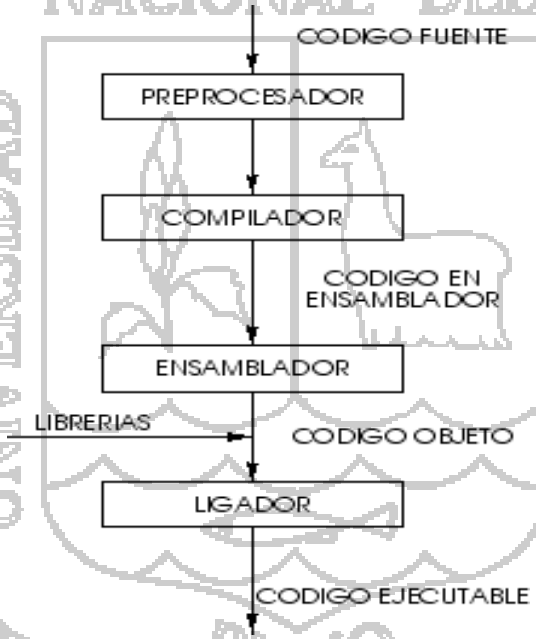


Figura 3.7 : Modelo de compilación en C.

3.8 Métricas de Calidad de McCall

El cálculo de cada factor de calidad se realizó con la fórmula de McCall

$$Fq = (C1 \times M1) + (C2 \times M2) + \ldots + (Cn \times Mn)$$

Dónde:

Fq : es un factor de calidad del software.

Cn : son coeficientes de regresión

M_n : son las métricas que afectan al factor de calidad

Para calcular el coeficiente de regresión se utilizó la formula

$$Cn = (M^T \times M)^{-1} \times (M^T \times M_Q)$$

Dónde :

M^T : es la respuesta a la matriz de las métricas.

M : es la matriz de las métricas

F_Q : es la matriz de los factores

3.9 Factores de calidad de McCall

3.9.1 Corrección

Al factor de corrección pertenecen a las métricas de completitud, consistencia y trazabilidad.

3.9.2 Fiabilidad

Al factor de fiabilidad pertenecen las métricas de exactitud, complejidad, consistencia, tolerancia a errores, modularidad y complicidad.

3.9.3 Eficiencia

Al factor de eficiencia pertenecen las métricas de concisión, eficiencia de ejecución y operatividad

3.9.4 Integridad

Al factor integridad pertenecen las métricas de facilidad de auditoría, instrumentación y seguridad.

3.9.5 Flexibilidad

Al factor de flexibilidad pertenecen las métricas de complejidad, concisión, consistencia, capacidad de expansión, generalidad, modularidad. Auto documentación y simplicidad.

3.9.6 Capacidad de Pruebas

Al factor de capacidad de pruebas pertenecen las métricas de facilidad de auditoria, complejidad, instrumentación, modularidad, auto documentación y simplicidad

3.9.7 Reusabilidad

Al factor reusabilidad pertenecen las métricas de generalidad, simplicidad, modularidad, auto documentación e independencia del sistema.

3.9.8 Usabilidad

Al factor de usabilidad pertenecen las métricas de operatividad y facilidad de formación.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1 Diseño

Para desarrollar el software de nuestro trabajo de investigación se usó el entorno de Desarrollo **CodeGear RAD Studio 2009**, que es uno de los entornos de desarrollo más potentes comparable con **Microsoft Visual Studio .NET 2010.**

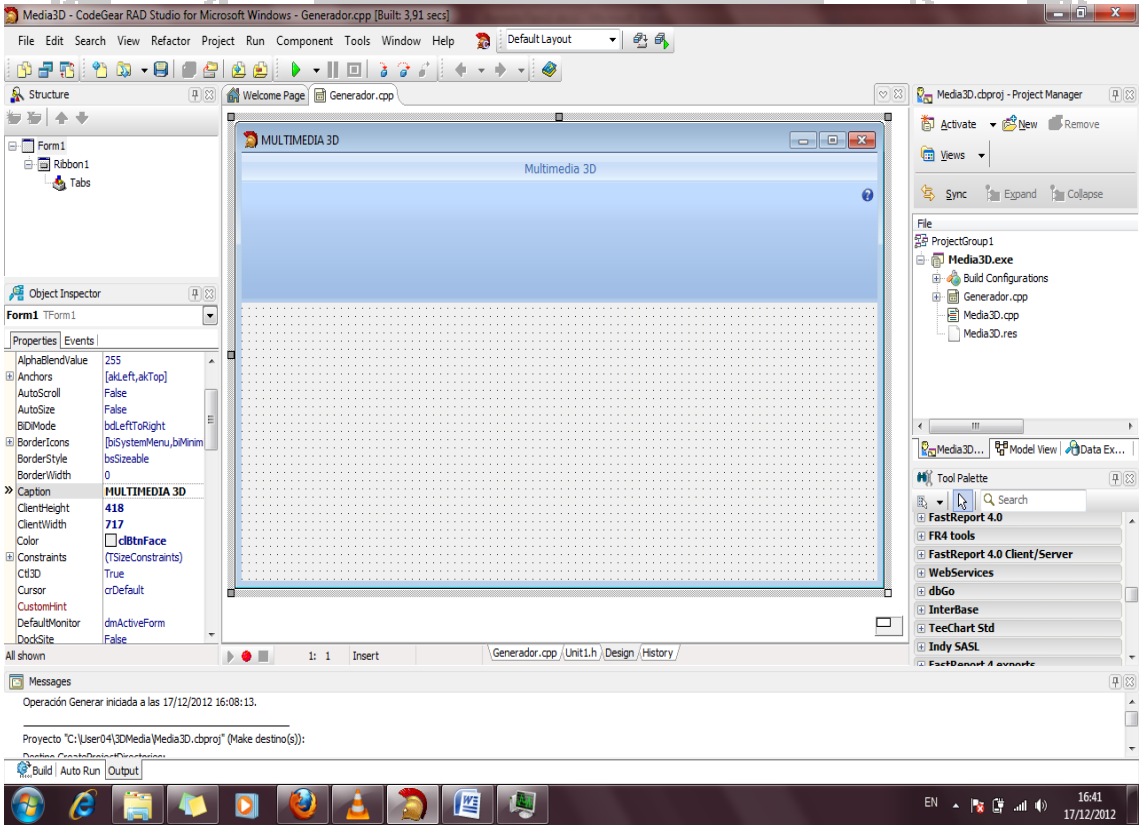


Figura 4.1 : Entorno de desarrollo CodeGear RAD Studio 2009

4.2 Interfaz Gráfica de Usuario

4.2.1 Elementos utilizados para el desarrollo

CONTROL	PROPIEDAD	VALOR
TForm	Name	Multimedia
	Caption	Multimedia 3D
TRibbon	Style	Office2007
	Tabs	✓ Principal
	NestedTabs	✓ Archivo ✓ Acabado ✓ Generar
TCanvas	Name	Canvas2D
	PaintMode	Manual
TButton	TArray TCollection	✓ Nuevo ✓ Abrir ✓ Guardar ✓ Generar

Cuadro 4.1 : Cuadro de los contenidos de los controles de Media3D

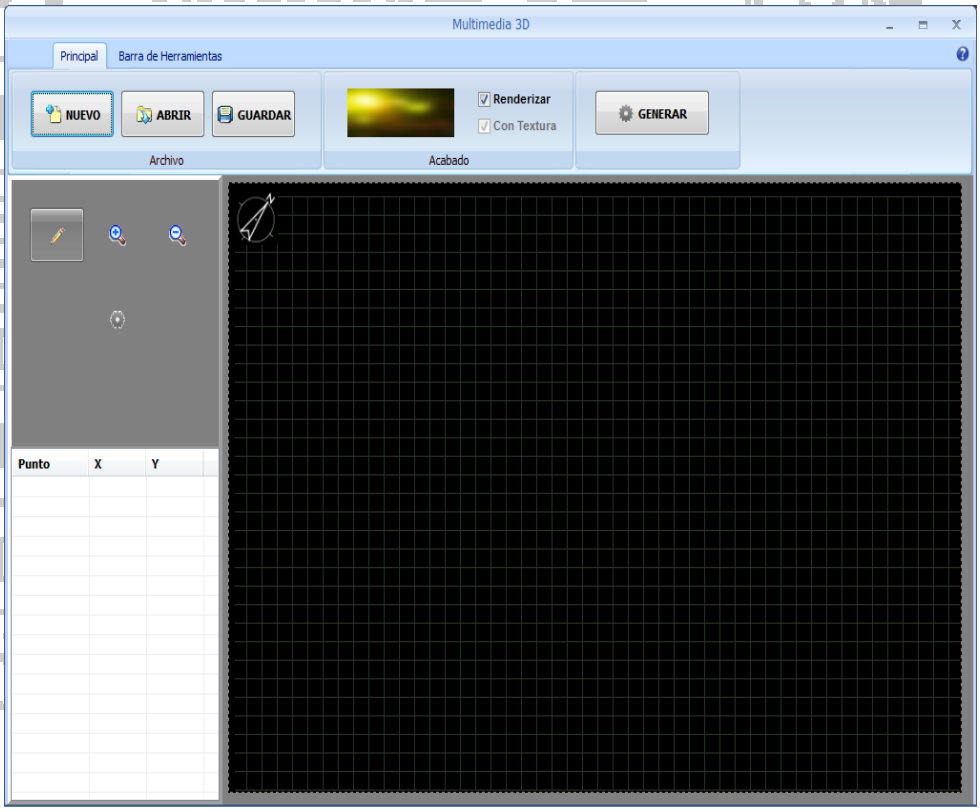


Figura 4.2 : Interfaz gráfica de Media 3D.

4.2.2 Elementos externos

- OpenGL : Librería de funciones gráficas para poder modelar los elementos gráficos generados, como las mallas y las texturas una vez renderizadas, para poder usar estas funciones es necesario agregar las librerías de forma manual como se muestra a continuación.

```
#include <GL/gl.h>

#include <GL/glu.h>
```

- Windows GDI : Librería de funciones graficas que están encapsuladas en el control **TCanvas**, por lo que será necesario acceder a los elementos miembro del objeto y estos usan a su vez los elementos de graficas como Elipses, Rectángulos, Line.
- Administración de Dispositivos : Control de Mouse y Teclado para permitir la movilidad del objeto renderizado con fines de modelamiento y obtención de perspectiva gráfica y visual sobre el objeto 2D convertido por nuestro software a una malla 3D y posteriormente renderizado para obtener un sólido.

4.3 Implementación

Concluido con la parte del diseño y desarrollo del software se realizó la correcta implementación e integración de los dispositivos tanto físicos como lógicos es decir software y hardware, los cuales realizan un excelente trabajo dando un adecuado uso al hardware es decir optimizando el rendimiento y aplicando soporte al sistema como una parte de la fase de implementación.

Se realizó la instalación del software el cual fue previamente empaquetado y se obtuvo el instalador para tal efecto se utilizó Install shield 10.5 Profesional, no siendo este un requisito fundamental, ya que en el mercado existen numerosos pack para crear instaladores además suelen ser tipo shareware, freeware, trialware, dependiendo del programador cuál de las muchas operaciones realizar ya que se debe de tener conocimiento para poder realizar scripts con la finalidad de poder desarrollar un buen instalador que además de ser Flexible sea portable.

Implementación de software

- Sistema Operativo Windows XP, Windows 7.
- C++ Builder RAD 6.0 Instalado.
- glPak 1.2 (Integrado al proyecto).
- Librerías Opengl32 Nativas. (Presente en Win32).

4.4 Prueba del sistema

Durante el periodo de prueba el cual tomo un tiempo considerable se estableció que el sistema presentaba varios tipos de estructuras cíclicas erráticas que afectaban en la operación del usuario, tanto en la parte lógica como en la parte externa de este ya que los cuadros de dialogo y dispositivos de contextos no reflejaban en su totalidad el comportamiento del usuario durante el desplazamiento.

Es así que se sometió nuevamente a pruebas el algoritmo del cual se fueron puliendo los errores conforme estos se iban presentado, dando así

cumplimiento a esta de diseño muy importante por cierto que tiene como nombre Fase de Prueba.

No está demás explicar que como todo software este se encuentra predispuesto a cualquier tipo de cambio que el usuario requiera con la finalidad de poder optimizar el mismo, esto incluye también las actualizaciones que por tiempo de vida de uso se deben de realizar periódicamente.

4.5 Resultados

4.5.1 Metas

Desarrollar una técnica para incrustar un visor OpenGL sobre una ventana grafica Windows, sin usar la librera GLUT, por ser inadecuadas para nuestra aplicación e incrustarla al Proyecto Media 3D, mediante las sentencias:

```
//-----------------------------------------------
void AlCrearFormulario()
  {
    TRect mRect = GetClientRect();
    glhWnd = mglCreateViewAsChild( this->Handle, 0, 0,
      mRect.Width(), mRect.Height() );
  // Send the Handler
  mglCreateHandler( glhWnd );
  mglDisplayFunc( OnV3dDisplay );
  }
  //-----------------------------------------------
  void AlRenderizar()
  {
    glClearDevice();
    ... funciones de dibujo
    mglSwapBuffers();
    }
  //-----------------------------------------------
```

- Desarrollar la interfaz principal, cumpliendo un diseño confortable y ergonómico.

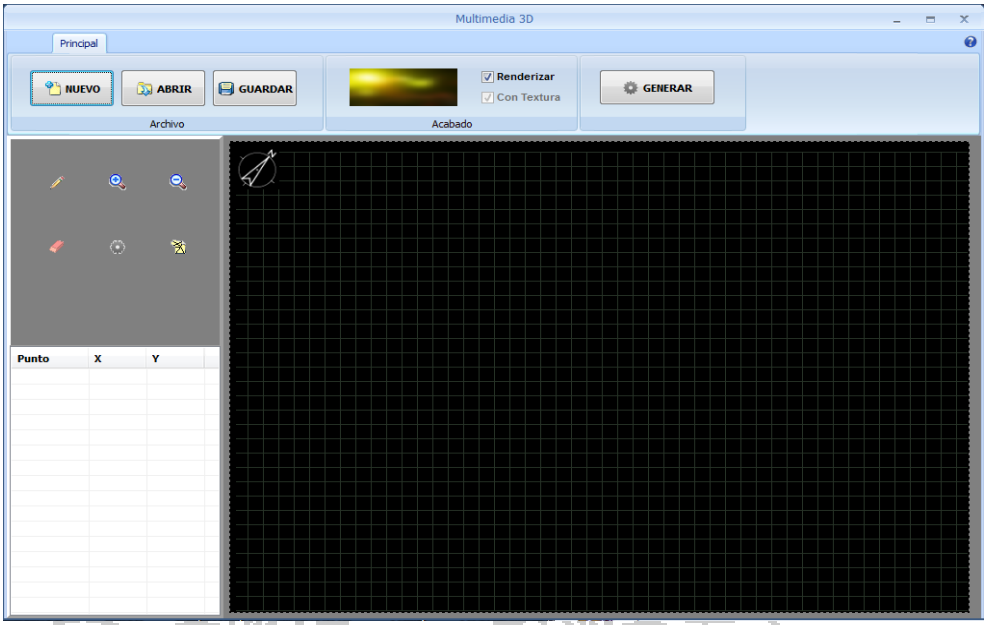


Figura 4.3 : Modulo principal de la aplicación de Media3D

- Analizar e implementar una herramienta de diseño gráfico en 2D, integrado al algoritmo de Triangulación de Delaunay para la generación del enmallado 3D.

