TRABAJO ESPECIAL DE GRADO

EXPANSIÓN DE LAS CAPACIDADES DE UN PROGRAMA DE VISUALIZACIÓN DE RESULTADOS PARA PROGRAMAS DE SIMULACIÓN NUMÉRICA MEDIANTE EL MÉTODO DE LOS ELEMENTOS FINITOS

Presentado ante la ilustre:
Universidad Central de Venezuela
Por los bachilleres:
Jaimes V. José E.
Jáuregui G. Martín G
Para optar al Título de Ingeniero Mecánico

Caracas 2009

TRABAJO ESPECIAL DE GRADO

EXPANSIÓN DE LAS CAPACIDADES DE UN PROGRAMA DE VISUALIZACIÓN DE RESULTADOS PARA PROGRAMAS DE SIMULACIÓN NUMÉRICA MEDIANTE EL MÉTODO DE LOS ELEMENTOS FINITOS

TUTOR ACADEMICO: Prof. Antonio Barragán

Presentado ante la ilustre
Universidad Central de Venezuela
Por los bachilleres:
Jaimes V. José E.
Jáuregui G. Martín G.
Para optar al Título de Ingeniero Mecánico

Caracas 2009

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE INGENIERIA
ESCUELA DE INGENIERIA MECANICA
DEPARTAMENTO DE DISEÑO

Caracas, 12 de mayo de 2.009

A C T A

Los abajo firmantes, miembros del jurado por el Consejo de Escuela de Ingeniería Mecánica, para evaluar el Trabajo Especial de Grado presentado por los bachilleres:

JOSÉ JAIMES y MARTÍN JÁUREGUI

Titulado:

"EXPANSIÓN DE LAS CAPACIDADES DE UN PROGRAMA DE VISUALIZACIÓN DE RESULTADOS PARA PROGRAMAS DE SIMULACIÓN NUMÉRICA MEDIANTE EL MÉTODO DE LOS ELEMENTOS FINITOS"

Consideran que el mismo cumple con los requisitos exigidos por el Plan de Estudios conducente al Titulo de Ingeniero Mecánico.

Prof. Rodolfo Berrios
Jurado

REPUBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD CENTRAL DE VENEZUELA
DPTO. DISEÑO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA MECÁNICA

Prof. Manuel Martínez
Jurado

Prof. Antonio Barragán
Tutor

DEDICATORIAS

Este trabajo va de dedicado:

- A mi Señor Dios todo poderoso, quien me ha demostrado que no soy dueño de la vida, sino es El quien guía mis pasos en el camino.

- A mi Padre, mi gran maestro, el ejemplo a seguir ayer hoy y siempre, sin ti Padre no estaría hoy aquí y no estaría orgulloso de ser el hombre que soy, gracias por tu apoyo incondicional, Te amo….

José Elías Jaimes Viloria

Este trabajo se lo dedico:

- A mi familia, sin el apoyo de ustedes simplemente no estaría aquí.
- A mi comunidad, ustedes son mi otra familia que siempre velan por mi.

Martín Gregorio Jáuregui Gamarra

AGRADECIMIENTOS

*A mi Tutor**, Profesor Antonio Barragán**, por la oportunidad brindada y por su apoyo desinteresado.*

*A **mi familia** que me apoyo en todo momento.*

*A **mi querida Katiuska**…. Porque detrás de un gran hombre siempre hay una gran mujer.*

*A **mi compañero Martín**, quien me demostró que la virtud de la paciencia es la que rinde más éxitos.*

A todos gracias.

José Elías Jaimes Viloria

*A ti **Dios Padre**, gracias por proveerme todo.*

*A ti **Papá, Mamá**, que me han ayudado en obtener esta meta.*

*A **Jhon Jairo, Eymar, José Miguel** y **Elías** que cada uno me ha enseñado cosas valiosas.*
*A **Eduardo, Moraima y Alexander**, por estar cerca de mí y aguantar mis malos ratos.*
*A ustedes, **Ana, Ines, Justi, Tircie** por darme siempre su apoyo.*

*A ti **Cristina**…. Cada día te quiero más.*

*A ti **pequeño rayito de sol** que me iluminabas en la sala de micros.*

*A **todos mis amigos** que han estado cerca, pendientes y se han calado a este tonto.*

*Por supuesto a usted **Profesor Antonio Barragán** por ser nuestro tutor y por compartir su conocimiento y su tiempo con nosotros, muchas gracias.*

*A **Elías** mi compañero, gracias por haberte acordado de mi. ¡¡Chamo lo logramos!!*

P.D. los que quedan por fuera de esta lista pero no de mi corazón.…… Gracias por todo.

Martín Gregorio Jáuregui Gamarra

*A la **Escuela** que nos brindo la oportunidad de trabajar en la **"Sala de Micros Cesar Ferrer"**, herramienta tan valiosa que sin ella no hubiésemos logrado nuestro objetivo.*

Elias y Martín

Jaimes V. José E., Jáuregui G. Martín G

EXPANSIÓN DE LAS CAPACIDADES DE UN PROGRAMA DE VISUALIZACIÓN DE RESULTADOS PARA PROGRAMAS DE SIMULACIÓN NUMÉRICA MEDIANTE DEL MÉTODO DE LOS ELEMENTOS FINITOS

Tutor Académico: Prof. Antonio Barragán.
Caracas, U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Mecánica. 2009. 104 Pág.

Elementos Finitos, Visualización Gráfica, Visualización Computacional, Mallado, Esfuerzo/Deformación.

En este trabajo se lleva a cabo la expansión de las capacidades de un programa visualizador de resultados obtenidos de un programa de simulación numérica mediante el método de los elementos finitos. Esta expansión consistió en triangular los elementos existentes, cuadriláteros de cuatro (4) nodos y hexaedros de ocho (8) nodos con los que solo trabajaba el programa visualizador original, e implementar y triangular los elementos adicionales con los cuales trabaja el simulador numérico, que son el cuadrilátero de ocho (8) nodos en el caso bidimensional y el hexaedro de veinte (20) nodos en el caso tridimensional. Con la finalidad de mejorar la visualización de resultados, esto porque los elementos triangulares son figuras geométricas más simples que los cuadriláteros y permiten solventar limitaciones que impone la visualización con elementos cuadriláteros. Estos objetivos se llevaron a cabo analizando el código del visualizador original, desarrollando un algoritmo de triangulación para los elementos existentes, e incorporando a este programa instrucciones que permiten el manejo de los nuevos elementos y su triangulación. La comprobación del funcionamiento del programa visualizador se hizo a través de la aplicación de análisis numérico a piezas de calibración y piezas reales, de las cuales sus mallas fueron desarrolladas en este trabajo.

Jaimes V. José E., Jáuregui G. Martín G

EXPANSION OF THE CAPABILITIES OF A PROGRAM OF VISUALIZATION OF RESULTS FOR PROGRAMS OF NUMERICAL SIMULATION BY THE METHOD OF FINITE ELEMENTS

Academic Tutor: Prof. Antonio Barragán.
Caracas, U.C.V. Faculty of Engineering. School of Mechanical Engineering. 2009.
104 Pages.

Finite elements, Graphical Display, Computational display, Mesh, Effort / deformation.

In this work is carried out the expansion of the capabilities of a program which displays results of a program of numerical simulation using the method of finite elements. This expansion consisted in making triangles of existing elements, quadrangles of four (4) nodes and hexahedrons of eight (8) nodes with which only worked the original displayer program, and consisted in implementing and making additional triangular elements with which works the numerical simulator, which are the quadrilateral of eight (8) nodes in the two dimensional case and the hexahedron of twenty (20) nodes in the three-dimensional case. In order to improve the display of results, this because the triangular elements are geometric figures simpler than may be quadrangles, and they allow solving constraints imposed by the visualization with elements quadrangles. These objectives were carried out by analyzing the code of the original displayer, developing an algorithm for triangulation of existing elements, and incorporating instructions to the program that enable the handling of the new elements and their triangulation. The verification of the displayer program's operation was made through the implementation of numerical analysis to the calibration parts and actual pieces, of which their meshes were developed in this work.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO I
INTRODUCCIÓN

En la escuela de ingeniería mecánica de la U.C.V, se ha venido desarrollando en los últimos años un grupo de programas para el análisis de esfuerzos haciendo uso de la simulación numérica a través del método de elementos finitos.

Las funciones de este paquete de programas se pueden clasificar en tres áreas fundamentales:

1. Pre procesamiento: área que se encarga de la generación de mallas de elementos para su posterior análisis así como de la generación de las condiciones de contorno.
2. Procesamiento: área encargada del cálculo numérico de los esfuerzos y deformaciones en la malla generada en el pre procesamiento
3. Post procesamiento: área encargada de tomar los datos emitidos en procesamiento y procesarlos de una manera tal que el usuario del programa tenga una comprensión clara del análisis realizado y que generalmente esto se logra a través de una visualización gráfica.

Actualmente se cuenta con un programa de visualización de resultados para programas de simulación numérica, el cual funcionaba solo para dos tipos de elementos, uno bidimensional el cual es un cuadrilátero de cuatro (4) nodos, y otro tridimensional el cual es un hexaedro de ocho (8) nodos, mientras que el simulador (programa MEFET) puede manejar hasta cuatro tipos de elementos, los otros elementos son cuadrilátero de ocho (8) nodos y hexaedro de veinte (20) nodos. Tomando en cuenta lo anterior, se observa que es necesario

implementar nuevos tipos de elementos para su visualización en el programa visualizador.

Con vista a mejorar las prestaciones de este programa de visualización, se propone la expansión de las capacidades del programa computacional que permite visualizar los resultados de programas de simulación numérica mediante el método de los elementos finitos u otro método que use mallado y cuyos resultados se obtengan en los nodos de la malla.

Esta expansión de las capacidades va a ser lograda además de implementando nuevos elementos para su visualización, realizando también la triangulación de las superficies de los elementos, técnica ampliamente utilizada en el área de la computación gráfica por las ventajas que presenta.

CAPÍTULO II

ANTECEDENTES

Desde el principio del desarrollo de las matemáticas y la geometría, las relaciones encontradas en ellas permitieron representar problemas matemáticos de forma gráfica y problemas gráficos de forma matemática, esto crea las bases de la graficación de problemas de índole matemático o numérico.