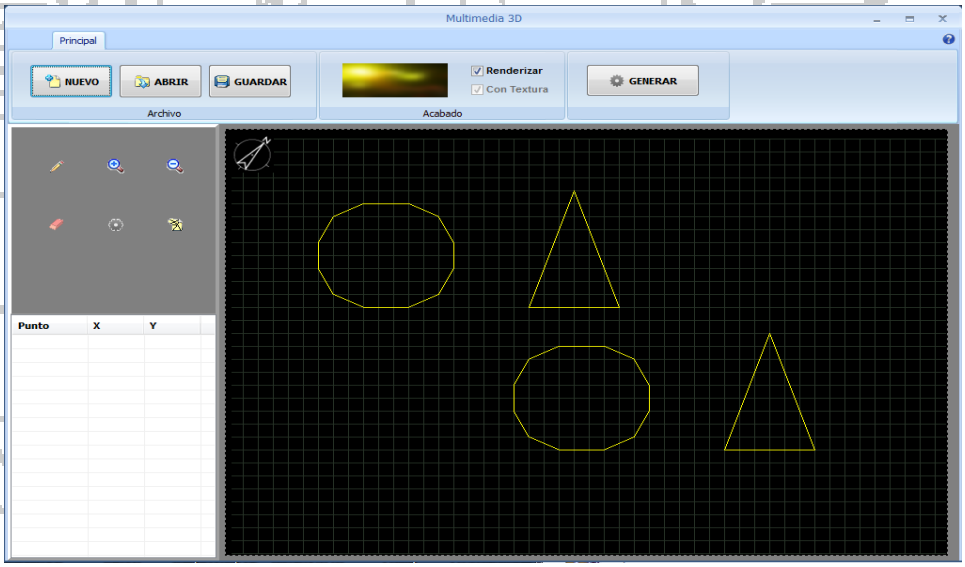


Figura 4.4 : Modulo graficador en 2 dimensiones en Media3D

- Implementar el algoritmo de Delaunay con el fin de generar mallas trianguladas, con el fin de poder generar un sólido con la

posibilidad de texturas modificables y agregamos la perspectiva de movimiento.

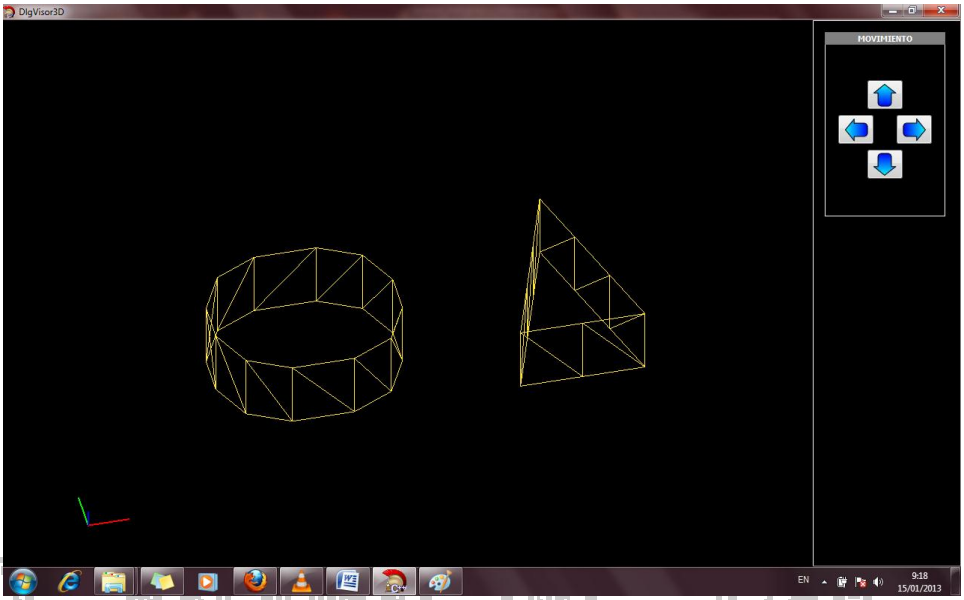


Figura 4.5 : Mallas resultantes en Media3D

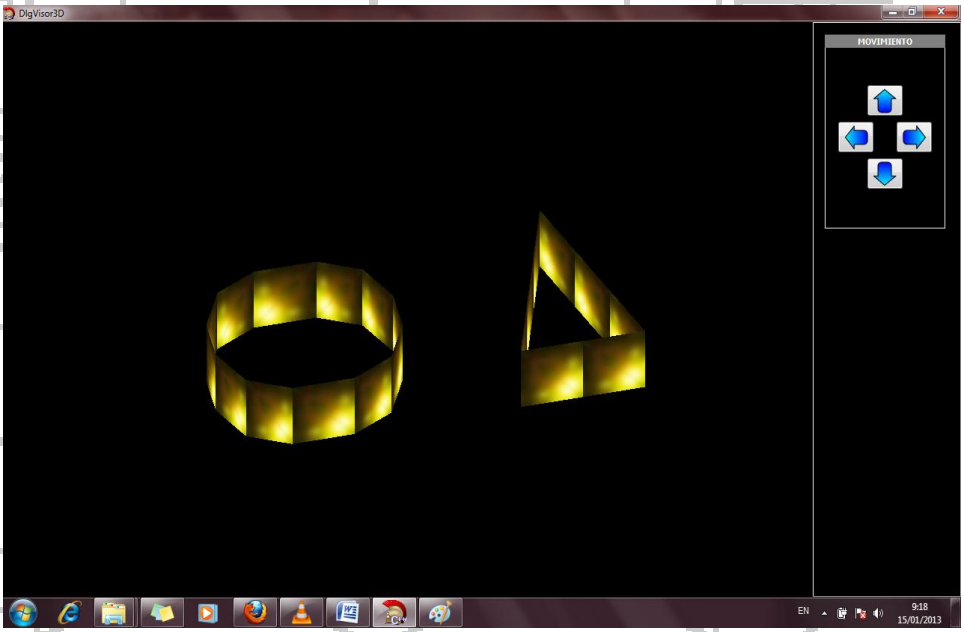


Figura 4.6 : Entorno renderizado en Media3D

4.5.2 Proceso de Refinamiento de Delaunay

El proceso de completado de puntos puede explicarse gráficamente:

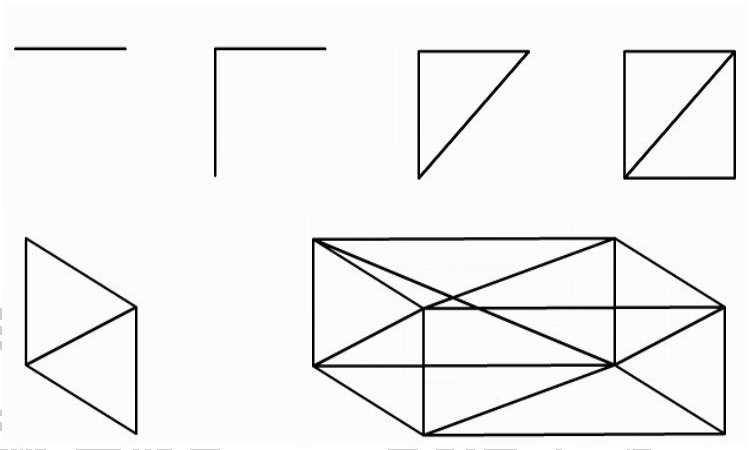


Figura 4.7: Proceso de Refinamiento de Delaunay

4.6 Métricas de calidad de McCall

4.6.1 Factores de calidad de McCall

Se aplicaron los factores de métricas de calidad de McCall para determinar si el sistema fue de alta calidad.

Para poder aplicar las pruebas de calidad de McCall se realizó una encuesta a cinco usuario, las respuestas hechas por los usuarios, los resultados de estas encuestas están representados en las tablas siguientes, las respuestas hechas por los usuarios están en un rango de 0 (bajo) a 10 (alto).

Tabla Nº 01. Puntuación de métricas según usuarios

METRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Exactitud	10	9	8	10	9.25
Compleción	8	7	9	6	7.50
Concisión	8	7	8	7	7.50
Consistencia	7	7	9	8	7.75
Estandarización de datos	8	8	9	6	7.75
Tolerancia de errores	8	7	8	7	7.50
Eficiencia de ejecución	9	8	8	7	8.00
Capacidad de Expansión	8	7	7	6	7.00
Generalidades	7	6	7	8	7.00
Instrumentación	8	9	9	9	8.75
Modularidad	6	8	7	8	7.25
Operatividad	7	8	8	7	7.50
Seguridad	8	9	7	7	7.75
Auto documentación	6	8	9	8	7.75
Simplicidad	7	7	8	7	7.25
Independencia del sistema	9	9	9	7	8.50
Trazabilidad	7	8	8	8	7.75
Facilidad de formación	8	7	8	8	7.75

Fuente: cuestionario de métricas de calidad de McCall

TABLA 02. Puntuación de Factores de calidad según Usuarios

FACTOR	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4
Corrección	8	7	9	7
Fiabilidad	9	10	9	9
Eficiencia	9	8	7	9
Integridad	9	8	8	8
Flexibilidad	8	8	9	8
Capacidad de pruebas	9	8	8	9
Reusabilidad	8	9	10	9
Usabilidad	9	8	10	8

Fuente: Cuestionarios de factores de calidad de McCall

A continuación se muestra el cálculo de cada factor de McCall:

4.6.2 Corrección

Los coeficientes de regresión obtenidos mediante la fórmula de McCall son:

$C_1 = 0.74$ $C_2=0.26$ $C_3 = 0$

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Compleción	8	7	9	6	7.5
Consistencia	8	7	9	9	8.25
Trazabilidad	7	8	9	9	8.25

Reemplazando tenemos:

$$Co = (0.\ 74 \times 7.5) + (0.26 \times 7.5) + (0 \times 7.75) = 7.\ 5$$

El factor de corrección tiene una puntuación de 7.5, el cual indica que el factor de calidad es aceptable.

4.6.3 Fiabilidad

Los coeficientes de regresión obtenidos mediante la fórmula de McCall son:

$C_1 = 0.02$ $C_2 = 0.01$ $C_3 = 0.03$ $C_4 = 0$

$C_5 = 0$ $C_6 = 1.06$

MÉTRICA	USUARIO	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Exactitud	10	9	8	10	9.25
Compleción	8	7	9	6	7.5
Consistencia	8	7	9	9	7.75
Tolerancia de Errores	8	7	8	6	7.5
Modularidad	6	7	6	8	7.25
Simplicidad	8	9	8	8	7.25

Reemplazando tenemos:

Fi = (0.02 x 9.25) + (0.01 x 7.5) + (0.03 x 7.75) + (0 x 7.5) + (0 x 7.25) + (1.06 x 7.25) = 8.18

El factor de fiabilidad tiene una puntuación de 8.18, en una escala de 0 a 10, lo cual indica que el factor de calidad es alto.

4.6.4 Eficiencia

Los coeficientes de regresión obtenidos mediante las fórmulas de McCall son:

$C_1 = -4.82E-18$ $C_2 = -2.021E-16$ $C_3 = 1$

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Concisión	5	7	8	7	7.50

Eficiencia de Ejecución	9	10	8	8	8.00
Operatividad	8	9	9	8	7.50

Entonces tenemos:

Ef = ((-4.82E - 18) x 7.50) + ((-2.021E - 16) x 8.00)+ (1x7.50) = 8.5

El factor de eficiencia tiene una puntuación de 8.5 en una escala de 0 a 10, lo cual indica que el factor de calidad es Alto.

4.6.5 Integridad

Los coeficientes de regresión obtenidos mediante la fórmula de McCall son:

$C_1 = 0.97$ $C_2 = 0.82$ $C_3 = -0.74$

MÉTRICA	USUARIO	USUARIO	USUARIO	USUARIO	PROMEDIO
Instrumentación	8	8	7	8	8.75
Seguridad	8	9	7	9	7.75
Facilidad de auditoria	7	8	9	9	8.25

Reemplazando tenemos:

In = (0.97x8. 75)+ (0,82x7. 75)+ (-0.74x8.25) = 8.25

El factor de Integridad tiene una puntuación de 8.25, en una escala de 0 a 10, lo cual indica que el factor de calidad es alto.

4.6.6 Flexibilidad

Los coeficientes de regresión obtenidos son:

$C_1 = 0.35$ $C_2 = 0.21$ $C_3 = 0$ $C_4 = 0.27$

$C_5 = 0$ $C_6 = 0$ $C_6 = 0.2 C_6 = 0$

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Compleción	8	7	9	6	7.50
Concisión	5	7	8	7	7.50
Consistencia	8	7	9	9	7.75
Capacidad de Expansión	9	8	9	10	7.00
Generalidad	8	7	8	7	7.00
Modularidad	6	7	6	8	7.25
Auto documentación	8	9	7	8	7.75
Simplicidad	8	9	8	8	7.25

Reemplazando tenemos:

Fl = (0. 35 x 7. 5) + (0.21 x 7. 5) + (0 x 7.75) + (0.27 x 7) + (0 x 7) +

(0 x 7. 25) + (0.2 x 7.75) + (0 x 7.25) = 7.6

El factor de flexibilidad tiene una puntuación de 7.6 en una escala de 0 a 10, lo cual indica que el factor de calidad aceptable.

4.6.7 Capacidad de Pruebas

Los coeficientes de regresión obtenidos son:

$C_1 = 1.41$ $C_2 = 0.71$ $C_3 = 0.71$ $C_4 = 0$

$C_5 = -1.70$ $C_6 = 0$

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Facilidad de Auditoria	7	8	9	9	8.25
Compleción	8	7	9	6	7.50
Instrumentación	8	8	7	8	8.75
Modularidad	6	7	6	8	7.25

Auto documentación	8	9	7	8	7.75
Simplicidad	8	9	8	8	8.25

Reemplazando tenemos:

Cp = (1.41 x 8.25) + (0. 71 x 7. 5) + (0. 71 x 8.75) + (0 x 7. 25) +

(-1.70 x 7.75) + (0 x 8.25) = 9.99

El factor de capacidad de pruebas tiene una puntuación de 9.99 en una escala de 0 a 10, lo cual indica que el factor de calidad es Alto.

4.6.8 Reusabilidad

Los coeficientes de regresión obtenidos mediante las fórmulas de McCall son: C_1 = 014 C_2 = 1.03 C_3 = 1.08

C_4 = - 0.97 C_5 = O

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Generalidad	8	7	8	7	7.00
Modularidad	6	7	6	8	7.25
Auto documentación	8	9	7	8	7.75
Simplicidad	8	9	8	8	7.25
Independencia del sistema	8	9	9	7	8.50

Reemplazando tenemos:

Re = (0.14 x 7) + (1.03 x 7.25) + (1.08 x 7.75) + (-0.97 x 7.25) +

(0 x 8.50) = 9.8

El factor de Reusabilidad tiene una puntuación de 9.8 en una escala de 0 a 10, lo cual indica que el factor de calidad es Alto.

4.6.9 Usabilidad

Los coeficientes de regresión obtenidos mediante las fórmulas de McCall son:

$C_1 = 1.51$ $C_2 = -0.49$

MÉTRICA	USUARIO 1	USUARIO 2	USUARIO 3	USUARIO 4	PROMEDIO
Operatividad	8	9	9	8	7.50
Facilidad de formación	8	9	8	9	7.75

Reemplazando tenemos:

$$Us = (1.51 \times 7.50) + (- 0.49 \times 7.75) = 7.53$$

El factor de usabilidad tiene una puntuación de 7.53 en una escala de 0 a 10, lo cual indica que el factor de calidad es Alto.

4.6.10 Cálculo del Factor de Calidad con la Fórmula de McCall

Para hallar la puntuación final de calidad de software se promediaron los resultados de los factores:

$$Pcs = \frac{Co + Fi + Ef + In + Fl + Cp + Re + Us}{8}$$

$$Pcs = \frac{7.5 + 8.18 + 8.5 + 8.25 + 8.2 + 7.6 + 9.8 + 7.53}{8} = 8.19$$

El valor de la puntuación de calidad de software final es de 8.19, en la escala de 0 a 10 este valor indica que factores que miden la calidad de software son de alto grado.

CONCLUSIONES

- Concluimos que el desarrollo e implementación de software multimedia mejora la generación de ambientes 3D a partir de figuras 2D aplicando el método de triangulación de Delaunay

- Concluimos que para el desarrollo e implementación del algoritmo de transformación y rotación de imágenes se aplicó la geometría computacional, obteniéndose de esta forma un aspecto sumamente realista al sentido visual.

- Concluimos que la implementación del algoritmo de generación de mallas basado en la triangulación de Delaunay funciona adecuadamente en la renderización de imágenes.

- Concluimos que el desarrollo del software Media3D permite transformar de manera óptima las figuras 2D en ambientes 3D.

RECOMENDACIONES

- Se recomienda investigar sobre la reconstrucción de sólidos no uniformes e incrementar la investigación con generación de mallas y su refinamiento, en especial para la sectorización de mallas a través de grafos por vecindades.
- Se recomienda investigar sobre segmentación de imágenes y generación de mallas basadas en imágenes topográficas y así poder distorsionarlas y darle una mejor aplicabilidad a la generación de mallas, por lo que esta investigación aporta en estos conceptos base, pudiendo llegar a plasmar mallas 3D no uniformes para áreas topográficas.

BIBLIOGRAFÍA

Beck, K. (2005). *Una Explicación de la Programación Extrema: Aceptar el Cambio*. Addison-Wesley Iberoamericana Espana, S.A., 1th edition.

Chromatic Jane (2004). *Guía de Bolsillo de Programación Extrema p.10*. O'Reilly, 1th edition.

Cuadros, D. A. (Mayo 2001). *Beta conexión: Familia de objetos 3D generados apartir de secciones planares*.

D. Ritchie, B. K. (1979). *El Lenguaje de Programación C*. Elsevier Science Inc., New York, NY, USA.

Bruce Kenneth (2009). *La Metodología RUP para el Desarrollo de la Ingeniería de Software*, Prentice-Hall Iberoamérica España, 2da Edición.

Knuth, D. E. (1984). T*he TEX Book. Computers & Typesetting*. Addison-Wesley, Reading, Massachusetts, 15th edition. Reprinted as Vol. A of Computers & Typesetting, 1986.

M0RPhEuS, N. (2008). *The NeHe OpenGL Lessons*. Independent, 1th edition.

Natividad Grandon, D. A. (2007). *3D object reconstruction with calibrated images. Ingeniería*, Revista Chilena de Ingeniera.

Nilsson, J. (2001). *Inteligencia Artificial. IA*. Editorial Concepción Fernadez - Madrid.

Pazera, E. (2009). *Focus on 2D in DirecX and Direct3D.* Microsoft Press, 1th edition.

Roger S. Pressman (2002), *Ingeniería del Software, Un enfoque práctico*, Mc Graw Hill, R.S. Pressman&Associates, Inc, Quinta Edición en Español (Dirigido por Luis Joyanes Aguilar – Madrid España).

Rational Software White Paper (2010. *Rational Unified Process for Systems Engineering RUP SE1.1*, Rational The Software Development Company.

Rumbaugh, James, Grady Booch and Ivar Jacobson (1999). *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999.

Rafael, C. (1996). T*ratamiento digital de imágenes*. Computación Grafica. Adison-Wesley Iberoamerica, 2nd edition.

Ruiz, L. R. (2002). *Pequeño programa de dibujo* - mini3D.

Schuller, D. (2011). *Programación de Juegos en C# : Para la Creación Seria de Juegos*. Course Technology PTR, Reading, Massachusetts, 1st edition. Apart of Cengage Learning.

Tejada, D. E. (2008). *Towards Meshless Volume Visualization*. PhD thesis, University of Stuttgart, Germany.

Vilca, M. O. A. (2009). *Estudio del refinamiento de mallas geométricas de triángulos rectángulos isósceles*.

ANEXOS

ANEXO A

CUESTIONARIO PARA LA EVALUACIÓN DE LAS MÉTRICAS DE CALIDAD

EVALUACIÓN DE CALIDAD

NOMBRE: .....

Evalúe cada punto en un rango de 0 (mínimo) y 10 (máximo)

Métricas de calidad del software.

MÉTRICA	EXPLICACIÓN	PUNTAJE
Exactitud	Exactitud de los cálculos y control	
Compleción	El grado con el que se ha logrado la implementación total de una función	
Concisión	Lo compacto	
Consistencia	El empleo de un diseño uniforme y de técnicas de documentación a lo largo del	
Estandarización de datos	el empleo de estructuras y tipos de datos estándares a lo largo del programa	
Tolerancia de Errores	El daño causado cuando el programa encuentra un error	
Eficiencia de ejecución	el rendimiento del funcionamiento del programa	
Capacidad de Expansión	El grado con el que se puede ampliar el diseño arquitectónico de datos o procedimientos	
Generalidad	La amplitud de aplicación potencial de los componentes del programa	
Instrumentación	El grado con el que el programa vigila su propio funcionamiento e identifica los errores que ocurren	
Modularidad	La independencia funcional de componentes de programa	
Operatividad	La facilidad de operación del programa	
Seguridad	la disponibilidad de mecanismos que controlan o protegen los programas y datos	
Auto documentación	El grado con el que el código fuente proporciona documentación significativa	
Simplicidad	El grado de facilidad con el que se puede entender el programa	
independencia del sistema	El grado de independencia del programa respecto a las características del lenguaje de	
Trazabilidad	La capacidad de seguir una representación del diseño o componente real del programa	
Facilidad de formación	El grado en el que ayuda a manejare! sistema a nuevos usuarios	

ANEXO B

CUESTIONARIO PARA LA EVALUACIÓN DE LOS FACTORES DE CALIDAD

EVALUACIÓN DE CALIDAD

NOMBRE:……

Evalúe cada punto en un rango de 0 (mínimo) y 10 (máximo)

Factores de calidad.

FACTORES	EXPLICACIÓN	PUNTAJE
Corrección	Hasta donde satisface el programa su especificación y logra los objetivos propuestos por el cliente	
Fiabilidad	Hasta donde se puede esperar que el programa lleve a cabo su función con la exactitud requerida	
Eficiencia	La calidad de recursos informáticos y de código necesarios para que un programa realice su función	
Integridad	Hasta donde se puede controlar el acceso del software o programa a los datos por personas no autorizadas	
Flexibilidad	El esfuerzo necesario para modificar el programa que ya está en funcionamiento	
Capacidad de pruebas	El esfuerzo necesario para probar el programa y asegurarse que se realiza correctamente su función	
Reusabilidad	Hasta donde se puede volver a emplear el programa en otras aplicaciones	
Usabilidad	El esfuerzo necesario para aprenderá operar el sistema	

ANEXO C

CODIGO FUENTE

C.1 Ventana de Renderizado con OpenGL

```
object DlgVisor3D: TDlgVisor3D
        Left = 0
        Top = 0
        Caption = 'DlgVisor3D'
        ClientHeight = 414
        ClientWidth = 624
        Color = clBlack
        Font.Charset = DEFAULT_CHARSET
        Font.Color = clWindowText
        Font.Height = -11
        Font.Name = 'Tahoma'
        Font.Style = []
        OldCreateOrder = False
        WindowState = wsMaximized
        OnClose = FormClose
        OnCreate = FormCreate
        OnKeyDown = FormKeyDown
        OnKeyUp = FormKeyUp
        PixelsPerInch = 96
        TextHeight = 13
        object ImgTextura: TImage
          Left = 619
          Top = 32
          Width = 128
          Height = 128
          AutoSize = True
          Picture.Data = {
            07544269746D617036C00000424D3000000000000036008000
            0000800000000100180000000000C000000000000000000000
            000051361D51361D51361D51361D51361D61D51361D5131E
            51361E51361E52361E52361E52361E52361E51E5236100000
            1E52361E52361E52361E52371E52371E52371E51E523711E2
            371E52371E52371E52371E52371E52371E52371E51237531F
            52371F52371F52371F52371F52371F52371F523713711F237
            1F52371F52371F52371F52371F52371F52371F52352352352
            371F52371F52371F52371F52371F52371F52371F5F523771F
```

```
37              52371F52371F52371F52371F52371F52371F5371F71F52237
38              1F52371F52371F52371F52371F52371F2371F52372371F232
39              371E52371E52371E52371E5237152371E52371E521E52371E
40              52371E52371E52371E5237E52361E52361E52361E5231E236
41              1E52361E52361E5231E51361E51361E51361E51361E1E5E51
42              361D51361D561D51361D51361D51361D51361E5361E5161E
43              351A00351A003E00351A00351A00351A00351A00503A051A
44              00351A00351A003551A00351A00351A00351A01A0AAAAAA
45              1A00}
46            Stretch = True
47            Visible = False
48          end
49          object ListBox1: TListBox
50            Left = 619
51            Top = -54
52            Width = 129
53            Height = 569
54            TabStop = False
55            ItemHeight = 13
56            TabOrder = 0
57            Visible = False
58          end
59  end
```

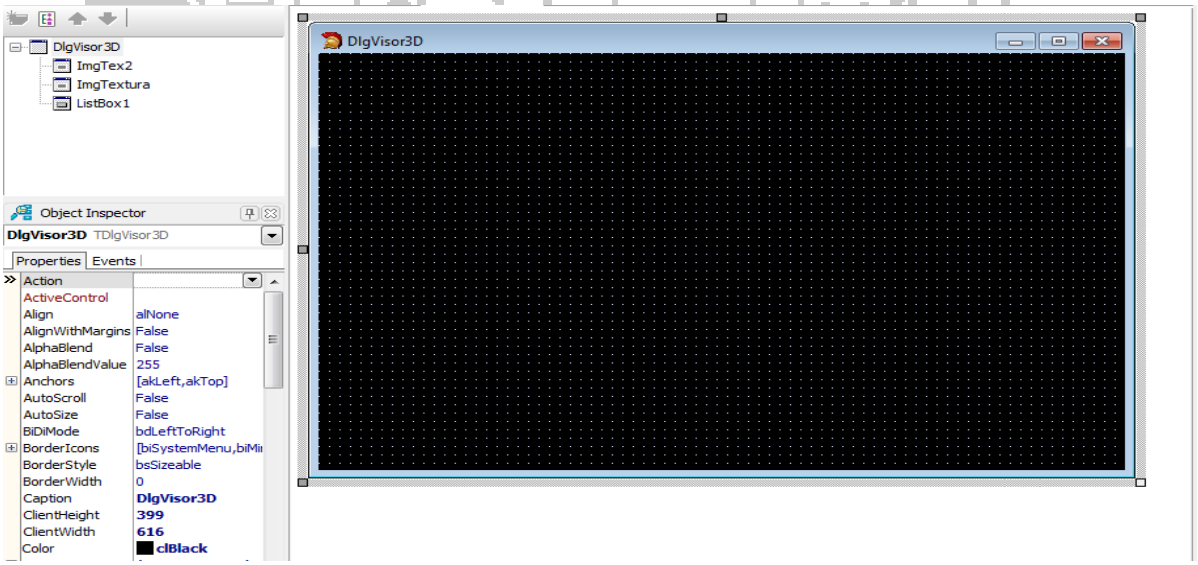


Figura C.1: Vista del Proyecto de Media3D en tiempo de desarrollo en CodeGear 2009

C.2 Ventana de Dibujo y Carga de Elementos 2D

```
1   object Principal: TPrincipal
```

```
2   Left = 0
3   Top = 0
4   Caption = 'MULTIMEDIA 3D'
5   ClientHeight = 689
6   ClientWidth = 774
7   Color = clBtnFace
8   Font.Charset = DEFAULT_CHARSET
9   Font.Color = clWindowText
10  Font.Height = -11
11  Font.Name = 'Tahoma'
12  Font.Style = [ ]
13  OldCreateOrder = False
14  OnCreate = FormCreate
15  PixelsPerInch = 96
16  TextHeight = 13
17  object itmSee2: TShape
18    Left = 8
19    Top = 154
20    Width = 769
21    Height = 527
22    Brush.Color = clBlack
23    Pen.Color = clGray
24    Pen.Style = psDot
25  end
26  object pbxArea: TPaintBox
27    Left = 8
28    Top = 160
29    Width = 760
30    Height = 513
31    Color = clBlack
32    ParentColor = False
33    OnPaint = pbxAreaPaint
34  end
35  object Ribbon1: TRibbon
36    Left = 0
37    Top = 0
38    Width = 774
39    Height = 147
40    Caption = 'Multimedia 3D'
41    Tabs = <
42      item
43        Caption = 'Principal'
44        Page = rbPage
```

```
45      end>
46    ExplicitLeft = 119
47    ExplicitTop = -8
48    ExplicitWidth = 914
49    DesignSize = (
50      774
51      147)
52    StyleName = 'Ribbon - Luna'
53    object rbPage: TRibbonPage
54      Left = 0
55      Top = 54
56      Width = 773
57      Height = 93
58      Caption = 'Principal'
59      Index = 0
60      ExplicitWidth = 716
61      object RibbonGroup1: TRibbonGroup
62        Left = 4
63        Top = 3
64        Width = 325
65        Height = 86
66        Caption = 'Archivo'
67        GroupIndex = 0
68      end
69      object RibbonGroup2: TRibbonGroup
70        Left = 331
71        Top = 3
72        Width = 262
73        Height = 86
74        Caption = 'Acabado'
75        GroupIndex = 1
76        object ImgTextura: TImage
77          Left = 25
78          Top = 15
79          Width = 112
80          Height = 42
81                Stretch = True
82                OnClick = ImgTexturaClick
83              end
84              object ChkRender: TCheckBox
85                Left = 162
86                Top = 16
87                Width = 97
```

```
88              Height = 17
89              Caption = 'Renderizar'
90              Ctl3D = False
91              Font.Charset = ANSI_CHARSET
92              Font.Color = clNavy
93              Font.Height = -11
94              Font.Name = 'Arial'
95              Font.Style = [fsBold]
96              ParentCtl3D = False
97              ParentFont = False
98              TabOrder = 0
99              OnClick = ChkRenderClick
100           end
101           object ChkTextura: TCheckBox
102             Left = 162
103             Top = 39
104             Width = 97
105             Height = 17
106             Caption = 'Con Textura'
107             Ctl3D = False
108             Enabled = False
109             Font.Charset = ANSI_CHARSET
110             Font.Color = clNavy
111             Font.Height = -11
112             Font.Name = 'Arial'
113             Font.Style = [fsBold]
114             ParentCtl3D = False
115             ParentFont = False
116             TabOrder = 1
117           end
118         end
119         object RibbonGroup3: TRibbonGroup
120           Left = 595
121           Top = 3
122           Width = 173
123           Height = 86
124           GroupIndex = 2
125         end
126       end
127     end
128     object btnNuevo: TBitBtn
129       Left = 24
130       Top = 72
```

```
131         Width = 89
132         Height = 42
133         Caption = 'NUEVO'
134         DoubleBuffered = True
135         Font.Charset = DEFAULT_CHARSET
136         Font.Color = clWindowText
137         Font.Height = -11
138         Font.Name = 'Tahoma'
139         Font.Style = [fsBold]
140         ParentDoubleBuffered = False
141         ParentFont = False
142         TabOrder = 0
143         OnClick = btnNuevoClick
144       end
145       object btnAbrir: TBitBtn
146         Left = 119
147         Top = 72
148         Width = 89
149         Height = 42
150         Caption = 'ABRIR'
151         DoubleBuffered = True
152         Font.Charset = DEFAULT_CHARSET
153         Font.Color = clWindowText
154         Font.Height = -11
155         Font.Name = 'Tahoma'
156         Font.Style = [fsBold]
157         ParentDoubleBuffered = False
158         ParentFont = False
159         TabOrder = 1
160         OnClick = btnAbrirClick
161       end
162       object BitBtn3: TBitBtn
163         Left = 214
164         Top = 72
165         Width = 89
166         Height = 42
167         Caption = 'GUARDAR'
168         DoubleBuffered = True
169         Font.Charset = DEFAULT_CHARSET
170         Font.Color = clWindowText
171         Font.Height = -11
172         Font.Name = 'Tahoma'
173         Font.Style = [fsBold]
```

```
174             ParentDoubleBuffered = False
175             ParentFont = False
176             TabOrder = 2
177           end
178           object GrdItems: TStringGrid
179             Left = 788
180             Top = 160
181             Width = 153
182             Height = 465
183             ColCount = 2
184             Ctl3D = False
185             DefaultColWidth = 55
186             DefaultRowHeight = 18
187             FixedCols = 0
188             RowCount = 2
189             Font.Charset = ANSI_CHARSET
190             Font.Color = clWindowText
191             Font.Height = -11
192             Font.Name = 'Bitstream Vera Sans Mono'
193             Font.Style = [fsBold]
194             ParentCtl3D = False
195             ParentFont = False
196             TabOrder = 3
197             ColWidths = (  76     55)
198           end
199           object BitBtn1: TBitBtn
200             Left = 624
201             Top = 73
202             Width = 121

203             Height = 40
204             Caption = 'GENERAR'
205             DoubleBuffered = True
206             Font.Charset = DEFAULT_CHARSET
207             Font.Color = clWindowText
208             Font.Height = -11
209             Font.Name = 'Tahoma'
210             Font.Style = [fsBold]
211             ParentDoubleBuffered = False
212             ParentFont = False
213             TabOrder = 5
214             OnClick = BitBtn1Click
215           end
```

```
216         object DlgAbrir: TOpenDialog
217           Filter = 'Mapas EnviroMaker|*.emmp|Todos los archivos|*.*'
218           Left = 800
219           Top = 304
220         end
221         object DlgSalva: TSaveDialog
222           Filter = 'Mapas EnviroMaker|*.emmp|Todos los archivos|*.*'
223           Left = 800
224           Top = 192
225         end
226         object DlgTextura: TOpenPictureDialog
227           Filter =
228             'All (*.jpg;*.jpeg;*.bmp;*.ico;*.emf;*.wmf)|*.jpg;*.jpeg;*.bmp;*.' +
229             'ico;*.emf;*.wmf|JPEG Image File (*.jpg)|*.jpg|Bitmaps (*.bmp)|*.'
230         +
231             'bmp|Enhanced Metafiles (*.emf)|*.emf|Metafiles (*.wmf)|*.wmf'
232           Left = 800
233           Top = 248
234         end
235 end
```