Con el advenimiento de las herramientas de cálculo moderna (calculadoras, computadoras, procesadores, entre otros) muchos problemas científicos y técnicos fueron abordados con métodos "automatizados" de cálculo, que valiéndose de la gran capacidad y velocidad de cálculo de los nuevos procesadores, se desarrollo una nueva herramienta científica y de ingeniería. Los métodos numéricos, en los cuales muchos cálculos pueden ser realizados, produciendo un gran volumen de resultado. Estos métodos permiten evaluar el fenómeno estudiado en muchos puntos de muestreo, que comúnmente forman el Volumen de Control o el sólido a estudiar, recorriendo el espacio de estudio y obtener soluciones que de manera manual sería imposible o inviable de implementar en la práctica.

Dado el gran poder que tienen estos métodos para resolver problemas de ingeniería, economía, ciencias y otros, el desarrollo de software capaz de llevar a cabo estos métodos ha generado opciones de paquetes que permiten resolver problemas de diversa índole. Sin embargo la posibilidad de uso de los diversos paquetes se ve limitada por el costo de las licencias necesarias para su uso y la accesibilidad de las mismas en nuestro país.

En la Escuela de Ingeniería Mecánica de la UCV se han venido desarrollando estudios en torno a la implementación de los métodos numéricos como herramienta para resolver problemas de ingeniería mecánica. Mediante varios trabajos especiales de grado, se ha desarrollado software que permite resolver los problemas de ingeniería mediante métodos numéricos en las áreas de: mecánica de los fluidos, mecánica de los sólidos, transferencia de calor, bioingeniería, entre otros, con el fin de potenciar el desarrollo de investigaciones aplicadas a estos campos.

Alonzo y Oramas [4] elaboraron un programa para representar gráficos bi y tridimensionales aplicables a componentes mecánicos y bioingeniería, además este programa es capaz de realizar rotaciones, traslaciones, reflexiones, cambios de escala, visibilidad y aproximaciones a curvas. Además se pueden utilizar aparte de formas geométricas preestablecidas formas combinadas de las mimas.

Noya L. y Rabinovich [12] desarrollan un método de cálculo para problemas no lineales, basado en el Método de los Elementos Finitos. Se implementó un programa de computación que permite calcular los desplazamientos y esfuerzos tanto en la región de contacto como en el resto de los cuerpos en órganos dentarios.

Ordáz y Pulido [13] se muestran soluciones a ciertos problemas geométricos asociados a la generación automática de mallas tridimensionales de elementos finitos, como lo son, determinación de la posición de un punto respecto a un sólido, determinación de la posición de un sólido respecto a un plano orientado, intersección entre: plano y plano, plano y sólido, recta y sólido,

recta y plano. Tales soluciones son logradas mediante la implantación de un programa en lenguaje C el cual trabaja con estructuras dinámicas de datos.

Puig y Pey [14] desarrollan un algoritmo para representar gráficamente cualquier función de dos variables en un sistema cartesiano con salida por impresora y pantalla, presentando los valores de la función por medio de bandas alfanuméricas obtenidas de una tabla previamente normalizada según el valor asociado de función. La representación de los valores de la superficie se obtiene mediante intersección de planos horizontales con la superficie, generando bandas alfanuméricas de valor constante quedando la calidad de los resultados en función del número de planos.

Lynes y Asquith [9] realizaron un algoritmo para realizar un trazado discreto lineal de curvas que representan los resultados obtenidos por el método de los elementos finitos. El algoritmo aplicado a zonas en las cuales no hubo una buena discretización genera una curva suave evitando así las oscilaciones debido a la no muy buena continuidad de esfuerzos en esa región. Este fue aplicado tanto a distribuciones de esfuerzos, temperaturas y a curvas de nivel.

Yeo [18] desarrollo un método para la representación de isolíneas basado en la aproximación lineal en los elementos analizados. El algoritmo propuesto se presenta como una alternativa a la rapidez exigida en los programas post – procesadores. Supone que la región que se va a analizar puede ser dividida en subregiones triangulares de tres nodos. La versatilidad del programa demuestra la gran importancia en el desarrollo de las matrices de conectividad para el análisis de mallas. El programa está dirigido a la

utilización de trazadores (plotters) y simulan la utilización de elementos de orden superior.

Akin y Gray [1] presentaron un procedimiento general para el trazado de isolíneas sobre superficies isoparamétricas utilizando para ello funciones de interpolación isoparamétricas, su teoría se basa en el uso de algebra lineal y transformación de coordenadas. Presento el incoveniente de generar espirales en vez de curvas cerradas producto del error que se comete al suponer incrementos sobre la misma curva (isolíneas). Su aplicación fue a elementos triangulares que modelaron las regiones llamadas de “ensilladuras”.

Akin y Gray [2] presentaron un mejora al método desarrollado en su trabajo anterior [1], haciendo una corrección analítica sobre la generación de puntos sobre la isolínea . La operación realizada diminuye el error creado en la generación de puntos mediante un algoritmo que genera puntos y automáticamente va corrigiendo la dirección en la cual se van generando. Este método es llamado predictor – corrector. Los resultados obtenidos demostraron lo acertado de este método y se llega a emplear como una herramienta matemática para el desarrollo de las representaciones gráficas.

Nakamae, Yamashita, Nobuyuki y Nakano [11] realizaron un trabajo para graficar isocromáticas mediante trazadores (plotters) a color y tarjetas graficadoras en superficies de 2 y 3 dimensiones aplicados a líneas de flujos magnéticos, con la característica de representar simultáneamente la magnitud, dirección y densidad de los campos magnéticos. En el caso de las representaciones en pantalla, los elementos triangulares de orden superior de seis nodos son divididos en triángulos más pequeños pero de tres nodos cada

uno y son llenados mediante un color que representa un valor de función, el cual es obtenido como valor promedio de los valores nodales del elemento. Este color se obtiene de una tabla que relaciona valor nodal/color.

Stelzer y Welzel [17] diseñaron un procedimiento para representación de isolíneas producto de los resultados obtenidos a través del método de los elementos finitos para el análisis de cualquier tipo de variables. Su trabajo incluyó el cálculo de coordenadas locales mediante el uso de funciones de interpolación y superficies isoparamétricas para elementos de 4 y 8 nodos. También se estudio la utilización de las coordenadas de pantalla en dos y tres dimensiones. Las aplicaciones directas fueron en el análisis de distribuciones de temperaturas y de esfuerzos.

Akin, Gray y Zhang [3] desarrollaron un algoritmo para la representación gráfica de variables mediante un color asociado sobre superficies isoparamétricas. Su basamento teórico recae en la utilización del método PREDICTOR – CORRECTOR. El cual genera puntos sucesivos de una isolínea usando funciones de interpolación isoparamétricas y sus derivadas, para producir trazos continuos de color sobre la superficie. Estos trazos se generan como una línea de barrido equivalente a una horizontal en la pantalla del computador y también sobre impresoras estándar (representados en forma de caracteres alfanuméricos), sin colores asociados. Su aplicación fue en la representación de resultados obtenidos a través del uso del método de los elementos finitos en elementos triangulares de tres nodos y elementos cuadriláteros de cuatro nodos.

Stelzer [16] desarrolla un programa para la representación de bandas a color de los resultados de análisis por elementos finitos. El método empleado se basa en la subdivisión de todos los elementos en dos y tres dimensiones en una cantidad libre de sub campos y luego realizar un llenado de estos campos utilizando la técnica de llenado de polígonos con colores establecidos en una tabla de correspondencia color/valor. Se hicieron análisis de distribuciones de esfuerzos y temperaturas.

Kenneeth y Croos [8] usa un modelo computacional en el cual expone un método para interpretar resultados numéricos. Este método tiene su origen en la representación de superficies de llenado de polígonos. Utilizando este principio, el autor, desarrolla una modalidad de la interpolación lineal para darle color a una línea de barrido que se mueve sobre un elemento. Se ilustra su trabajo utilizando elementos cuadriláteros de cuatro nodos en distribuciones de temperatura y flujo magnético.

Quiroz y Dufeo [15] presentan las consideraciones que deben tomarse en cuenta para construir un sistema post procesador gráfico para el análisis de elementos finitos y para la utilización del programa SAP – IV (Programa general de elementos finitos). Muestran algunos algoritmos recomendados para facilitar la obtención de gráficos como son: la representación de curvas en general de dos y tres dimensiones, la eliminación de líneas ocultas en representaciones tridimensionales, la representación de curvas de deformaciones de elementos de vigas y tuberías y un algoritmo para determinar los puntos de cambio de nivel de valores nodales en la superficie de un objeto. También se analizo la independencia que debe existir entre las salidas gráficas del sistema con respecto a los periféricos.

Jaén y López [7] elaboran un sistema postprocesador gráfico para representar los resultados arrojados de la aplicación del Método de los Elementos Finitos al análisis de esfuerzos. Utilizaron una metodología basada en conceptos propios del método, como lo es la representación de Superficies Isoparamétricas mediante funciones de interpolación.

Barragán [5] presento técnicas de postprocesamiento para ser utilizadas en la representación gráfica de los resultados de aplicación del método de los elementos finitos. Con esto se persigue facilitar la interpretación de los grandes volúmenes de resultados, típicos del método. El objetivo del postprocesamiento consiste en dar una representación visual de los resultados numéricos mediante bandas de colores (isozonas), y se realiza con base en el elemento finito de cuatro lados (4 y 8 nodos).

Martín [10] quien desarrolló un programa de visualización de resultados para programas de simulación numérica con la capacidad de mostrar en pantalla resultados obtenidos mediante el método de los elementos finitos y otros métodos, para mallas de elementos cuadriláteros de cuatro nodos y hexaedros de ocho nodos.

CAPÍTULO III

MARCO TEÓRICO

Método de los Elementos Finitos

Los cálculos realizados en el área de la ingeniería ya sea para la fabricación, diseño y/o control de procesos y productos también se han visto beneficiados por el uso del computador, específicamente el área de análisis numérico ha permitido el uso de técnicas como el **Método de Elementos Finitos**

El **Método de Elementos Finitos**, es una técnica de simulación numérica en la cual una pieza geométrica, estructural o una región en el espacio es dividida en porciones menores (elementos), en los cuales pueden ser aplicados métodos de aproximación para hallar los valores de la variable en estudio, por lo tanto es un procedimiento que no depende de las variables o el fenómeno en estudio y es tomado como un procedimiento general.