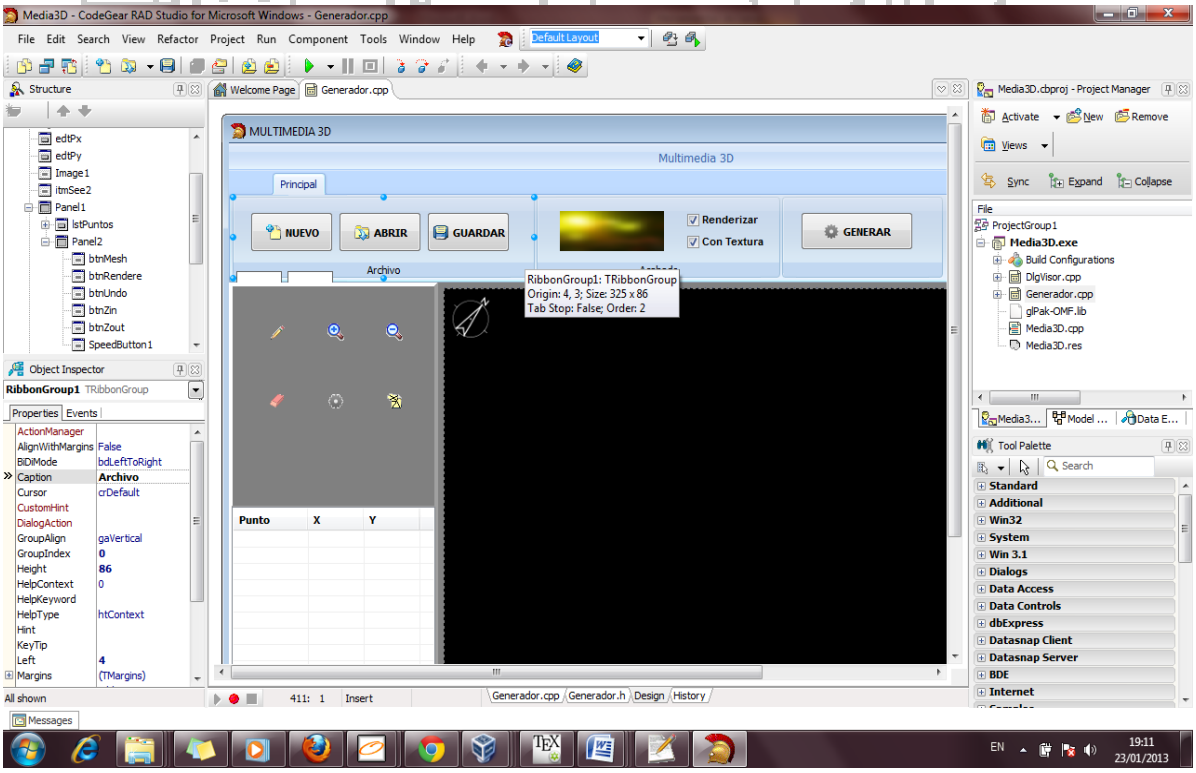


Figura C.2 : Vista del proyecto Media3D en CodeGear 2009 mostrando la interfaz gráfica.

C.3 Código para el Proyecto: CodeGear 2009

```
<Project xmlns="http://schemas.microsoft.com/developer/2003">
        <PropertyGroup>
            <ProjectGuid>
              {F3BC5AC0-142D-4A70-A5AC-DD1798E19D37}
            </ProjectGuid>
            <ProjectVersion>12.0</ProjectVersion>
            <MainSource>Media3D.cpp</MainSource>
            <Config Condition="'$(Config)'==''">Debug</Config>
        </PropertyGroup>
        <PropertyGroup
            Condition="'$(Config)'=='Base' or '$(Base)'!=''">
            <Base>true</Base>
        </PropertyGroup>
        <PropertyGroup
            Condition="'$(Config)'=='Debug' or '$(Cfg_1)'!=''">
            <Cfg_1>true</Cfg_1>
            <CfgParent>Base</CfgParent>
            <Base>true</Base>
        </PropertyGroup>
        <PropertyGroup
            Condition="'$(Config)'=='Release' or '$(Cfg_2)'!=''">
            <Cfg_2>true</Cfg_2>
            <CfgParent>Base</CfgParent>
            <Base>true</Base>
        </PropertyGroup>
        <PropertyGroup Condition="'$(Base)'!=''">

    <LinkPackag.bpi;vclx.bpi;vclactnband.bpi</LinkPackageImports>
            <Multithreaded>true</Multithreaded>
            <ProjectType>CppVCLApplication</ProjectType>

      <PackageImports>vcl.bpi;rtl.bpi;bcbie.bpi;vclx.bpi;vclactnband.bpi
    ;xmlrtl.bpi;bcbsmp.bpi;dbrtl.bpi;vcldb.bpi;vcldbx.bpi;bdertl.bpi;dsn
    ap.bpi;dsnapcon.bpi;TeeUI.bpi;TeeDB.bpi;Tee.bpi;adortl.bpi;vclib.bp
    i;ibxpress.bpi;IndyCore.bpi;IndySystem.bpi;IndyProtocols.bpi;inet.b
    pi;intrawebdb_100_120.bpi;Intraweb_100_120.bpi;VclSmp.bpi;vclie.
    bpi;websnap.bpi;webdsnap.bpi;inetdbbde.bpi;inetdbxpress.bpi;soa
    prtl.bpi;vclribbon.bpi;dbexpress.bpi;DbxCommonDriver.bpi;DataSn
    apIndy10ServerTransport.bpi;DataSnapProviderClient.bpi;DataSna
```

```
40          pServer.bpi;DbxClientDriver.bpi;DBXInterBaseDriver.bpi;DBXMySQ
41          LDriver.bpi;dbxcds.bpi;DBXSybaseASEDriver.bpi;DBXSybaseASADri
42          ver.bpi;DBXOracleDriver.bpi;DBXMSSQLDriver.bpi;DBXInformixDri
43          ver.bpi;DBXDb2Driver.bpi;fs12.bpi;frx12.bpi;frxcs12.bpi;fsDB12.bpi
44          ;frxDB12.bpi;fsBDE12.bpi;frxBDE12.bpi;fsADO12.bpi;frxADO12.bpi;
45          fsIBX12.bpi;frxIBX12.bpi;frxDBX12.bpi;frxe12.bpi;fsTee12.bpi;frxTe
46          e12.bpi</PackageImports>
47
48          <OutputExt>exe</OutputExt>
49
50            <AllPackageLibs>rtl.lib;vcl.lib;vclribbon.lib;vclx.lib;vclactnband.lib
51          </AllPackageLibs>
52
53          <DCC_CBuilderOutput>JPHNE</DCC_CBuilderOutput>
54                      <Defines>NO_STRICT</Defines>
55                      <DynamicRTL>true</DynamicRTL>
56                      <UsePackages>true</UsePackages>
57
58            <IncludePath>C:\User04\3DMedia;$(CG_BOOST_ROOT)\boost\tr1
59          \tr1;$(BDS)\include;$(BDS)\include\dinkumware;$(BDS)\include\
60          vcl;$(CG_BOOST_ROOT)</IncludePath>
61
62            <ILINK_LibraryPath>C:\User04\3DMedia;$(BDS)\lib;$(BDS)\lib\
63          obj;$(BDS)\lib\psdk</ILINK_LibraryPath>
64                      <BCC_wpar>false</BCC_wpar>
65
66           <BCC_OptimizeForSpeed>true</BCC_OptimizeForSpeed>
67                </PropertyGroup>
68                <PropertyGroup Condition="'$(Cfg_1)'!=''">
69
70            <BCC_OptimizeForSpeed>false</BCC_OptimizeForSpeed>
71
72           <BCC_DisableOptimizations>true</BCC_DisableOptimizations>
73                      <DCC_Optimize>false</DCC_Optimize>
74                      <DCC_DebugInfoInExe>true</DCC_DebugInfoInExe>
75                      <Defines>_DEBUG;$(Defines)</Defines>
76
77           <BCC_InlineFunctionExpansion>false</BCC_FunctionExpansion>
78           <IntermediateOutputDir>Debug</IntermediateOutputDir>
79
80           <ILINK_DisableIncrementalLinking>true</ILINK_mentalLinking>
81
82           <BCC_UseRegisterVariables>None</BCC_UseRegisterVariables>
```

```
83                      <DCC_Define>DEBUG</DCC_Define>
84
85          <BCC_DebugLineNumbers>true</BCC_DebugLineNumbers>
86
87          <TASM_DisplaySourceLines>true</TASM_DisplaySourceLines>
88                      <BCC_StackFrames>true</BCC_StackFrames>
89
90          <ILINK_LibraryPath>$(BDS)\lib\debug;$(ILINK_LibraryPath)
91          </ILINK_LibraryPath>
92                      <ILINK_FullDebugInfo>true</ILINK_FullDebugInfo>
93                      <TASM_Debugging>Full</TASM_Debugging>
94
95          <BCC_SourceDebuggingOn>true</BCC_SourceDebuggingOn>
96                  </PropertyGroup>
97                  <PropertyGroup Condition="'$(Cfg_2)'!=''">
98                      <DynamicRTL>false</DynamicRTL>
99                      <Defines>NDEBUG;$(Defines)</Defines>
100
101         <IntermediateOutputDir>Release</IntermediateOutputDir>
102         <ILINK_LibraryPath>$(BDS)\lib\release;$(ILINK_LibraryPath)</I
103     LINK_LibraryPath>
104                     <TASM_Debugging>None</TASM_Debugging>
105             </PropertyGroup>
106             <ItemGroup>
107                 <CppCompile Include="DlgVisor.cpp">
108                     <Form>DlgVisor3D</Form>
109                     <DependentOn>DlgVisor.h</DependentOn>
110                     <BuildOrder>3</BuildOrder>
111                 </CppCompile>
112                 <LibFiles Include="glPak-OMF.lib">
113                     <IgnorePath>true</IgnorePath>
114                     <BuildOrder>4</BuildOrder>
115                 </LibFiles>
116                 <CppCompile Include="Generador.cpp">
117                     <Form>Principal</Form>
118                     <DependentOn>Generador.h</DependentOn>
119                     <BuildOrder>2</BuildOrder>
120                 </CppCompile>
121                 <CppCompile Include="Media3D.cpp">
122                     <BuildOrder>0</BuildOrder>
123                 </CppCompile>
124                 <ResFiles Include="Media3D.res">
125                     <BuildOrder>1</BuildOrder>
```

```
126                     </ResFiles>
127                     <FormResources Include="DlgVisor.dfm"/>
128                     <FormResources Include="Generador.dfm"/>
129                     <BuildConfiguration Include="Base">
130                             <Key>Base</Key>
131                     </BuildConfiguration>
132                     <BuildConfiguration Include="Release">
133                             <Key>Cfg_2</Key>
134                             <CfgParent>Base</CfgParent>
135                     </BuildConfiguration>
136                     <BuildConfiguration Include="Debug">
137                             <Key>Cfg_1</Key>
138                             <CfgParent>Base</CfgParent>
139                     </BuildConfiguration>
140             </ItemGroup>
141             <Import Project="$(BDS)\Bin\CodeGear.Cpp.Targets"
142       Condition="Exists('$(BDS)\Bin\CodeGear.Cpp.Targets')"/>
143             <ProjectExtensions>
144             <Borland.Personality>.Personality.12</Borland.Personality>
145   <Borland.ProjectType>CppVCLApplication</Borland.ProjectType>
146                     <BorlandProject>
147                             <CPlusPlusBuilder.Personality>
148             <VersionInfoKeys Name="CompanyName"/>
149             <VersionInfoKeys Name="FileDescription"/>
150             <VersionInfoKeys name="FileVersion">1.0.0.0
151             </VersionInfoKeys>
152             <VersionInfoKeys Name="InternalName"/>
153             <VersionInfoKeys Name="LegalCopyright"/>
154             <VersionInfoKeys Name="LegalTrademarks"/>
155             <VersionInfoKeys Name="OriginalFilename"/>
156             <VersionInfoKeys Name="ProductName"/>
157             <VersionInfoKeys Name="ProductVersion">1.0.0.0
158             </VersionInfoKeys>
159             <VersionInfoKeys Name="Comments"/>
160             </VersionInfoKeys>
161             <Debugging>
162             <Debugging Name="DebugSourceDirs"/>
163             </Debugging>
164             <Parameters>
165             <Parameters Name="RunParams"/>
166             <Parameters Name="Launcher"/>
167             <Parameters Name="UseLauncher">False</Parameters>
168             <Parameters Name="DebugCWD"/>
```

```
169                 <Parameters Name="HostApplication"/>
170                 <Parameters Name="RemoteHost"/>
171                 <Parameters Name="RemotePath"/>
172                 <Parameters Name="RemoteParams"/>
173                 <Parameters Name="RemoteLauncher"/>
174                 <Parameters Name="UseRemoteLauncher">False
175                 </Parameters>
176                 <Parameters Name="RemoteCWD"/>
177                 <Parameters Name="RemoteDebug">False</Parameters>
178                 <Parameters Name="Debug Symbols Search Path"/>
179                 <Parameters Name="LoadAllSymbols">True</Parameters>
180                 <Parameters Name="LoadUnspecifiedSymbols">False
181               </Parameters>
182               </Parameters>
183
184               <ProjectProperties>
185                 <ProjectProperties Name="AutoShowDeps">False
186               </ProjectProperties>
187               <ProjectProperties Name="ManagePaths">True
188               </ProjectProperties>
189               <ProjectProperties Name="VerifyPackages">True
190             </ProjectProperties>
191             </ProjectProperties>
192             <Source>
193                   </CPlusPlusBuilder.Personality>
194                   <ModelSupport>False</ModelSupport>
195                   </BorlandProject>
196                   <ProjectFileVersion>12</ProjectFileVersion>
197             </ProjectExtensions>
198 </Project>
199
```

C.4 ESTRUCTURA DINAMICA DE DATOS BASADA EN TEMPLATES

Se ha usado una librería en lenguaje C que permita contener datos sin tener un tamaño fijo es decir ya no es necesario que definamos **arrays** de estructuras para nuestro programa de la forma:

struct Puntos2D pts2D[1000];

struct Puntos3D pts3D[1000];

Sino que al usar una estructura de datos basada en **templetes** podremos definir una colección de tamaño variable, pues el mismo tiene implementado una lista de doble enlace que permitirá realizar operaciones de mantenimiento como agregar, borrar, insertar o destruir todo el contenido. Entonces una vez agregada la librería quedaría definir del siguiente modo:

CContenedor <Puntos2D> pts2D;

CContenedor <Puntos3D> pts3D;

```
1  // Ejemplo 1 : Agregado de datos de modo dinamico
2  //              usando la E.D. CCollection
3
4  #include <stdio.h>
5  #include <conio.h>
6
7  #include "MLColection.h"
8
9  class CRect{
10
11   public:
12         int left,
13             top,
14             right,
15             bottom;
16
17    // los constructores de clase
18    public:
19          CRect(){};
20          CRect( int lft, int tp, int rgt, int btm )
21          {
22                left=lft,
23                top=tp,
24                right=rgt,
25                bottom=btm;
          };
};
```

```
26
27  int main()
28  {
29          // primero agregamos los datos una cantidad no fija
30          //
31          hLista.Add(  CRect(10,50,30,20)  );
32          hLista.Add(  CRect(20,40,31,30)  );
33          hLista.Add(  CRect(30,30,32,40)  );
34          hLista.Add(  CRect(40,20,33,50)  );
35          hLista.Add(  CRect(50,10,34,60)  );
36
37          //
38          // Ahora mostrar los elementos del contenedor.
39          // hacemos uso del 'operador []'
40          //
41
42          printf( "Total Rectangulos = %d \n",  hLista.GetSize() );
43
44          hLista.MoveFirst();
45          while(  hLista.Get()  != NULL )
46          {
47              printf( "Rectangulo [%d] = %d %d %d %d \n",
48                              hLista.Get()->left, hLista.Get()->top,
49                              hLista.Get()->right, hLista.Get()->bottom  );
50
51              hLista.MoveNext();
52          }
53
54          printf( "Con Operadores \n"  );
    }
```

Nótese que no se ha definido un tamaño para la estructura de datos sino que es la librería la que se encarga por nosotros de contener los datos que necesitemos, únicamente usamos a la función miembro:

MoveFirst ( ) : Pone el puntero al primer elemento de la colección de datos.

MoveNext ( ) : Pone el puntero al siguiente elemento de la colección, es como hacer **p++**

MoveLast ( ) : Pone el puntero al último elemento

Get( ) : Obtiene el tipo y el contenido del puntero actual, ojo que es un tipo **template** así que se definirá en tiempo de diseño el valor de retorno.

```
1   //-------------------------------------------------------------------
2   // Contenedor : MicroLogic © Software  - 2006 - HCHL
3   //          Liberado con fines educativos
4   //          Noten lo pequeño y abstracto.
5   //          no necesita 667 lineas solo 156
6   //--
7   //-- Fecha : 04-01-2013
8   //-- Hora  : 04:57 a.m.
9   //-------------------------------------------------------------------
10
11
12  #if !defined __MICRO_LOGIC_COLECTION__
13  #define __MICRO_LOGIC_COLECTION__
14
15
16  template < class _T_ValueType >
17  class CMLColection {
18
19          private:
20          typedef struct tagElement{
21                  _T_ValueType    Item;  // la VAR verdadera
22                  tagElement      *Next;   // Next Node : Linear List
23          }HELEMENT;
24
25          HELEMENT *_Nde_First, *_Nde_Last, *_Nde_Current;
26
27          long            _T_Total;
28          long            _T_Cursor;
29
30          public:
31          CMLColection( )
32                  : _T_Total(0), _T_Cursor(0)
33          {
34
```

```
35              // Asignación no estática
36              // aqui viviran las listas.
37              // pues todo será dinámico
38              //
39
40              _Nde_First = _Nde_Last = _Nde_Current = NULL;
41          };
42
43
44          ~CMLColection( )
45          {       Clear( );       }
46
47          inline bool Add( _T_ValueType _T_NwVar )
48          {       HELEMENT  * New = new HELEMENT;
49              if( ! New ) return false; // Por si se acaba la memoria
50              New->Next = NULL;   // Sgte Nodo, vacio...
51              New->Item = _T_NwVar; // el Tipo de Dato y le Dato
52              if( _Nde_First==NULL ){     _Nde_First = New; }
53              else{    _Nde_Last->Next = New; }
54              _Nde_Last = _Nde_Current = New;    _T_Total++;
55              return true;
56          };
57
58          //
59          // OJO : De estar vacio no inserta, use Add(...)
60          // Hay dos metodos de inserción en torno a Listas
61          // 1. Inserción Antes De.   ( [0] ... [2] [3] )
62          // 2. Inserción Despues De. ( [0] [1] ... [3] )
63          // en nuestro caso usaremos el Método 1. Antes de
64          // entonces no se asusten si nuestros indices I, _Ind_
65          // llevan _Ind_-1, pues necesitamos el elemento [A.D]
66          // para asi poder trabajar con [A.D]->Next, etc.
67          // lo mismo para Erase( ... )
68          //
69          bool Insert( _T_ValueType _T_NwVar, int _Ind_ )
70          {       if( _Ind_ >= _T_Total ) return false;
71              BeforeAt( (_Ind_<0)?0:_Ind_ );
72              HELEMENT  * New = new HELEMENT;
73              if( ! New ) return false; // Por si se acaba la memoria
74              New->Item = _T_NwVar;
75              if( _Nde_Current==_Nde_First && _Ind_<=0 )
76              {  _Nde_First=New;  New->Next=_Nde_Current; }
77              else{    New->Next = _Nde_Current->Next;
```

```
78                                  _Nde_Current->Next = New;}
79              _Nde_Current = New;_T_Total++;    return true;
80          };
81
82          bool Erase( int _Ind_ )
83          {       if( _Ind_ >= _T_Total ) return false;
84                  BeforeAt( (_Ind_<0)?0:_Ind_ );
85                  if( _Nde_Current==_Nde_First && _Ind_<=0 )
86                          _Nde_First = _Nde_First->Next;
87                  else if( _Nde_Current->Next == _Nde_Last )
88                  {       _Nde_Last = _Nde_Current;
89                          _Nde_Last->Next = NULL; // END
90                          _Nde_Current = _Nde_Current->Next;
91                  }else{  HELEMENT  * Bkp = _Nde_Current;
92                                  _Nde_Current = _Nde_Current->Next;
93                                  Bkp->Next = Bkp->Next->Next;      }
94                  delete _Nde_Current; _Nde_Current = _Nde_First; _T_Total--;
95                  return true;
96          };
97
98          void Clear( )
99          {       _Nde_Current = _Nde_First;
100                 // Ahora si esta todo claro : no hay ?
101                 while( _Nde_Current != NULL ){
102                         // _Nde_First = _Nde_First->Next  salvar el sgte
103                         _Nde_First = _Nde_First->Next;
104                         // puntero pero eliminar el actual
105                         delete _Nde_Current;
106                        // _Nde_Current = _Nde_First, ese nodo delete
107                         _Nde_Current = _Nde_First;
108                 }       _Nde_First = _Nde_Current = _Nde_Last = NULL;
109                 _T_Total = _T_Cursor = 0;
110         };
111
112         inline long GetSize( )
113         { return _T_Total;    }
114
115         inline void MoveFirst( )
116         {       _Nde_Current = _Nde_First; }
117
118         inline void MoveLast( )
119         {       _Nde_Current = _Nde_Last; }
120
```

```
121             inline void MoveNext( )
122             {       _Nde_Current = _Nde_Current->Next; }
123
124             inline _T_ValueType *Get( )
125             {       return &(_Nde_Current->Item); }
126
127             // Solo es efectivo al hacer Val=Templt[0]
128             // mas no funciona al hacer Tmplt[0]=Val
129             //
130             _T_ValueType operator[]( int _Ind_ ) //const
131             {       return *( LocateIn(_Ind_) );            };
132
133
134             // solo es _Ind_ible Tmplt = Val => direct.
135             operator= ( _T_ValueType NwVar ) //const
136             {       _Nde_Current->Item = NwVar;       return 0; };
137
138             _T_ValueType * LocateIn( int _Ind_ )
139             {       if(_Ind_>=_T_Total) return NULL;
140                     _T_Cursor  = 0;
141                     _Nde_Current = _Nde_First;
142                     return  &(GetAt(_Ind_)->Item);      };
143
144             private:
145             HELEMENT  * GetAt( int _Ind_ )
146             {       if( _T_Cursor++==_Ind_ ) return _Nde_Current;
147                     _Nde_Current = _Nde_Current->Next;
148                     return GetAt(_Ind_); };
149             // Antes de 0-1 el _T_Cursor es 0, en los
150             // otros casos _T_Cursor=_Ind_-1.
151             void BeforeAt( int _Ind_ )
152             {       if(_Ind_>=_T_Total) return;
153                     _T_Cursor  = 0;
154                     _Nde_Current = _Nde_First;
155                     GetAt( ((_Ind_-1)<0)?0:_Ind_-1 );           };
156             //void Serialize( );
157 };
158 #endif //... CMLColection ... That's All Folks !
```

C.5 Algoritmo de Rotación de Imágenes

La edición digital de imágenes se ocupa de la edición apoyada en computadores de Imágenes digitales, comúnmente un gráfico rasterizado, en la mayoría de los casos fotos o documentos escaneados. Estas imágenes son modificadas para optimizarlas, manipularlas, retocarlas, etc. con el fin de alcanzar la meta deseada.

Para lograr estos resultados, se requiere adaptar las fórmulas matemáticas para transformaciones geométricas en algoritmos computacionales, respetando los conceptos matemáticos. La matriz de rotación geométrica tiene la forma:

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

Para el caso de los puntos ***X,Y*** y poder obtener nuevos puntos ***X',Y'*** y evitarnos plantear una matriz real, solo hacemos la operación tratándola como matriz aunque no lo sea. El algoritmo planteado para la operación de estas matrices puede verse de la siguiente forma:

```
1  //
2  //  F-Media3D : Algoritmo de Rotacion adaptado para C/C++
3  //
4
5  void FncRotacion(float x, float y, float *rx, float *ry, float angulo)
6  {
7      #define PI 3.141592654
8
9      angulo = ( PI * angulo ) / 180.0;
```

```
10
11      *rx = x*cos(angulo) - y*sin(angulo);
12      *ry = x*sin(angulo) + y*cos(angulo);
13  }
14
15  void TFrmLona::FncAgregaPunto( int pX, int pY )
16  {
        ptPunto[ptTotal].x = pX;
        ptPunto[ptTotal].y = pY;
        ptTotal++;
    }
```

C.6 Refinamiento de Mallas con Delaunay

En esta sección se revisan otros algoritmos de refinamiento de mallas, específicamente los que trabajan en mallas Delaunay. El objetivo principal de los algoritmos de refinamiento de mallas Delaunay es insertar nuevos puntos en lugares estratégicos con el objetivo de eliminar elementos de baja calidad, de tal manera que ningún vértice caiga dentro del circuncirculo de los triángulos, manteniendo las propiedades Delaunay y por lo tanto mejorando la calidad de la malla resultante.

A continuación se revisan los algoritmos más destacados de refinamiento de mallas Delaunay. La estrategia del algoritmo es hacer mejoras locales con el objetivo de eliminar triángulos flacos (triángulos que tienen algún _ángulo menor a una cota Alpha). Cada mejora implica agregar un nuevo vértice a la triangulación y en consecuencia volver a triangular. Para un segmento S el círculo con ***S*** como diámetro es referido como su círculo diametral, y se dice que un vértice invade un segmento S si cae en el círculo diametral de ***S***. En esencia el algoritmo divide triángulos flacos añadiendo un vértice en su circuncentro que puede caer no necesariamente en el triangulo vecino

adyacente, a menos que el circuncirculo del triangulo invada algún segmento, en tal caso se divide el segmento (o los segmentos invadidos) en lugar del triangulo.

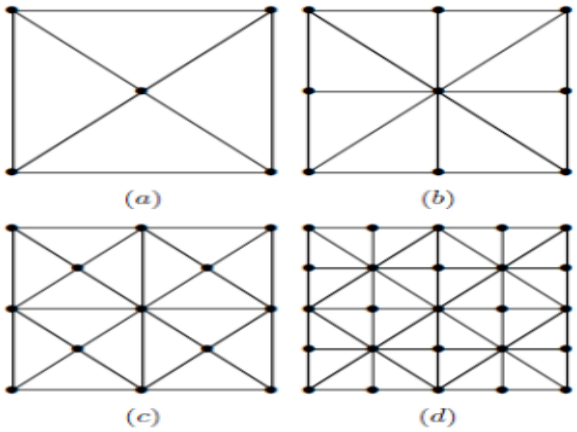


Figura 6.1 – Refinamiento de mallas por triangulación de Delaunay.

Algoritmo de refinamiento de mallas de delaunay

***Entrada*:** Un conjunto *P* de puntos en el plano.

Salida: La Triangulación de Delaunay de *P*.

1. Sean p_{-1}, p_{-2} y p_{-3} tres puntos tales que *P* está contenido en el triángulo que forman.

2. Inicializamos *T* como una triangulación de un único triángulo $p_{-1}\ p_{-2}\ p_{-3}$

3. Realizar una permutación cualquiera $p_1, p_2, \ldots, p_n$ de *P*

4. for $r:=1$ **to** n

5. hacer (* Insertar p_r en *T* *)

6. Encontrar un triángulo $p_i\ p_j\ p_k$ de *T* que contenga a p_r

7. si p_r cae en el interior del triángulo $p_i\ p_j\ p_k$

8. entonces

Añadir aristas desde p_r a los tres vértices de $p_i\ p_j\ p_k$, dividiendo este triángulo en tres

9. LEGALIZA_LADO (p_r, p_i p_j,T)

10. LEGALIZA_LADO (p_r, p_j p_k,T)

11. LEGALIZA_LADO (p_r, p_k p_i,T)

12. en caso contrario

(* p_r cae encima de uno de los lados del triángulo p_i p_j p_k, por ejemplo el lado p_i p_j *)

13. Añadir aristas desde p_r a p_k y al tercer vértice p_l del otro triángulo que comparte la arista p_i p_j, de esta forma dividimos los dos triángulos que comparten la arista p_i p_j en cuatro triángulos.