En un estudio por el MEF, se deben realizar tres pasos:

- Pre-proceso: que consiste en la generación de la malla que luego será utilizada en los procesos de cálculo.
- Proceso: Resolución del sistema de ecuaciones que representa el fenómeno en estudio.
- Post-proceso: Corresponde a la presentación de los resultados obtenidos en el proceso de cálculo, en una forma fácilmente comprensible como la visualización científica. Un estudio realizado a una pieza mecánica puede dar como resultado una tabla de varias decenas de miles de puntos en el espacio, por lo tanto, las herramientas que permitan

condensar esta información y presentarla de manera sencilla resultan vitales tanto para el análisis de los cálculos hechos como para la comunicación y publicación de los mismos.

Visualización Gráfica:

La computación gráfica ha avanzado mucho desde sus primeros inicios en los años 60, gracias a los avances tecnológicos de las últimas décadas, se ha logrado avanzar en pasos agigantados respecto a la visualización gráfica en computación, tanto en lo que se refiere a hardware y software. Gracias a estos avances hoy se logran generar imágenes que hace 20 años eran imposibles de realizar, y que hoy producen una sensación de realidad en dichas imágenes que a cualquier usuario podrían engañar.

Respecto al software de visualización desarrollado para la visualización gráfica computacional, se han implementado técnicas y algoritmos de programación que unido al hardware ofrecen capacidades casi ilimitadas de graficación.

Dentro del ámbito de la graficación computacional, el desarrollo de la visualización en tres dimensiones (3D) ha sido el campo en donde se han aunado más esfuerzos para su desarrollo, tanto en el área de hardware como de software. Una prueba de ello es que la inmensa mayoría de la plataforma tecnológica disponible en el mercado está enfocada y diseñada para el manejo adecuado de gráficos 3D.

Las potencialidades de la graficación son posibles gracias a dos archivos del computador: La API y el controlador, la API (del inglés Application Programming Interface), es una biblioteca de funciones de muy bajo nivel que permite a los desarrolladores de programas utilizar las potencialidades del

hardware grafico del pc; por su parte el controlador (uno o varios archivos tipo *.dll), es el que implementa la comunicación entre el programa en ejecución y el hardware gráfico, para llevar a cabo las funciones de la API respectiva.

En los que respecta a computación gráfica, las tres APIs principales que han sido desarrolladas son:

- **Open GL**: (Open Graphic Library) desarrollada en principio por Silicon graphics Inc. Y convertida luego en estándar del mercado.
- **Direct 3D**: Desarrollada por Microsoft Corp. Con el fin de competir con Open GL, en el mercado de las aplicaciones gráficas.
- **RenderMAN**: desarrollada por los estudios de animación Pixar, está principalmente orientada a la animación y ha permitido a los estudios Pixar la creación de películas que han marcado pauta en la animación cinematográfica.

La biblioteca de aplicaciones gráficas Open GL fue diseñada para las estaciones de trabajo de Silicon Graphics, Iris Work Station, llamándose Iris Graphics Library, luego, en un esfuerzo por hacer el código portable a otras plataformas tanto propias como de otros fabricantes, nace Open GL como software para implementación de gráficos por computador independiente de la plataforma (hardware).

Luego de que se estableciera como estándar del mercado, Open GL aún es desarrollado por el ARB (Architecture Review Board), conformado en principio por: Microsoft Corp, Silicon Graphics Inc., Hewlet-Packard, Digital Equipment Corporation e IBM, y luego por las empresas fabricantes de hardware gráfico. Este equipo multidisciplinario ha llevado las especificaciones de la API desde su primera publicación a lo que es hoy en día el estándar Open

GL 2.0, añadiendo cada vez más características que desarrolladores independientes de software han creado y difundido.

La API Open GL comprende el manejo de información en dos y tres dimensiones que permite crear de forma estructurada geometrías complejas, y manipularlas de manera conveniente para obtener efectos como: proyecciones perspectivas e isométricas, rotación, escalamiento, deformación, etc. Incluye el manejo del color, sombras, efectos de iluminación etc. Que crean con realismo escenas de geometría que asemejan situaciones realistas, permite la optimización del hardware para la implementación del movimiento, creando animaciones y secuencias de imágenes. Por último, es posible interactuar con el objeto dibujado de manera gráfica a fin de implementar aplicaciones como manipular la información e interacción con objetos animados de juegos, etc.

En el desarrollo de software de visualización se emplean una amplia gama de algoritmos y técnicas para la visualización gráfica, en este trabajo podemos mencionar el empleo de la triangulación de Delaunay, el llenado de polígonos y el método de Sombreado de Phong. Estas técnicas son usadas para la visualización.

Triangulación de Delaunay

Se le denomina así por el matemático ruso Boris Nikolaevich Delone (Борис Николаевич Делоне, 1890 - 1980) quien lo inventó en 1934 [6].

Una 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 para ordenar, especialmente en gráficos 3D por computadora.

La aplicación de la triangulación de Delaunay se debe a que en gráficos por computadora se usan redes de polígonos para modelar objetos tridimensionales, juntando los polígonos para imitar la superficie del objeto. En general se usan triángulos porque son los polígonos más simples y tienen muchas propiedades favorables, como que representan una superficie coplanar.

La triangulación de Delaunay maximiza los ángulos interiores de los triángulos de la triangulación. Esto es muy práctico porque al usar la triangulación como modelo tridimensional los errores de redondeo son mínimos. Por eso, en general se usan triangulaciones de Delaunay en aplicaciones gráficas.

Según la definición de Delaunay la circunferencia circunscrita a un triángulo es *vacía*, si no contiene otros vértices aparte de los tres que la definen (Fig. 1).

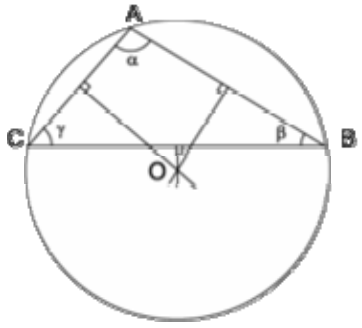
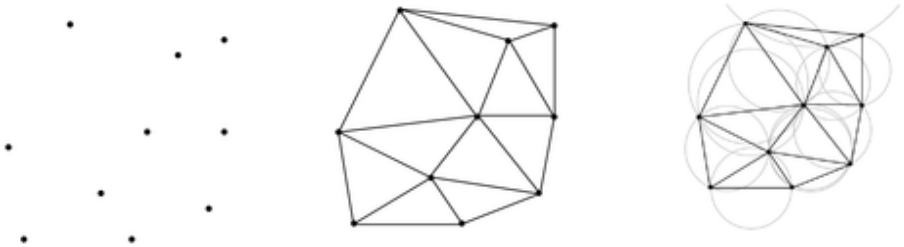


Fig. 1.- Circunferencia circunscrita a un triángulo

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(Fig 2). Es posible ampliarla para espacios tridimensionales usando la esfera circunscrita en vez de la circunferencia circunscrita.



De algunos puntos se quiere construir una triangulación.

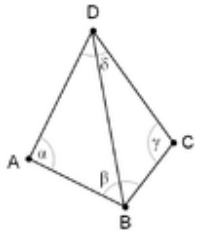
Es fácil construir cualquiera triangulación simplemente conectando los vértices.

Con la condición de Delauany se puede examinar si la triangulación es útil.

Fig. 2.- Condición de Delaunay

Esta 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.

De la geometría de los triángulos se puede deducir una característica importante: Contemplando dos triángulos ABD y BCD con la arista común BD, si la suma de los ángulos α y γ es menor o igual a 180°, los triángulos cumplen la condición de Delaunay. Eso es importante porque de esta propiedad se puede deducir el *flipping* (del inglés *flip* «invertir»). Si los dos triángulos no cumplen la condición de Delaunay, reemplazando la arista común BD por la arista común AC produce una triangulación de Delaunay:



Esta triangulación no cumple la condición de Delaunay.

Esta triangulación no cumple la condición de Delaunay.

Flipping de la arista común produce una triangulación que cumple la condición de Delaunay.

Fig. 3.- Flipping

Llenado De Polígonos

La tarea del algoritmo de llenado de polígonos [6] se puede separar en dos partes:

1. La decisión de que pixeles llenar (esto depende de la forma de la primitiva), y
2. La decisión más sencilla de cual valor utilizar para el relleno.

En general, determinar que pixeles llenar consiste en tomar líneas de rastreo sucesivas que interceptan la primitiva y llenar en intervalos (spans) de pixeles adyacentes que están dentro de la primitiva de izquierda a derecha.

Para llenar un rectángulo con un color sólido, se asigna a cada pixel sobre una misma línea de rastreo desde el borde izquierdo al borde derecho el mismo valor de pixel; o sea llenamos cada intervalo de xmin a xmax.

Se aprovecha de varios tipos de coherencias, coherencia se refiere al grado en que una escena o un objeto graficado presenta similitudes locales, no solamente para convertir primitivas de 2D, también de 3D.

- Los intervalos explotan la coherencia espacial (spatial coherence) de una primitiva: el hecho que las primitivas a menudo no cambian de pixel en pixel dentro de un intervalo o de línea de rastreo a línea de rastreo. Se explota esta coherencia buscando solo aquellos pixeles donde ocurren cambios.

- Para una primitiva trazada de forma sólida, se asigna el mismo valor a todos los pixeles en un mismo intervalo, proporcionando coherencia de intervalo (span coherence).

- Un rectángulo trazado de forma sólida también muestra una fuerte coherencia de línea de rastreo (scan-line coherence) ya que líneas de rastreo consecutivas que intersectan el rectángulo son idénticas; más tarde se usa también coherencia de arista (edge coherence) ya que la visibilidad de una arista visible cambia cuando se intersecta con otra arista visible, para los lados de polígonos generales.

La capacidad de manejar múltiples pixeles en un intervalo de forma idéntica es especialmente importante porque se debe escribir el frame buffer una palabra a la vez para minimizar el número de accesos a memoria.

La siguiente figura ilustra el procedimiento de la línea de rastreo para el llenado sólido de polígono.