14. LEGALIZA_LADO (p_r, p_i p_l,T)

15. LEGALIZA_LADO (p_r, p_l p_j,T)

16. LEGALIZA_LADO (p_r, p_j p_k,T)

17. LEGALIZA_LADO (p_r, p_k p_i,T)

18. Descartar p_{-1}, p_{-2} *y* p_{-3} y todas las aristas que parten de ellos de T

19. devuelve *T*

```
1   // Archivo : Generador.cpp
2
3   #include <vcl.h>
4
5   #include <math.h>
6   #include <fstream.h>
7
8   #pragma hdrstop
9
10  #include "Generador.h"
11  #include "DlgVisor.h"
12
13  //---------------------------------------------------------------------------
14  #pragma package(smart_init)
15  #pragma resource "*.dfm"
16  TPrincipal *Principal;
17  //---------------------------------------------------------------------------
18  __fastcall TPrincipal::TPrincipal(TComponent* Owner)
19          : TForm(Owner)
20  {
```

```
21
22  }
23  //---------------------------------------------------------------------------
24  void __fastcall TPrincipal::FormCreate(TObject *Sender)
25  {
26        GrdItems->ColWidths[0] = 80;
27        GrdItems->ColWidths[1] = 44;
28
29        arrCuantos = 0;
30
31        ptTotal = 0;
32  }
33
34  //---------------------------------------------------------------------------
35
36  void __fastcall TPrincipal::btnNuevoClick(TObject *Sender)
37  {
38     if( ptTotal<=1 )
39     {
40        MessageBox( "No hay suficientes Puntos", "Error", MB_ICONSTOP );
41        return;
42     }
43
44     arrCont[ arrCuantos ] = ptTotal;
45     for( int i=0; i<ptTotal; i++ )
46        arrPunto[arrCuantos][ i ] = ptPunto[i];
47
48     arrCuantos++;
49
50     // reiniciar
51     ptTotal = 0;
52
53     // Redibujar todo
54           pbxArea->Invalidate();
55  }
56  //---------------------------------------------------------------------------
57
58  void __fastcall TPrincipal::btnAbrirClick(TObject *Sender)
59  {
60           int i, j;
61
62           if( DlgAbrir->Execute() == mrOk )
63           {
```

```
64                  //fstream arch( DlgAbrir->FileName.c_str() , ios::in );
65                  fstream arch( DlgAbrir->FileName.w_str() , ios::in );
66
67      if( arch.fail() )
68      {
69        ShowMessage( "Imposible Abrir" );
70                          return;
71      }
72
73      char tmp[80];
74
75      // Get Header
76      arch.getline( tmp, 80 );
77      arch.getline( tmp, 80 );
78
79      arch >>arrCuantos;
80
81      //Caption = arrCuantos;
82      for( i=0; i<arrCuantos; i++ )
83        arch >>arrCont[i];
84
85      for( i=0; i<arrCuantos; i++ )
86        for( j=0; j<arrCont[i]; j++ )
87
88       arch >>arrPunto[i][j].x >>arrPunto[i][j].y;
89      arch.close();
90
91       Redibujar();
92       pbxArea->Invalidate();
93    }
94
95  }
96
97  //---------------------------------------------------------------------------
98
99  void TPrincipal::Redibujar( )
100 {
101           ptTotal = 0;
102
103           // Redibujar todo
104           pbxArea->Invalidate();
105 }
106 //---------------------------------------------------------------------------
```

```
107 void TPrincipal::FncAgregaPunto( int pX, int pY )
108 {
109         ptPunto[ptTotal].x = pX; //StrToInt(edtPx->Text);
110         ptPunto[ptTotal].y = pY; //StrToInt(edtPy->Text);
111         ptTotal++;
112 }
113 //---------------------------------------------------------------------------
114 void __fastcall TPrincipal::pbxAreaPaint(TObject *Sender)
115 {
116         FncRedibujar();
117 }
118 //---------------------------------------------------------------------------
119 //---------------------------------------------------------------------------
120 #define BX 60
121 #define BY 400
122
123 // valores de escala de dibujo
124 int Scala = 16;
125
126
127 void TPrincipal::FncRedibujar()
128 {
129         int i, j;
130
131         // el punto central
132         int BaseX = BX;
133         int BaseY = BY;
134
135         // primero la grilla de puntos
136         pbxArea->Canvas->Pen->Width = 1;
137         pbxArea->Canvas->Pen->Color = TColor( RGB(40,52,40) );
138         for( i=0; i<=50; i++ )
139         {
140                 // +
141                 pbxArea->Canvas->MoveTo( BaseX + i*Scala, 0 );
142                 pbxArea->Canvas->LineTo( BaseX + i*Scala,  Width );
143
144                 // -
145                 pbxArea->Canvas->MoveTo( BaseX - i*Scala, 0 );
146                 pbxArea->Canvas->LineTo( BaseX - i*Scala,  Width );
147         }
148
149         for( i=0; i<=40; i++ )
```

```
150         {
151             // +
152             pbxArea->Canvas->MoveTo( 0, BaseY - i*Scala );
153             pbxArea->Canvas->LineTo(  Width, BaseY - i*Scala );
154
155             // -
156             pbxArea->Canvas->MoveTo( 0, BaseY + i*Scala );
157             pbxArea->Canvas->LineTo(  Width, BaseY + i*Scala );
158         }
159
160         // despues los ejes, color de los ejes
161         pbxArea->Canvas->Pen->Color = clBlack;
162         pbxArea->Canvas->Pen->Width = 2;
163         pbxArea->Canvas->MoveTo( 0, BaseY );
164         pbxArea->Canvas->LineTo( pbxArea->Width, BaseY );
165
166         pbxArea->Canvas->MoveTo( BaseX, 0 );
167         pbxArea->Canvas->LineTo( BaseX, pbxArea->Height );
168
169
170         // los puntos de color Azul
171         pbxArea->Canvas->Pen->Color = clYellow;
172         pbxArea->Canvas->Pen->Width = 2;
173
174         if( ptTotal > 0 )
175         {
176             // Redibujar
177             Area->Canvas->MoveTo(  BaseX + ( Scala * ptPunto[0].x ),
178                                    BaseY - ( Scala * ptPunto[0].y )  );
179
180             // ahora movemos el lapiz de dibujo
181             for( i=0; i<ptTotal; i++ )
182             {
183                 // dibujamos punto a punto
184                 pbxArea->Canvas->LineTo(
185                                 BaseX + ( Scala * ptPunto[i].x ),
186                                 BaseY - ( Scala * ptPunto[i].y )
187                                 );
188             }
189         }
190
191         pbxArea->Canvas->Pen->Color = clYellow;
192         pbxArea->Canvas->Pen->Width = 1;
```

```
193
194         // ahora la coleccion de poligonos listos
195         for( i=0; i<arrCuantos; i++ )
196         {
197             pbxArea->Canvas->MoveTo(
198                 BaseX + ( Scala * arrPunto[i][0].x ),
199                 BaseY - ( Scala * arrPunto[i][0].y )  );
200
201             for( int j=0; j<arrCont[i]; j++ )
202             {
203                 // dibujamos punto a punto
204                 pbxArea->Canvas->LineTo(
205                     BaseX + ( Scala * arrPunto[i][j].x ),
206                     BaseY - ( Scala * arrPunto[i][j].y )
207                 );
208             }
209         }
210
211         // extra
212         FncMostrarXY();
213 }
214
215
216
217 void TPrincipal::FncMostrarXY()
218 {
219         String tmp;
220
221         GrdItems->Cells[0][0] = "(x,y)";
222         GrdItems->Cells[1][0] = "dx";
223
224         if( ptTotal == 0 )
225         {
226             GrdItems->RowCount = 2;
227             GrdItems->Cells[0][1] = "--";
228             GrdItems->Cells[1][1] = "-";
229             return;
230         }
231
232         lstPuntos->Items->Clear();
233         for( int i=0; i<ptTotal; i++ )
234         {
235             GrdItems->Cells[0][1+i] = "(" +String(ptPunto[i].x)+
```

```
236                                 "," +String(ptPunto[i].y)+ ")";
237
238                     GrdItems->Cells[0][1+i] = tmp.sprintf( "(%.1f,%.1f)" ,
239                                                 ptPunto[i].x, ptPunto[i].y );
240
241                     TListItem *itm = lstPuntos->Items->Add();
242                     itm->Caption = tmp.sprintf( "(%.1f,%.1f)",
243                                                 ptPunto[i].x, ptPunto[i].y  );
244
245                     itm->Caption = tmp.sprintf( "(%.1f,%.1f)" ,
256                                                 ptPunto[i].x, ptPunto[i].y  );
247
248                     if( i>0 )
249                     {
250                        float  dx = sqrt( pow(ptPunto[i].x-ptPunto[i-1].x,2)
251                                        +  pow(ptPunto[i].y-ptPunto[i-1].y,2) );
252
253                        GrdItems->Cells[1][1+i] = tmp;
254                     }
255
256
257                     TListViewItem *itm = lstPuntos->Items->Add();
258                     TListItem *itm = new TListItem();
259                     lstPuntos->Items->AddItem( itm, 0 );
260                     itm->Item[0]->Caption = "(x,y)";
261                     itm->Item[1]->Caption = "(y,z)";
262                     itm->Array->Insert();
263               }
264  }
265
266  //---------------------------------------------------------------------------
267
268  void __fastcall TPrincipal::ChkRenderClick(TObject *Sender)
269  {
270       ChkTextura->Enabled = ChkRender->Checked;
271       ChkTextura->Checked = false;
272  }
273  //---------------------------------------------------------------------------
274
275  void __fastcall TPrincipal::ImgTexturaClick(TObject *Sender)
276  {
277     if( DlgTextura->Execute() == mrOk )
278     {
```

```
279         ImgTextura->Picture->LoadFromFile( DlgTextura->FileName );
280     }
281 }
282 //---------------------------------------------------------------------------
283
284 void __fastcall TPrincipal::BtnRenderClick(TObject *Sender)
285 {
286         btnNuevoClick(Sender);
287         Sleep( arrCuantos*250 );
288
289         // Cargar el dialogo resultante
290         TDlgVisor3D *pVisor3D = new TDlgVisor3D(this);
291
292         int i, j, k;
293         for( i=0, k=0;i<arrCuantos;i++ )
294             for( j=0; j<arrCont[i]; j++, k++ )
295                 arrVertices[k] = arrPunto[i][j];
296
297         pVisor3D->glEnviaTextura( ImgTextura->Picture->Bitmap );
298         pVisor3D->glEnviaMapa(  arrCuantos, arrCont, arrVertices,
299                                 ChkRender->Checked,
300                                 ChkTextura->Checked );
301
302
303         pVisor3D->Show();
304
305 }
306
307 //---------------------------------------------------------------------------
308
309 void __fastcall TPrincipal::pbxAreaMouseMove( TObject *Sender,
310                                 TShiftState Shift, int X,   int Y)
311 {
312         // el punto central
313         int BaseX = BX;
314         int BaseY = BY;
315
316         int pDX = ((X-BaseX) / Scala) * Scala;
317         int pDY = -((Y-BaseY) / Scala) * Scala;
318
319         // primero la grilla de puntos
320         pbxArea->Canvas->Pen->Width = 1;
321         pbxArea->Canvas->Pen->Color = clOlive;
```

```
322
323             pbxArea->Canvas->Rectangle( BaseX+pDX-2, BaseY-pDY-2,
324                                         BaseX+pDX+2, BaseY-pDY+2 );
325 }
326 //---------------------------------------------------------------------------
327
328 void __fastcall TPrincipal::pbxAreaMouseUp( TObject  *Sender,
329               TMouseButton Button, TShiftState Shift,  int X, int Y )
330 {
331             // el punto central
332             int BaseX = BX;
333             int BaseY = BY;
334
335             edtPx->Text = (X - BaseX) / Scala;
336             edtPy->Text = -(Y - BaseY) / Scala;
337
338             FncAgregar();
339 }
340 //---------------------------------------------------------------------------
341
342 void TPrincipal::FncAgregar()
343 {
344             if( edtPx->Text.IsEmpty() || edtPy->Text.IsEmpty() )
345             {
346                         Application->MessageBox(  _T("Se necesita un numero"),
347                                                   _T("Ingreso"), MB_ICONSTOP );
348                         return;
349             }
350
351             FncAgregaPunto( StrToInt(edtPx->Text), StrToInt(edtPy->Text) );
352             pbxArea->Invalidate();
353 }
354
355 void __fastcall TPrincipal::BitBtn3Click(TObject *Sender)
356 {
357             if( DlgSalva->Execute() == mrOk )
358             {
359                         btnNuevoClick(Sender);
360
361                         fstream arch(  DlgSalva->FileName + ".malla") , ios::out );
362
363                         if( arch.fail() )
364                         {
```

```
365                     ShowMessage( "Imposible guardar" );
366                     return;
367                 }
368
369                 arch <<"Media3D - 2013" <<endl
370                      <<"---------------------" <<endl;
371
372                 arch <<arrCuantos <<endl;
373
374                 for( i=0; i<arrCuantos; i++ )
375                     arch<<"\t" <<arrCont[i] <<endl;
376
377                 for( i=0; i<arrCuantos; i++ )
378                     for( j=0; j<arrCont[i]; j++ )
379                         arch << arrPunto[i][j].x <<".0 \t"
380                              << arrPunto[i][j].y <<".0"
381                              << endl;
382     arch.close();
383   }
384
385 }
```

```
1   // Archivo : DlgVisor.cpp
2   //
3
4   #include <vcl.h>
5   #include <windows.h>
6   #pragma hdrstop
7
8   #include "DlgVisor.h"
9   #include "Generador.h"
10
11  #include <gl/gl.h>
12  #include <gl/glu.h>
13  #include "glPak.h"
14
15  //---------------------------------------------------------------------
16  #pragma package(smart_init)
17  #pragma resource "*.dfm"
18  TDlgVisor3D *DlgVisor3D;
19
```

```
20  void OnV3dInit(void);
21  void OnV3dDisplay( void );
22  void OnV3dDrawAxis( void );
23  void OnV3dGrid(void);
24
25  void OnV3dDrawSkeleton(void);
26  void OnV3dDrawRendered(void);
27
28  void OnV3dMotion(int,int,int);
29  void OnV3dMouseDown(int,int,int);
30  void OnV3dMouseUp(int,int,int);
31
32  int   oldXX, oldYY;
33  int   antSpinXX;
34  int   antSpinYY;
35
36  float Alzada = 7.7;
37  float Alejar = 10;
38  float angAlpha = -60;    // x
39  float angTheta = 0;      // y
40  float angGamma = 30;   // z
41
42
43
44  int      ptsTotal;
45  const int   *arrCount;
46  ptPUNTO    *ptsVector;
47
48  GLuint  texture;
49  bool    drawRendered = false;
50  bool    drawTextured = false;
51
52  Graphics::TBitmap *bitmap;
53  //---------------------------------------------------------------------------
54  __fastcall TDlgVisor3D::TDlgVisor3D(TComponent* Owner)
55              : TForm(Owner)
56  {
57    antSpinXX = angGamma;
58    antSpinYY = angAlpha;
59  }
60  //---------------------------------------------------------------------------
61  void __fastcall TDlgVisor3D::FormCreate(TObject *Sender)
62  {
```

```
63              TRect mRect = GetClientRect();
64
65              glhWnd = mglCreateViewAsChild( this->Handle, 0, 0, 1024, 768 );
66
67              //glhWnd = mglCreateViewAsChild( this->Handle, 0, 0,
68              //                mRect.Width(), mRect.Height() );
69
70              // Send the Handler
71              mglCreateHandler( glhWnd );
72              mglDisplayFunc( OnV3dDisplay );
73
74      // para manipular el Mouse
75      mglMouseMoveFunc( OnV3dMotion );
76      mglMouseDownFunc( OnV3dMouseDown );
77      mglMouseUpFunc( OnV3dMouseUp );
78  }
79  //---------------------------------------------------------------------------
80
81  void __fastcall TDlgVisor3D::FormClose(TObject *Sender, TCloseAction
82  &Action)
83  {
84              mglDestroyHandler( glhWnd );
85  }
86  //---------------------------------------------------------------------------
87  void  TDlgVisor3D::glEnviaTextura( Graphics::TBitmap *pBmp )
88  {
89              bitmap = pBmp;
90  }
91  //---------------------------------------------------------------------------
92  void  TDlgVisor3D::glEnviaMapa( int Total, const int *pArrCount, void
93  *pVector,
94                                                                  bool render, bool
95  textura )
96  {
97      ptsTotal  = Total;
98      arrCount  = pArrCount;
99      ptsVector = (ptPUNTO*) pVector;
100
101     drawRendered = render;
102     drawTextured = textura;
103
104     // buscar los puntos mas lejanos en X y Y para obtener la proyeccion
105     // adecuada a los puntos
```

```
106             float maxX = 0;
107             float maxY = 0;
108
109     ListBox1->Items->Add( ptsTotal );
110     for( int k=0, i=0; i<ptsTotal; i++ )
111     {
112       ListBox1->Items->Add( "VecT = " + String(arrCount[i]) );
113       for( int j=0; j<arrCount[i]; j++, k++ )
114       {
115         ListBox1->Items->Add( "(" +String(ptsVector[k].x)+","+
116 String(ptsVector[k].y) + ")" );
117
118         // en X, Y
119         if( ptsVector[k].x > maxX ) maxX = ptsVector[k].x;
120         if( ptsVector[k].y > maxY ) maxY = ptsVector[k].y;
121       }
122
123       // de ambos
124       Alejar = maxY;
125       if( maxX > maxY ) Alejar = maxX;
126     }
127
128     OnV3dInit();
129
130
131 void __fastcall TDlgVisor3D::FormKeyUp(TObject *Sender, WORD &Key,
132 TShiftState Shift)
133
134 {
135         FormProcessKey( Key );
136 }
137 //---------------------------------------------------------------------------
138 void __fastcall TDlgVisor3D::FormKeyDown(TObject *Sender, WORD &Key,
139 TShiftState Shift)
140
141 {
142           FormProcessKey( Key );
143 }
144 //---------------------------------------------------------------------------
145 void __fastcall TDlgVisor3D::FormProcessKey( WORD &Key )
146 {
147
148           if( Key == 27  )
```

```
149                 Close();
150
151             if( Key == VK_DOWN  )  angAlpha++;
152             if( Key == VK_UP    )  angAlpha--;
153
154             if( Key == VK_RIGHT ) angTheta++;
155             if( Key == VK_LEFT  ) angTheta--;
156
157             if( Key == VK_PRIOR ) angGamma--;
158             if( Key == VK_NEXT  ) angGamma++;
159
160             if( Key == VK_RETURN )
161             {
162                 TRect rct = GetClientRect();
163                 this->WindowState = wsMaximized;
164                 SetWindowPos( glhWnd, HWND_NOTOPMOST, 0, 0,
165                               rct.Width(), rct.Height(), SWP_NOMOVE );
166             }
167
168             mglPostRedisplay();
169 }
170 //---------------------------------------------------------------------------
171
172 void OnV3dInit()
173 {
174   GLubyte bits[128][128][4];
175   for(int i = 0; i <128; i++)
176   {
177       for(int j = 0; j <128; j++)
178     {
179       bits[i][j][0]= (GLbyte)GetRValue(bitmap->Canvas->Pixels[i][j]);
180       bits[i][j][1]= (GLbyte)GetGValue(bitmap->Canvas->Pixels[i][j]);
181       bits[i][j][2]= (GLbyte)GetBValue(bitmap->Canvas->Pixels[i][j]);
182       bits[i][j][3]= (GLbyte)255;
183     }
184   }
185   glPixelStorei(GL_UNPACK_ALIGNMENT, 4);
186   glGenTextures(1, &texture);
187   glBindTexture(GL_TEXTURE_2D, texture);
188   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
189   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
190   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_FILTR, GL_NEAREST);
191   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_FILTR, GL_NEAREST);
```

```
192     glTexImage2D(GL_TEXTURE_2D, 0,  0, GL_RGBA, GL_UNSIGNED, bits);
193
194     // habilitar la textura
195     glEnable(GL_TEXTURE_2D);
196     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
197
198     // para profundidad sin sobreposicion
199     glEnable(GL_DEPTH_TEST);
200     glEnable(GL_CULL_FACE);
201
202     glEnable(GL_DEPTH_TEST);
203     glDepthFunc(GL_LESS);
204
205     glMatrixMode(GL_PROJECTION);
206     glLoadIdentity();
207     glMatrixMode(GL_MODELVIEW);
208
209     // proyeccion ortogonal en los tres ejes
210     // de acuerdo a los valores calculados
211     glOrtho( -4.0, Alejar+4,   -4.0, Alejar+4,    -77.0, 77.0  );
212  }
213
214  void OnV3dDrawSkeleton()
215  {
216     GLint I, J, K;
217
218     OnV3dDrawAxis();
219
220     // Proceso de completado
221     glColor3f( 1.0f, 0.9f, 0.2f );
222     glDisable(GL_TEXTURE_2D);
223
224     // cada conjunto de vectores sera leido desde el segundo vertice
225     // se leera N-1 vertices, de forma que no altere el contenido
226     for( K=0, I=0; I<ptsTotal; I++ )
227     {
228        K++; // extra
229        glBegin( GL_LINE_STRIP );
230        for( J=1; J<arrCount[I]; J++, K++ )
231        {
232           // primer triangulo
233           glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, 0 );
234           glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, Alzada );
```

```
235         glVertex3f( ptsVector[K  ].x, ptsVector[K  ].y, Alzada );
236         glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, 0 );
237
238         // segundo triangulo
239         glVertex3f( ptsVector[K].x, ptsVector[K].y, 0 );
240         glVertex3f( ptsVector[K].x, ptsVector[K].y, Alzada );
241       }
242       glEnd();
243     }
244
245 }
256
247 void OnV3dDrawRendered(void)
248 {
249    GLint I, J, K;
250
251    OnV3dGrid();
252
253    // crear la textura si se permite
254    if( drawTextured )
255    {
256       glEnable(GL_TEXTURE_2D);
257       glBindTexture(GL_TEXTURE_2D, texture);
258    }
259    else
260    {
261       glDisable(GL_TEXTURE_2D);
262    }
263
264    // Proceso de Renderizado Triangular
265    //
266    // cada conjunto de vectores sera leido desde el segundo vertice
267    // se leera N-1 vertices, de forma que no altere el contenido
268    //
269    for( K=0, I=0; I<ptsTotal; I++ )
270    {
271       K++; // extra
272
273       // establecer perspectiva contraria para solapar
274       glDisable(GL_CULL_FACE);
275       glBegin( GL_TRIANGLES );
276       for( J=1; J<arrCount[I]; J++, K++ )
277       {
```

```
278             // primer triangulo
279             glColor3f( 0.15f, 1.0f, 0.715f );
280
281             glTexCoord2f(0.0,0.0);
282             glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, 0 );
283             glTexCoord2f(1.0,0.0);
284             glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, Alzada );
285             glTexCoord2f(1.0,1.0);
286             glVertex3f( ptsVector[K  ].x, ptsVector[K  ].y, Alzada );
287
288             // segundo triangulo
289             glColor3f( 0.0f, 1.0f, 0.7f );
290             glTexCoord2f(0.0,0.0);
291             glVertex3f( ptsVector[K-1].x, ptsVector[K-1].y, 0 );
292             glTexCoord2f(1.0,0.0);
293             glVertex3f( ptsVector[K  ].x, ptsVector[K  ].y, 0 );
294             glTexCoord2f(1.0,1.0);
295             glVertex3f( ptsVector[K  ].x, ptsVector[K  ].y, Alzada );
296         }
297         glEnd();
298
299
300
301         // reestablecer vista simple
302         glEnable(GL_CULL_FACE);
303     }
304 }
305
306 void OnV3dDrawAxis()
307 {
308             glLineWidth(2);
309             glBegin( GL_LINES );
310                     glColor3f( 1.0f, 0.0f, 0.0f ); // X Red
311                     glVertex3f( 0.0f, 0.0f, 0.0f );
312                     glVertex3f( 2.0f, 0.0f, 0.0f );
313
314                     glColor3f( 0.0f, 1.0f, 0.0f ); // Y Green
315                     glVertex3f( 0.0f, 0.0f, 0.0f );
316                     glVertex3f( 0.0f, 2.0f, 0.0f );
317
318                     glColor3f( 0.0f, 0.0f, 1.0f ); // Z Blue
319                     glVertex3f( 0.0f, 0.0f, 0.0f );
320                     glVertex3f( 0.0f, 0.0f, 2.0f );
```

```
321             glEnd();
322             glLineWidth(1);
323 }
324
325 void OnV3dGrid()
326 {
327    float flDonde = Alejar+2; //30.0f;
328
329    glColor3f( 0.15f, 0.15f, 0.15f );
330    glBegin( GL_LINES );
331    for( float i=0.0f; i<=flDonde; i+=0.5f )
332    {
333       glVertex3f( 0, i, -0.2 );
334       glVertex3f( flDonde, i, -0.2 );
335
336       glVertex3f( i, 0, -0.2 );
337       glVertex3f( i, flDonde, -0.2 );
338    }
339    glEnd();
340 }
341
342
343
344 void OnV3dMotion( int iShift, int xPos, int yPos )
345 {
346          if( iShift == MK_LBUTTON )
347          {
348                // tomar anterior mas actual
349                angGamma = antSpinXX + xPos - oldXX;
350                angAlpha = antSpinYY + yPos - oldYY;
351
352                mglPostRedisplay();
353          }
354 }
355
356 void OnV3dMouseDown( int iShift, int xPos, int yPos )
357 {
358             //SetCapture( this->Handle );
359
360             oldXX = xPos;
361             oldYY = yPos;
362 }
363
```

```
364  void OnV3dMouseUp( int iShift, int xPos, int yPos )
365  {
366          // este es el truco
367          antSpinXX = angGamma;  // angX
368          antSpinYY = angAlpha;  // angY
369
370          ReleaseCapture();
371  }
372
373  void __fastcall TDlgVisor3D::spnUpClick(TObject *Sender)
374  {
375          // =
376          angAlpha = angAlpha - 5;
377          mglPostRedisplay();
378  }
379
380  void __fastcall TDlgVisor3D::spnDownClick(TObject *Sender)
381  {
382          angAlpha = angAlpha + 5;
383          mglPostRedisplay();
385  }
```

```
1   // Archivo : 3DMedia.cpp
2   //
3
4   //---------------------------------------------------------------------------
5   #include <vcl.h>
6   #pragma hdrstop
7   #include <tchar.h>
8   //---------------------------------------------------------------------------
9   USEFORM("Generador.cpp", Principal);
10  USEFORM("DlgVisor.cpp", DlgVisor3D);
11  //---------------------------------------------------------------------------
12  WINAPI _tWinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
13  {
14          try
15          {
16                  Application->Initialize();
17                  Application->MainFormOnTaskBar = true;
18                  Application->CreateForm(__classid(TPrincipal), &Principal);
19                  Application->Run();
20          }
21          catch (Exception &exception)
22          {
23                  Application->ShowException(&exception);
24          }
25          catch (...)
26          {
27                  try
28                  {
29                          throw Exception("");
30                  }
31                  catch (Exception &exception)
32                  {
33                          Application->ShowException(&exception);
34                  }
35          }
36          return 0;
37  }
38  //---------------------------------------------------------------------------
```