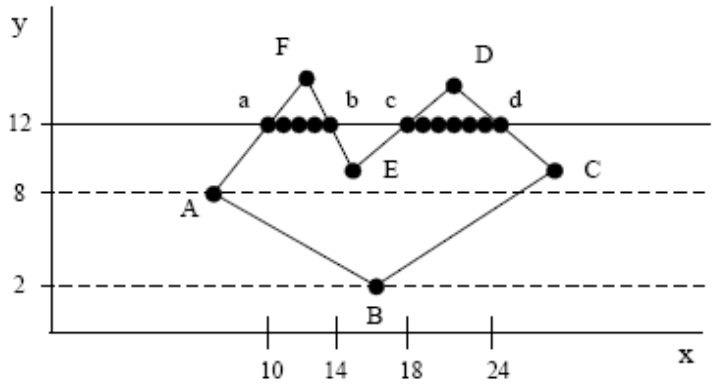


Fig 4 Llenado de Polígonos

Para cada línea de rastreo que cruza un polígono el algoritmo de llenado de áreas debe:

- Localizar los puntos de intersección de la línea de rastreo con las aristas del polígono,
- Definir los intervalos de llenado (spans) para cada línea de rastreo.

Esto se repetiría para todas las líneas de rastreo que intersectan el polígono.

Método de Phong

Un método más exacto para generar una superficie de polígono es el de interpolar vectores normales a ésta, y luego aplicar el modelo de iluminación a cada punto superficial. Este método, desarrollado por Phong Bui Tuong, llamado el sombreado de Phong [6], o de sombreado de interpolación del vector normal. Esto muestra toques de luz más realistas sobre una superficie y reduce

enormemente el efecto de Bandas de Mach, ilusión óptica que parte de una imagen con dos bandas, una iluminada y la otra oscura, separadas por una estrecha banda central coloreada con un gradiente de iluminado a oscuro. El ojo humano percibe dos estrechas bandas de diferente luminosidad, que no están presentes en la imagen verdadera, a cada lado del gradiente.

Fig. 5.- Bandas de Mach

Una superficie poligonal es obtenida usando el método de sombreado de Phong realizando los pasos siguientes:

- Determina el promedio del vector unitario normal en cada vértice del polígono.
- Interpola linealmente las normales de vértice sobre la superficie del polígono.
- Aplica un modelo de iluminación a lo largo de cada línea de exploración (rastreo) para calcular intensidades de pixel proyectadas para los puntos superficiales.

La interpolación de normales a superficiales a lo largo de un borde de polígono entre dos vértices es ilustrada en la figura 1. El vector normal para el punto de intersección de línea de exploración a lo largo del borde entre los vértices 1 y 2 puede ser obtenido interpolando verticalmente entre normales de los extremos de la línea de borde o frontera.

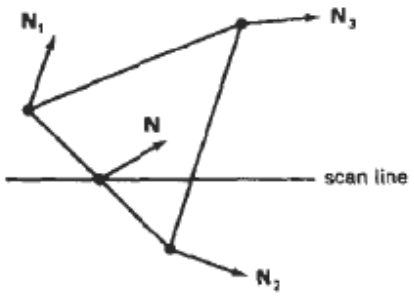


Fig 6.- Escaneo de Normales Método de Pong

CAPÍTULO IV

DESARROLLO DEL PROGRAMA

Funcionamiento General Del Programa De Visualización

El programa original de visualización de resultados para programas de simulación numérica mediante el método de los elementos finitos (MySmarting) funciona básicamente con los siguientes pasos.

El programa debe recibir dos archivos, uno de datos y uno de resultados, los cuales contienen la información procesada por el programa simulador en donde este calcula los esfuerzos y deformaciones en la pieza a estudiar. El archivo de datos contiene las especificaciones del tipo de problema a estudiar, las lista de los nodos de la malla con sus coordenadas, la lista de los elementos junto con la conectividad de sus nodos, la lista de los nodos cargados y la lista de nodos restringidos. El archivo de resultados tiene los valores de desplazamiento nodales y los valores de esfuerzo en los nodos.

Una vez que el programa es puesto en modo de ejecución, procesa estos dos archivos y presenta en pantalla un ventana la cual se subdivide en dos, la ventana principal que posee la vista de la pieza y una subventana en el lado superior derecho con los valores referenciales de esfuerzo comprendidos entre el mínimo y el máximo valor del esfuerzo que se registra en el archivo de resultados, los cuales se interpolan a una escala de colores que se plasma en una barra vertical dentro de esta subventana. En la ventana principal se observa la pieza, que puede ser apreciada en tres formatos específicos.

El primer formato es el de la pieza sólida, en donde la pieza muestra la superficie rellena con colores según la escala de color antes mencionada. Estos colores pueden variar según un conjunto de paletas preestablecidas que el usuario puede seleccionar según sea su conveniencia al momento de observar la pieza sometida a estudio.

El segundo formato es la pieza mostrada con su mallado, donde solo se observan las aristas de los elementos, de igual manera se utiliza la escala de colores para rellenar estas aristas. En este segundo formato se aprecia claramente el tipo de elemento con el cual el visualizador trabaja que para este caso son el cuadrilátero de cuatro (4) nodos para los análisis de tipo bidimensional y hexaedros de ocho (8) nodos para los análisis de tipo tridimensional.

El tercer formato consiste en presentar la pieza mostrando solo los nodos, los cuáles también se muestran coloreados según la escala.

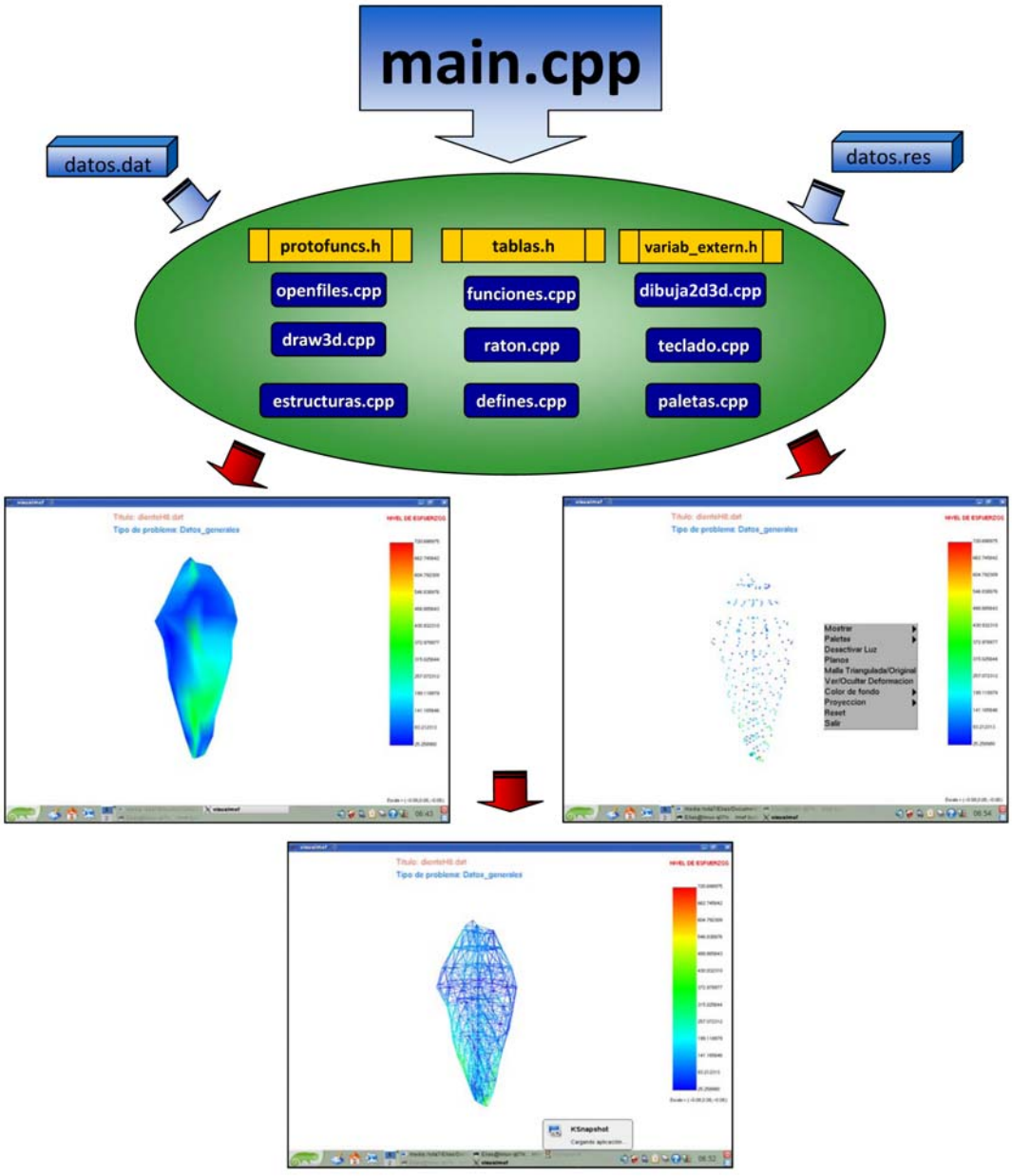


Fig. 7.- Diagrama de funcionamiento general del programa

DESARROLLO DEL PROGRAMA

El programa a mejorar, está desarrollado en lenguaje C++ y originalmente estaba compuesto por varios archivos, los cuales cumplen labores específicas e indispensables para su funcionamiento. Estos archivos son:

1. main.cpp
2. openfiles.cpp
3. funciones.cpp
4. dibuja2d3d.cpp
5. draw3d.cpp
6. raton.cpp
7. teclado.cpp
8. estructuras.h
9. defines.h
10. paletas.h
11. protofuncs.h
12. variab_extern.h
13. datos.dat (nombre genérico)
14. datos.res (nombre genérico)

ARCHIVO main.cpp.

Este archivo contiene las instrucciones principales y el orden de ejecución de las funciones del programa. Inicialmente debemos definir el programa como un conjunto de instrucciones introducidas al computador que le permiten realizar un proceso.

A continuación se describen de una manera global las instrucciones el archivo **main.cpp**.

a) Llamada a bibliotecas de funciones propias del lenguaje de programación, OpenGL y archivos de cabecera desarrollados para este programa.
b) Declaración de variables generales.
c) Definición de la función **main**, esta es la función central encargada de organizar y ejecutar todas las instrucciones para que el visualizador funcione. A gran escala esta función esta organizada así:
    - Llamada a la función que recibe y almacena en memoria los archivos de datos y resultados (openfiles).
    - Cálculo del factor de interpolación de color (openfiles).
    - Llamado a inicialización de las bibliotecas gráficas.
    - Mostrar en consola el menú de ayuda.
    - Llamado a funciones de OpenGL, encargadas de dibujar la ventana principal, colocar los títulos, dibujar el objeto que se desea visualizar, cargar las funciones de interfaz de usuario, teclado y mouse y dibujar la subventana que contiene la paleta de colores con la escala de esfuerzos.
d) Definición de la función **display** encargada de dibujar la ventana principal y llamar a la función que dibuja el objeto a visualizar.
e) Definición de la función **render** la cual especifica a cual función llamar para dibujar el objeto, caso bidimensional o caso tridimensional.

f) Definición de la función **reshape** encargada de redibujar la ventana cuando sea necesario.
g) Definición de la función **init** la cual es la función de inicialización.
h) Definición de la función **displaysubwindow** encargada de dibujar la subventana que contiene la paleta de colores utilizada, valores de esfuerzos y escala del objeto visualizado.

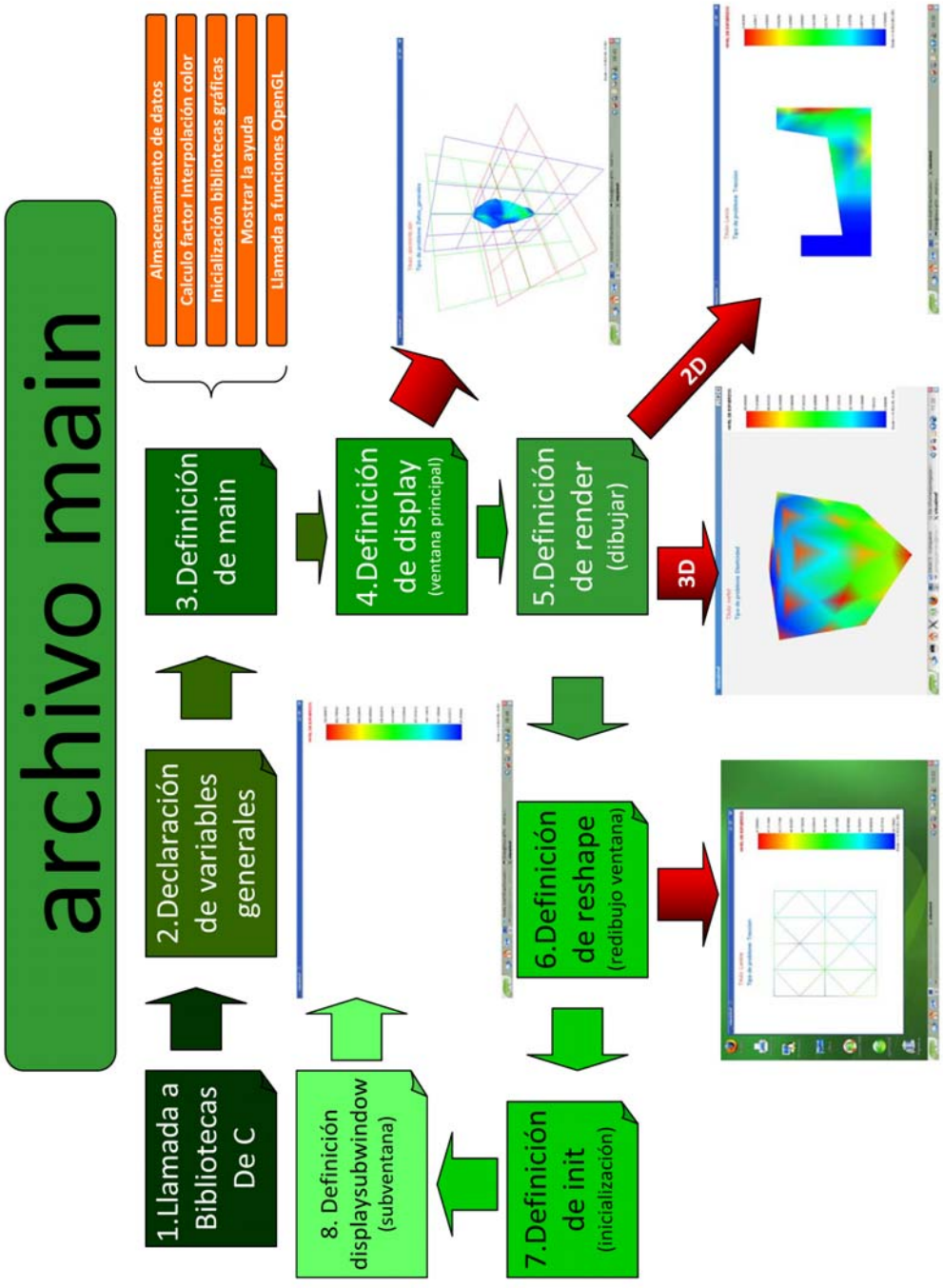


Fig. 8.- Diagrama de bloque del funcionamiento del archivo *main.cpp*

ARCHIVO openfiles.cpp.

Este archivo contiene la rutina *openfiles* la cual tiene las instrucciones para leer y almacenar los datos de los archivos que recibe del simulador. Esta información viene en dos archivos, el primero contiene toda la información que define la geometría de la pieza, y el segundo posee la información de las deformaciones y los esfuerzos en la pieza a estudiar.

Otra de las funciones que posee esta rutina es la llamada a calcular los vectores normales a cada nodo, esto con la finalidad de usar luego el valor de esta normal para el cálculo de la iluminación y el color de la pieza en cada nodo y superficie generada.

A continuación se describen de una manera global las instrucciones en el archivo **openfiles.cpp**.

a) Llamada a bibliotecas de funciones propias del lenguaje de programación, OpenGL y archivos de cabecera desarrollados para este programa.
b) Definición de la función *openfiles*
c) Declaración de variables
d) Verificación de existencia de archivos de datos.
e) Lectura de datos de cabecera del primer archivo.
f) Limpieza e inicialización de variables tipo estructuras de almacenamiento de datos.

g) Lectura y almacenamiento del primer bloque de datos del primer archivo donde están el número de los nodos y las coordenadas que lo componen.

h) Cálculo del centro de coordenadas de la pieza.

i) Lectura y almacenamiento del segundo bloque de datos del primer archivo donde están el número de los elementos y la conectividad de los nodos que lo componen.

j) Cálculo del vector normal en cada nodo.

k) Lectura y almacenamiento de los desplazamientos y esfuerzos en el segundo archivo de datos.

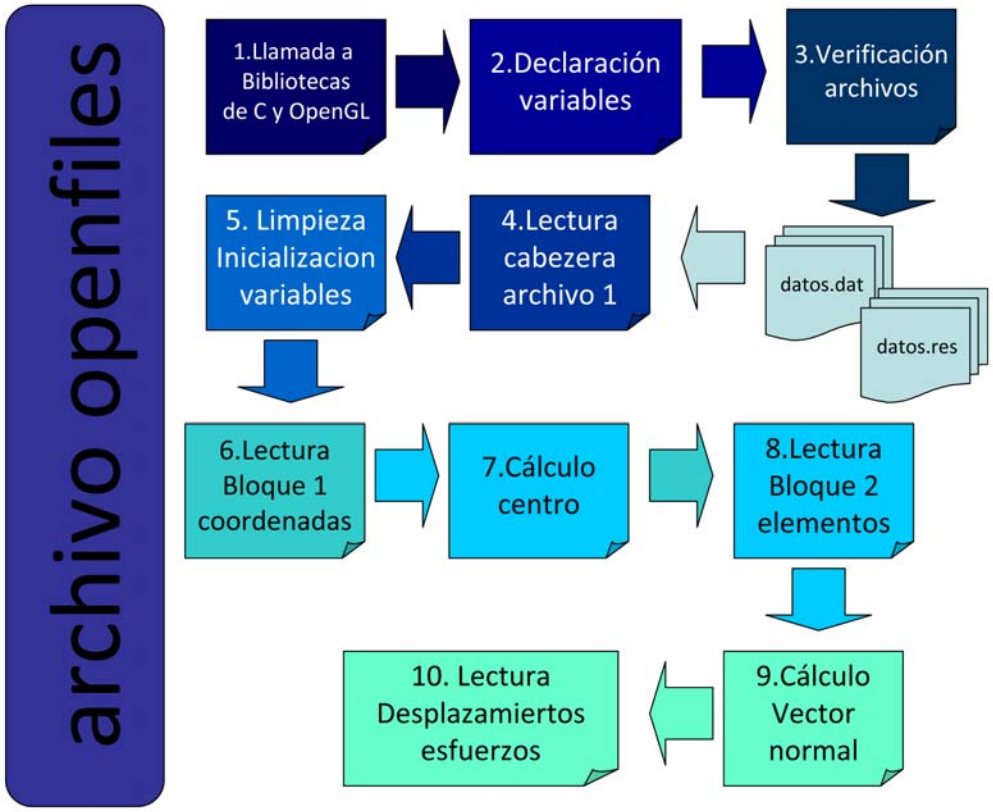


Fig 9.- Diagrama de bloque del archivo *openfiles.cpp*

La estructura del primer archivo de datos es como sigue a continuación:

Línea 1	Tipo de problema, análisis lineal o no lineal, verificación de la malla
Línea 2	Nombre de la pieza
Línea 3	Tipo de problema
Línea 4	Número de Dimensiones, grados de libertad, numero de propiedades,
Línea 5	Número de nodos, número de elementos, número de nodos cargados número de nodos restringidos, numero de materiales, condición de esfuerzo deformación, orden de integración, escala, número de nodos por elemento.
Línea 6	Número del material, modulo de young, modulo de poisson, variable falsa.
BLOQUE 1	Numero del nodo y sus coordenadas en x, y (z si es en tres dimensiones)
BLOQUE 2	Número del elemento y su conectividad de nodos
BLOQUE 3	Número del nodo, carga en x, carga en y, (carga en z si es en tres dimensiones)
BLOQUE 4	Número del nodo, restricción en x, restricción en y, (restricción en z si es en tres dimensiones)