ANEXO D

MANUAL DE USUARIO

Software Media3D

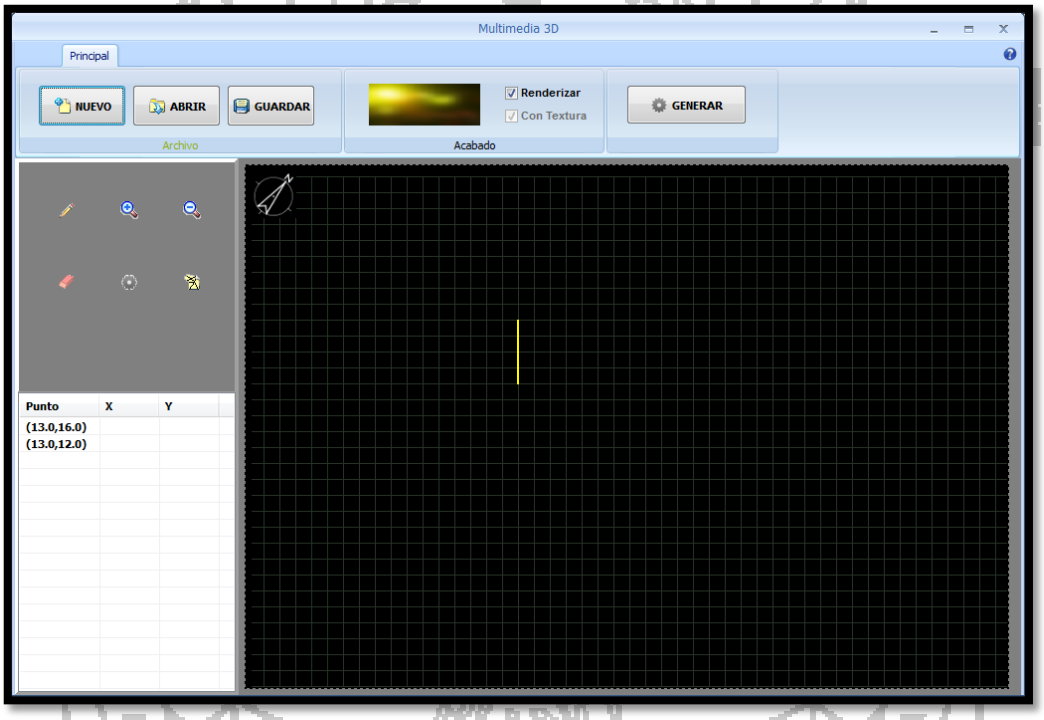
CONTENIDO

1. **Ejecución del Programa Media3D**
2. **Trazado de Figuras y Primitivas**
3. **Administración de Archivos**
4. **Selección de Texturas**
5. **Renderización y Perspectiva**
6. **Opciones Extra**

1. Ejecución del Programa Media3D

Lo primero es ejecutar la aplicación desde el explorador de archivos, tendremos una vista como:

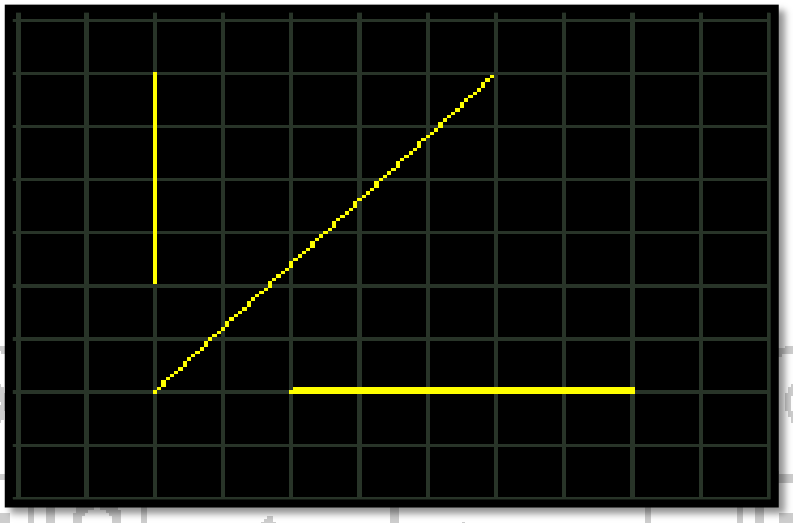
Bastara hacerle doble ***click*** sobre la aplicación y seguidamente se obtendrá la siguiente ventana.