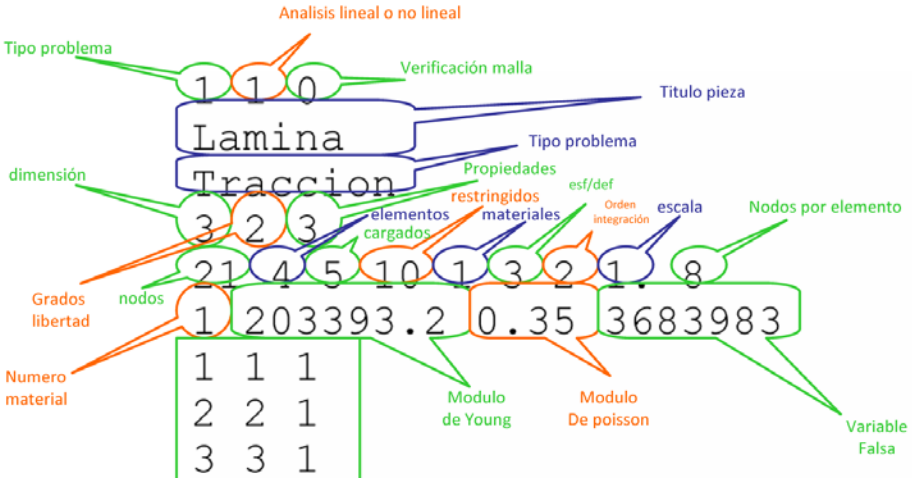


Fig 10.- Imagen de los datos de cabecera del archivo *lamina.dat* (archivo 1 de datos)

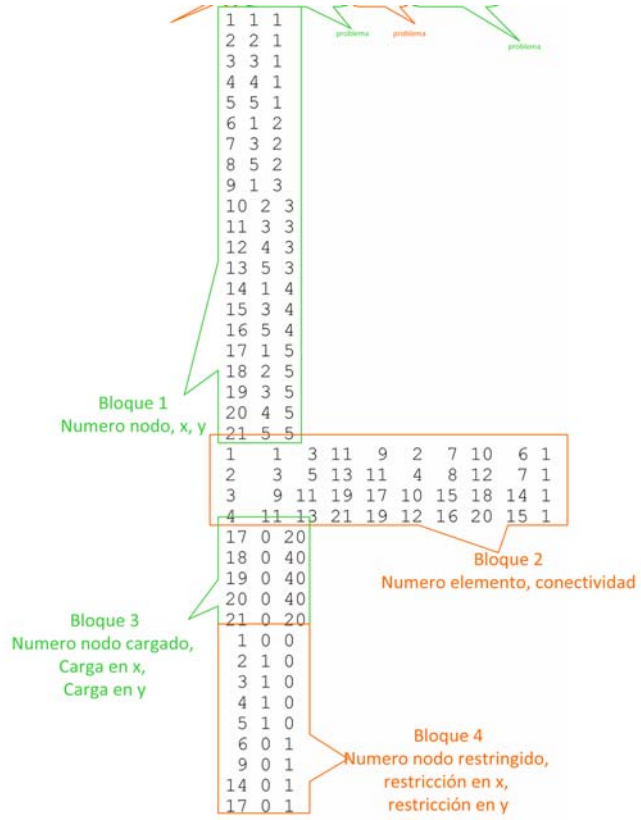


Fig 11.- Imagen de los bloques de datos del archivo *lamina.dat* (archivo 1 de datos)

ARCHIVO funciones.cpp

Este archivo contiene las funciones para calcular el vector normal a una superficie en un nodo, también contiene la función para cambiar la proyección entre perspectiva y paralela, adicionalmente tiene la función *eventomenu* la cual contiene todas las opciones con las cuales el usuario puede interactuar con el programa, estas son:

- Ver mallado
- Ver puntos
- Ver relleno
- Cambio de paleta de colores
- Color de fondo
- Ver deformación
- Mostrar malla original o triangulada
- Mostrar planos
- Activar o desactivar luz
- Reiniciar visualización
- Vista isométrica o perspectiva
- Salir

Finalmente contiene la función ayuda la cual presenta en consola en forma de texto el menú de ayuda con las opciones del programa.

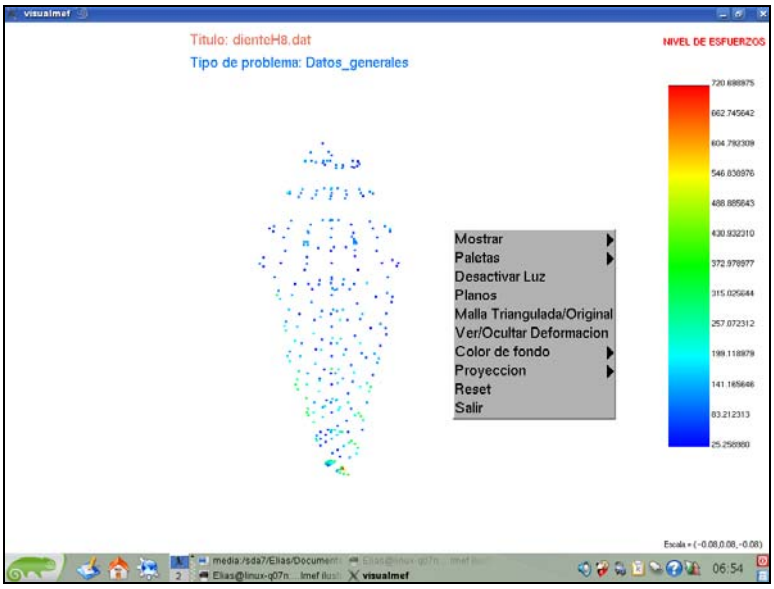


Fig 12.- Imagen de diente en formatos de nube de nodos del programa *VisualMEF*

ARCHIVO dibuja2d3d.cpp

Este archivo posee la función *dibuja2D* la cual se encarga de dibujar en la pantalla principal las piezas de dos dimensiones. De manera general los pasos que sigue esta función son los siguientes:

a) Llamada a bibliotecas de funciones propias del lenguaje de programación, y archivos de cabecera desarrollados para este programa.
b) Declaración de variables locales.
c) Cargado en memoria de los nodos y los desplazamientos de los mismos.
d) Calculo del factor de interpolación de color según el nivel de esfuerzo para cada nodo.

e) Asignación del color correspondiente a cada nodo según su factor de interpolación.
f) Dibujado de los puntos en la ventana principal según las coordenadas y el desplazamiento de cada nodo.
g) Asignación del vector normal a cada nodo.

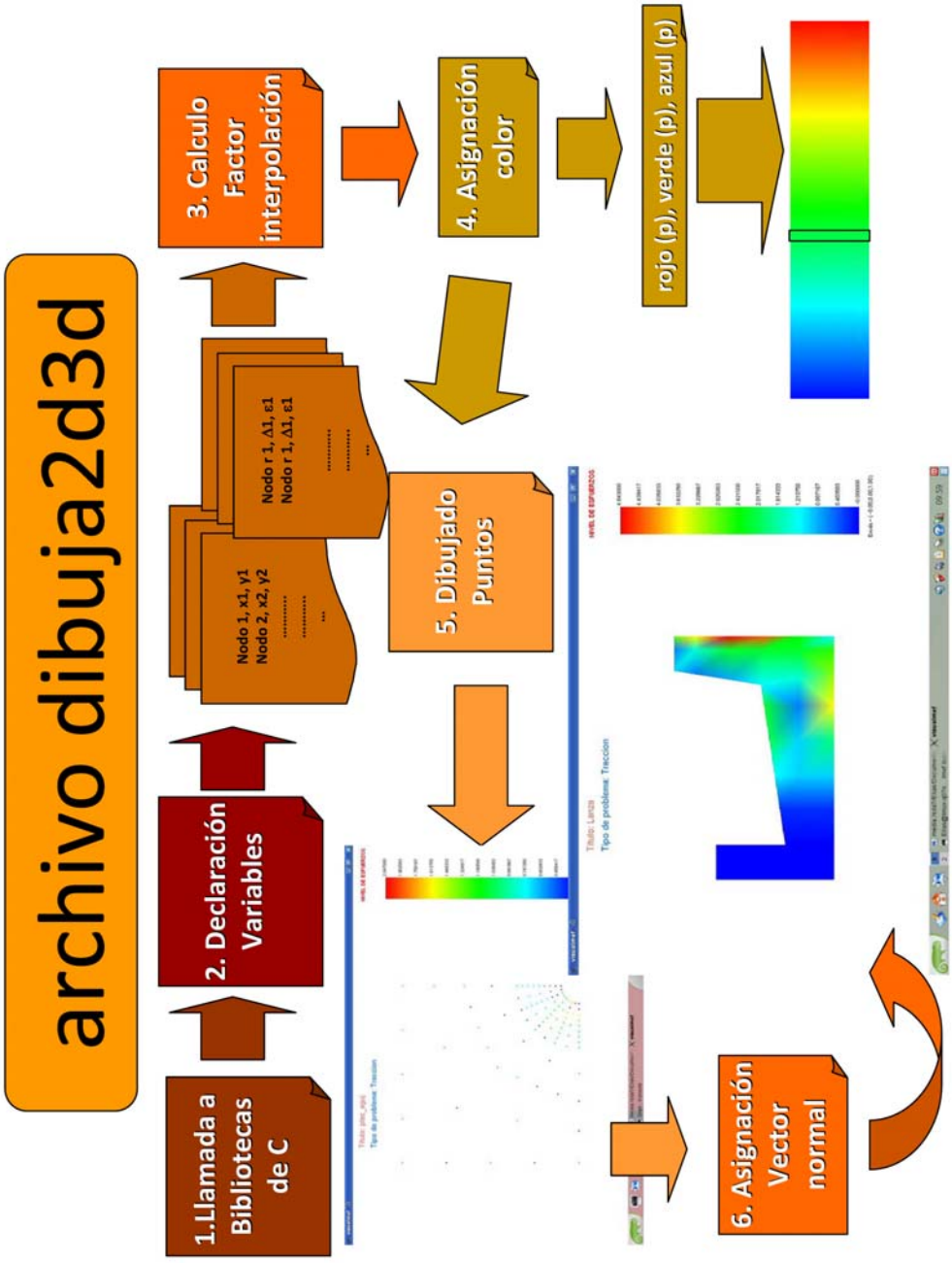


Fig 13.- Diagrama de bloque del archivo *dibuja2d3d.cpp*

ARCHIVO draw3d.cpp

Este archivo posee la función *draw3D* la cual se encarga de dibujar en la ventana principal las piezas de tres dimensiones. De manera general los pasos que sigue esta función son los siguientes:

a) Llamada a bibliotecas de funciones propias del lenguaje de programación, y archivos de cabecera desarrollados para este programa.
b) Declaración de variables locales.
c) Cargado en memoria de los nodos y los desplazamientos de los mismos.
d) Cálculo del factor de interpolación de color según el nivel de esfuerzo para cada nodo.
e) Asignación del color correspondiente a cada nodo según su factor de interpolación.
f) Dibujado de los puntos en la ventana principal según las coordenadas y el desplazamiento de cada nodo.
g) Asignación del vector normal a cada nodo.

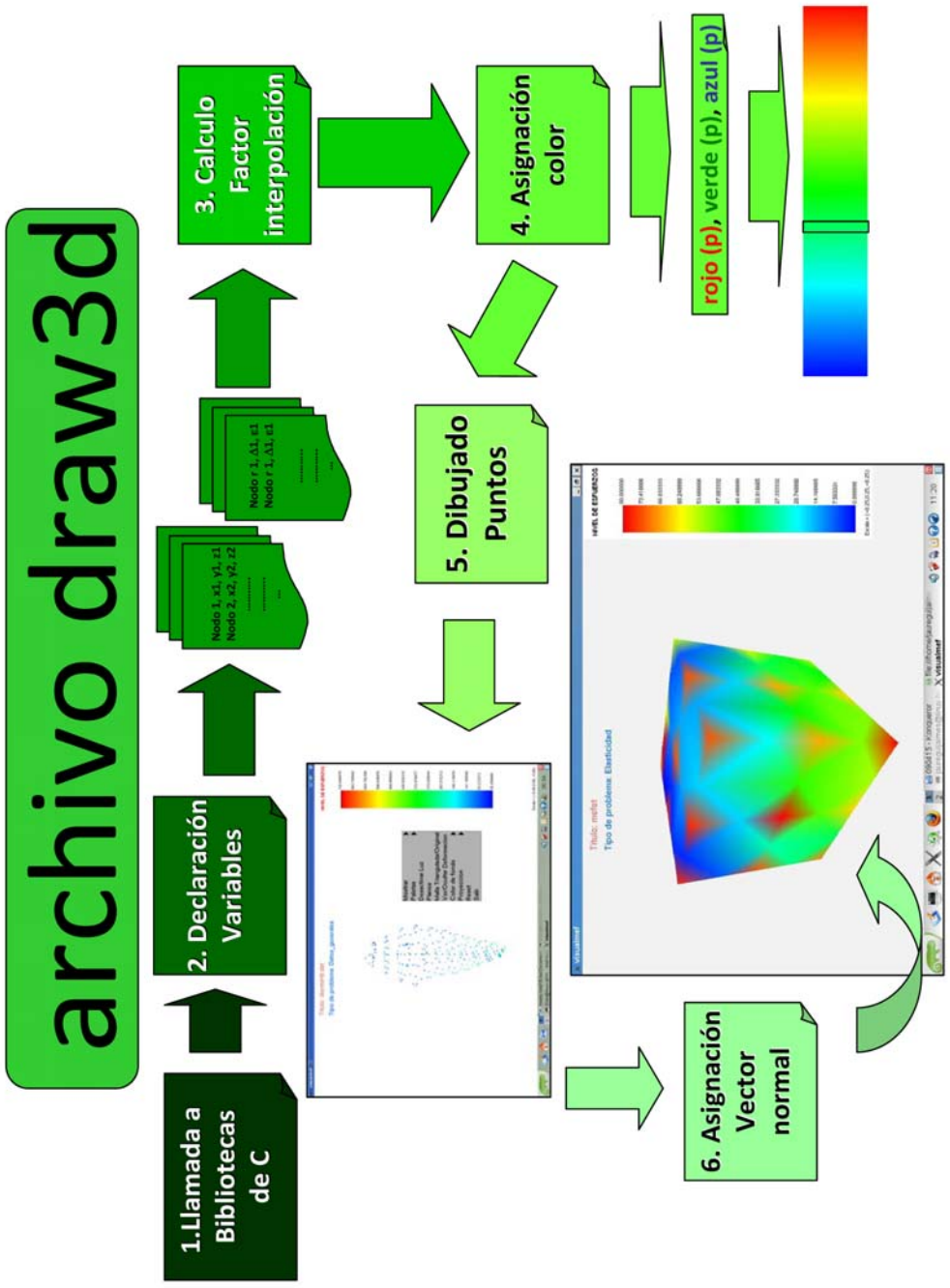


Fig 14.- Diagrama de bloque del archivo *drawd3d.cpp*

ARCHIVO raton.cpp

Este archivo contiene la función *Mouse*, la cual se encarga de dar la posibilidad de poder hacer girar la pieza dentro de la ventana principal con la finalidad de tener distintos puntos de vista de la pieza. También despliega un menú dinámico dentro de la pantalla principal con distintas opciones.

ARCHIVO teclado.cpp

Este archivo contiene la función *keyboard*, la cual se encarga de controlar las funciones que se despliegan al momento de presionar ciertas teclas en el teclado, estas funciones son:

1. Desplegar el menú
2. Ver la pieza en formato de relleno
3. Ver la pieza en formato de puntos
4. Ver la pieza en forma de mallado
5. Activar o desactivar la luz (iluminación de la pieza)
6. Desplazar verticalmente u horizontalmente la pieza en la pantalla principal.
7. Cambiar la paleta de colores.

ARCHIVO estructuras.h

Este archivo contiene todas las estructuras construidas para el manejo de datos dentro del programa, allí se describen que tipo de variables son cada una de estas estructuras.

ARCHIVO define.h

Este archivo contiene gran parte de las variables que durante la corrida del programa obtienen un valor constante en el proceso.

ARCHIVO paletas.h

Este archivo contiene todas las paletas de colores usadas en la visualización de las piezas, estas se muestran en un arreglo de 3x25 cada una, basadas en el formato de colores RGB, donde el primer valor corresponde al rojo, el segundo al verde y el tercero al azul.

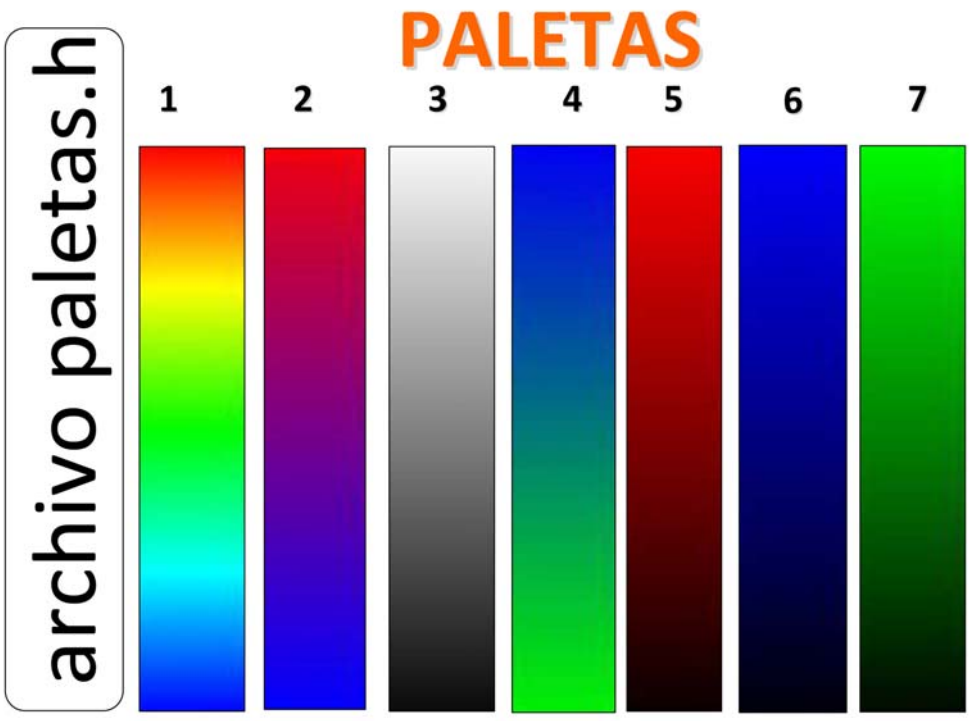


Fig 15.- Imagen de las 7 paletas que se usan en el programa *VisualMEF*.

ARCHIVO protofuncs.h

Este archivo contiene una lista de todas las funciones que se usan dentro del programa. Tiene como función mantener en memoria estas funciones.

ARCHIVO variab_extern.h

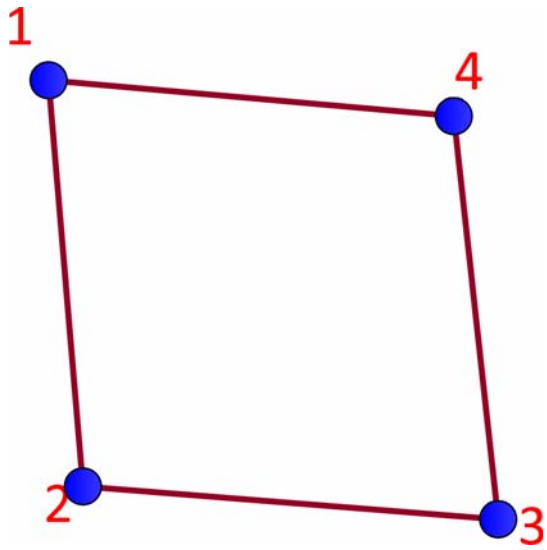
En este archivo están listadas todas las variables que se usan en distintos archivos sin que estas sean internas a cada función, estas son variables comunes entre todas las funciones, de allí la condición de variable externa.

ARCHIVO datos.dat y datos.res (la palabra "datos" es general y se substituye según sea el nombre del archivo de datos numero 1 y 2 de la pieza a estudiar)

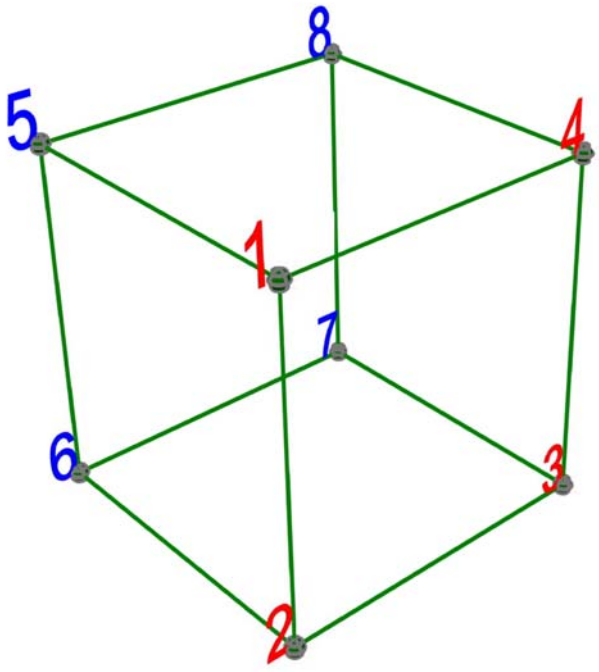
En el primer archivo cuya extensión es *.dat* como se explicó anteriormente están todos los datos que definen la geometría de la pieza, mientras que en el segundo archivo cuya extensión es *.res* están todos los desplazamientos de los nodos y los esfuerzos en los mismos.