Software Media3D para trazar figuras geométricas en 2 dimensiones, para ser renderizadas y transformadas a 3D.

2. Trazado de Figuras y Primitivas

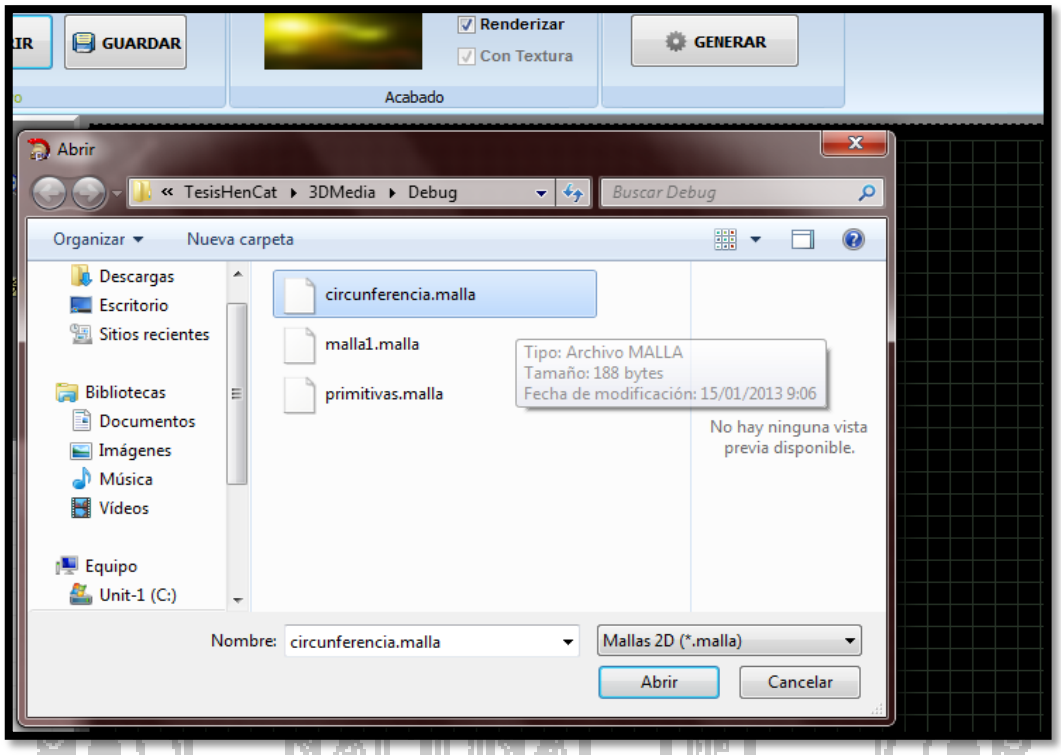
Para el trazado presione con el botón izquierdo del Mouse para establecer el punto inicial de trazado, para obtener el segundo punto de trazado presione nuevamente y se trazara una línea entre los puntos.

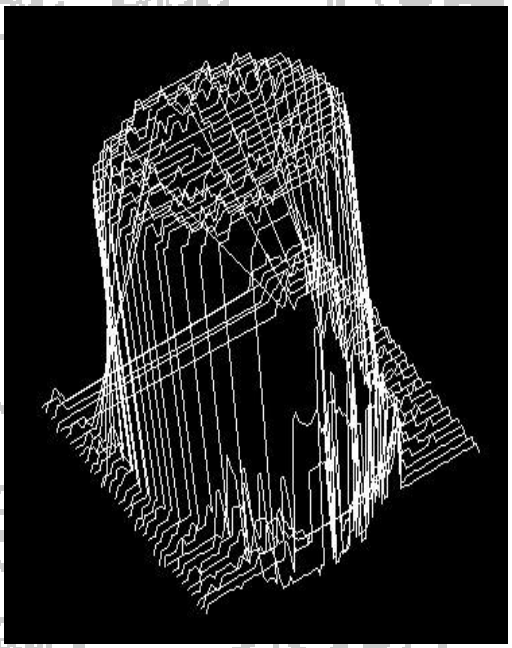


3. Administración de Archivos

Se dispone de una barra de herramientas que permitirá esta labor de forma sencilla y rápida.

Seguidamente al desear aperturar un proyecto previo se mostrara un cuadro de dialogo para seleccionar un proyecto y poder mostrar el proyecto sobre el área de dibujo.

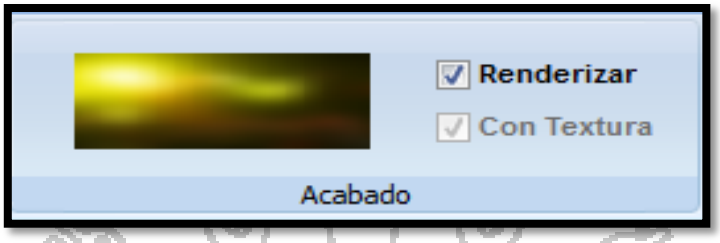




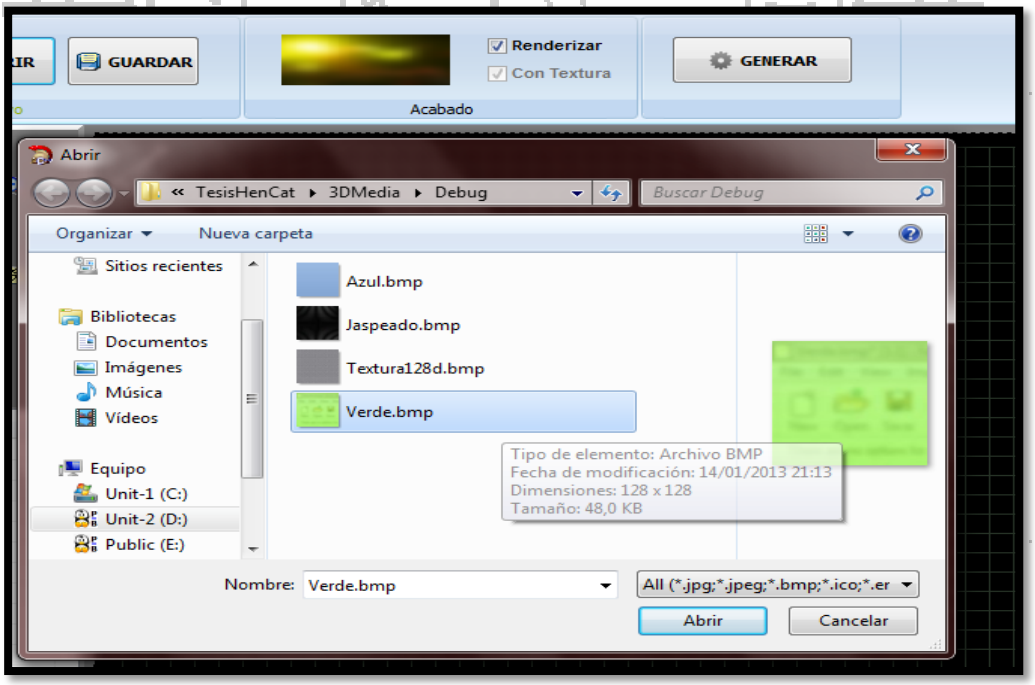
Funciona de igual forma cuando se desea guardar un proyecto que haya sido previamente diseñado en el área de dibujo de Media3D.

4. Selección de Texturas

Se dispone de una barra de herramientas que permitirá esta labor de forma sencilla y rápida.

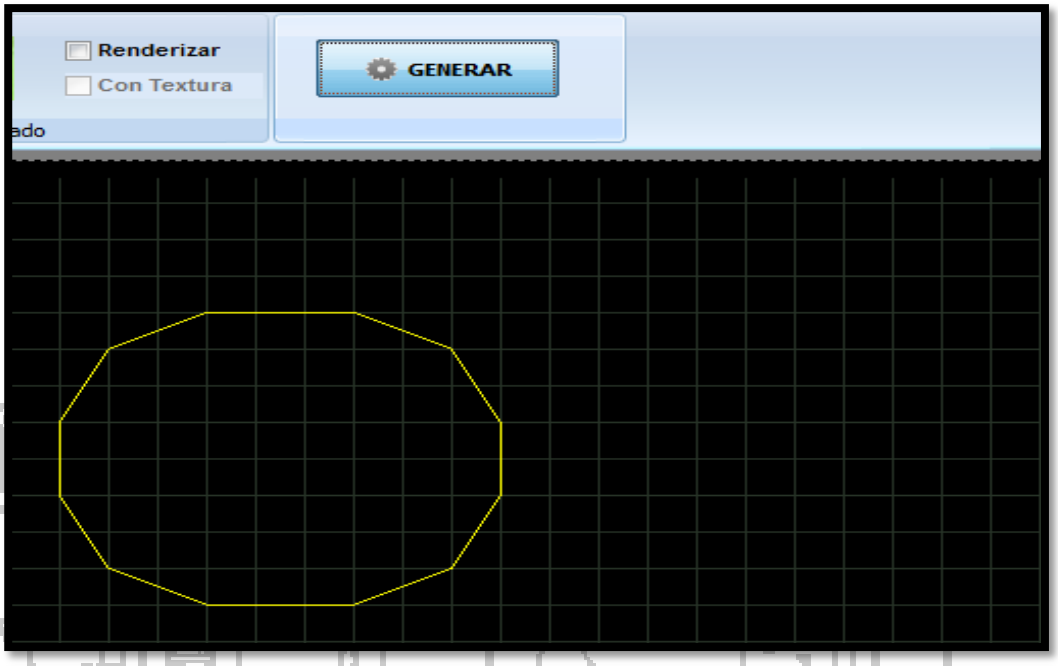


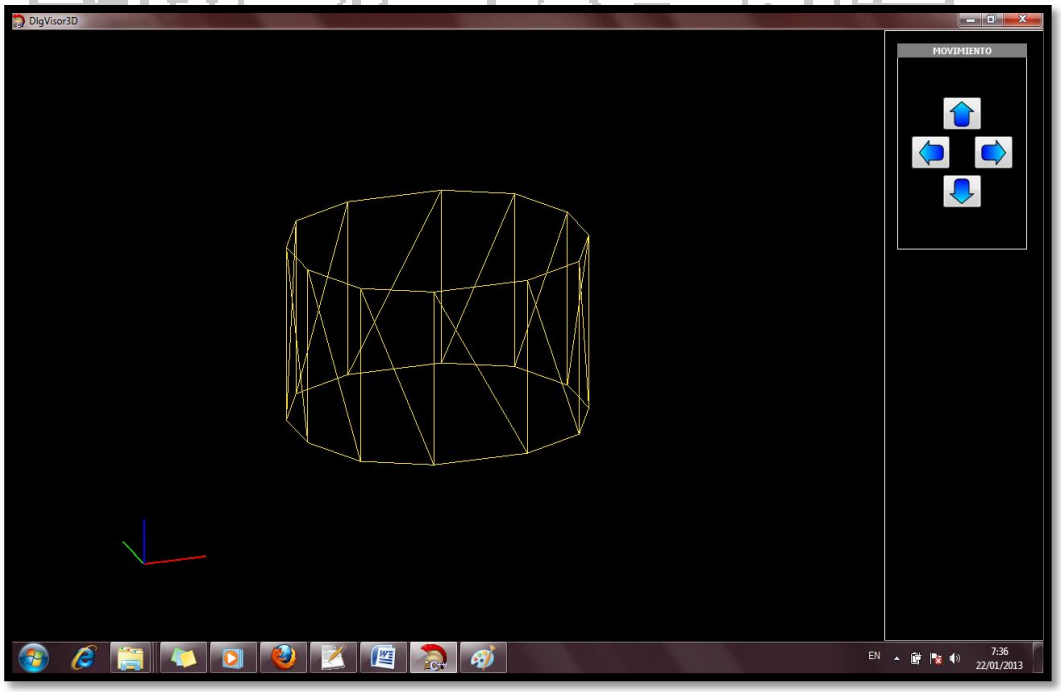
Una vez hecho ***click*** sobre la textura podrá seleccionarse otras texturas disponibles, debe tener en cuenta que las texturas deben ser archivo BMP (Portable Bitmap) con dimensiones que sean múltiplo de 4, como por ejemplo. 64x64, 128x128, 256x256, 512x512 de lo contrario la textura no podrá ser renderizada por OpenGL.



5. Renderización y Perspectiva

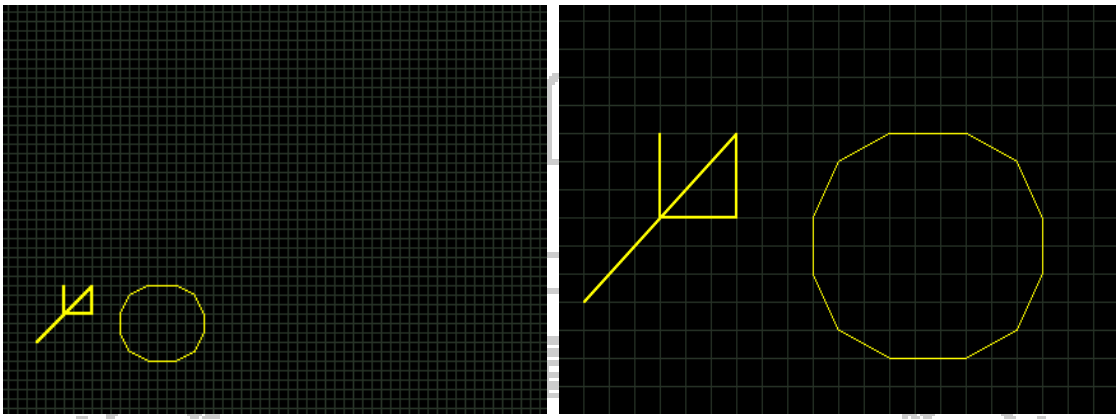
En la parte derecha de la barra de herramientas tenemos el botones que nos permitirá renderizar las figuras que tracemos para posteriormente apreciar la malla 3D generada.





6. Opciones Extra

En la parte izquierda de la aplicación podemos notar unos elementos extra que nos permitirán entre otras cosas seguir dibujando, borrar ultimo trazo, acercar, alejar y el listado de los puntos que hemos trazado, incluso para poder modificarlo en tiempo de ejecución.



Vista con Zoom Out (Alejar) y Zoom In (Acercar) para trabajar con detalle sobre polígonos mas compuestos y poder obtener una malla óptima para el renderizado y perspectiva 3D.