EXPANSIÓN DEL PROGRAMA

El visualizador como se dijo anteriormente solo trabajaba con dos tipos de elementos, para el caso bidimensional solamente utilizaba el cuadrilátero de cuatro (4) nodos y en el caso tridimensional el hexaedro de ocho (8) nodos



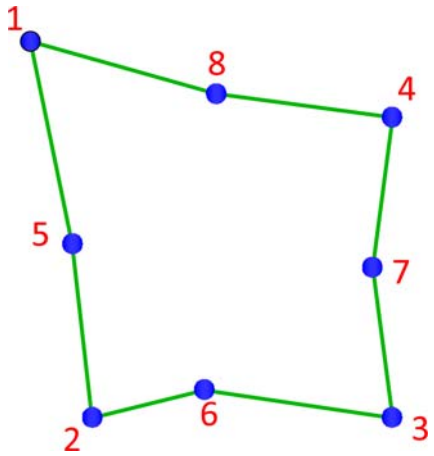
Cuadrilátero de 4 nodos



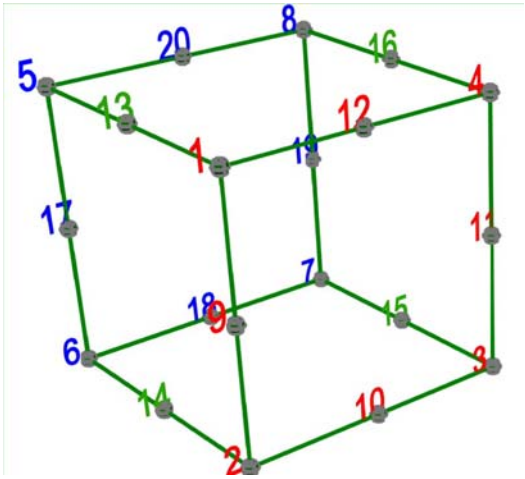
Hexaedro de 8 nodos

Fig 16.- Tipos de elementos manejados por el visualizador antes de la expansión.

La expansión del visualizador consistió en una primera etapa en donde se implemento en este otros tipos de elementos, los cuales si maneja el simulador encargado de suministrarle los dos archivos de datos necesarios para que el visualizador logre generar una imagen de una pieza en análisis. Los tipos de elementos implementados son el cuadrilátero de ocho (8) nodos y el hexaedro de veinte (20) nodos.



Cuadrilátero de 8 nodos



Hexaedro de 20 nodos

Fig 17 .- Tipos de elementos agregados al visualizador.

La segunda etapa de la expansión consistió en triangular cada uno de los tipos de elementos que maneja el visualizador. Recordando que hasta este punto los tipos de elementos que maneja el visualizador son:

- Cuadrilátero de 4 nodos.
- Cuadrilátero de 8 nodos
- Hexaedro de 8 nodos
- Hexaedro de 20 nodos.

Las razones principales por las cuales se decidió triangular cada uno de los tipos de elementos que maneja el visualizador son las siguientes:

- El triángulo es la figura geométrica más simple.
- El triángulo siempre se define en un plano.
- Según la teoría de Delanuay, cualquier figura geométrica se puede descomponer en triángulos.
- La normal a la superficie generada por un triángulo es constante en toda esta debido a que el triángulo se define en un plano.
- Al utilizar los tipos de elementos que maneja el visualizador y transformarlos en elementos triangulares más pequeños se aumenta el número de elementos con los cuales se dibuja la pieza y por consiguiente se debe obtener una mejor visualización de la figura.

Es importante mencionar que la triangulación de los elementos respeta la malla original por lo que no se generan nuevos nodos. Para poder triangular cada uno de los elementos manejados por el visualizador fue necesario crear un algoritmo que permitiera generar un código con el cual este programa pueda

realizar la triangulación. Esto se realizó para cada uno de los cuatro tipos de elementos que maneja el visualizador. A continuación se presentan cada uno de los algoritmos creados.

ALGORTIMO DE TRIANGULACIÓN DE ELEMENTO CUADRILATERO DE 4 NODOS (C4)

1. Lectura del numero elemento y de los cuatro nodos que lo componen
2. Calculo de la diagonal menor del cuadrilátero
3. Generación de los dos triángulos según sea la diagonal menor cumpliendo con la condición de Delanuay.

Fig 18.- Diagrama del algoritmo de triangulación del elemento C4

ALGORTIMO DE TRIANGULACIÓN DEL ELEMENTO TIPO HEXAEDRO DE 8 NODOS (H8)

1. Lectura del elemento y de los ocho nodos que lo componen
2. Determinación de los nodos que componen las caras del hexaedro.
3. Determinación de la diagonal menor de cada cara
4. Generación de los dos triángulos según sea la diagonal menor por cada cara del hexaedro y cumpliendo con la condición de Delanuay

Es importante destacar que el algoritmo aplicado al H8 es una aplicación del algoritmo del caso C4 ejecutado en cada una de las caras del H8, por lo tanto se deduce que el algoritmo C4 se aplica seis veces por cada hexaedro ya que un hexaedro posee seis caras y como resultado de este algoritmo se obtienen doce triángulos por cada H8.

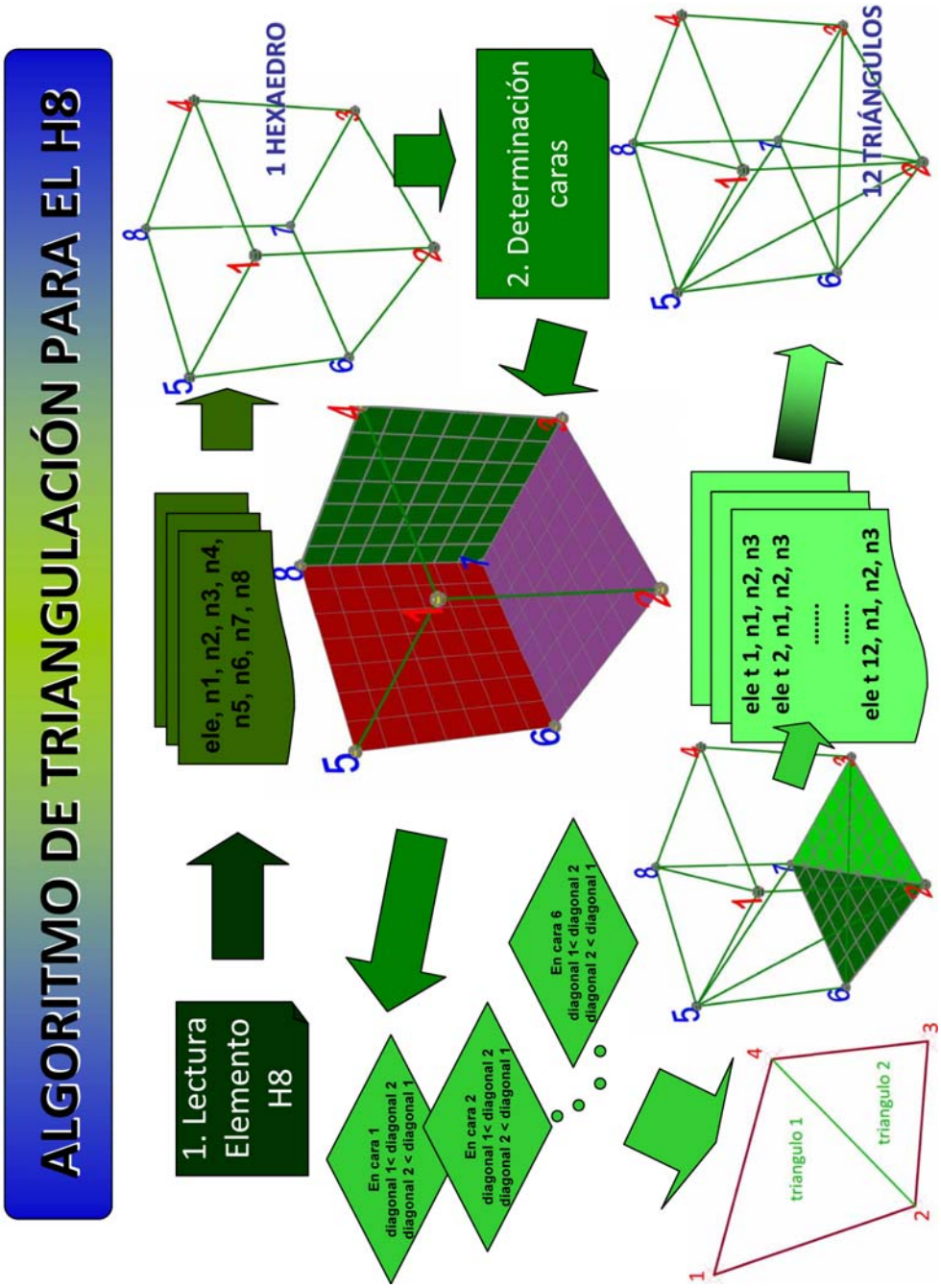


Fig 19.- Diagrama del algoritmo de triangulación del H8

ALGORTIMO DE TRIANGULACIÓN DE ELEMENTO CUADRILATERO DE 8 NODOS (C8)

1. Lectura del número de elemento y de los ocho nodos que lo componen
2. Generación de los primeros cuatro triángulos correspondientes a las esquinas del C8.
3. Calculo de la diagonal menor del C4 formado por los nodos intermedios del C8.
4. Generación de los dos triángulos según sea la diagonal menor del C4 de nodos intermedios, cumpliendo con la condición de Delanuay.

En este algoritmo nuevamente se usa el de triangulación del C4 dentro del C8. Se deduce entonces que al final de aplicar esta lista de pasos, se obtendrán 6 triángulos por cada C8.

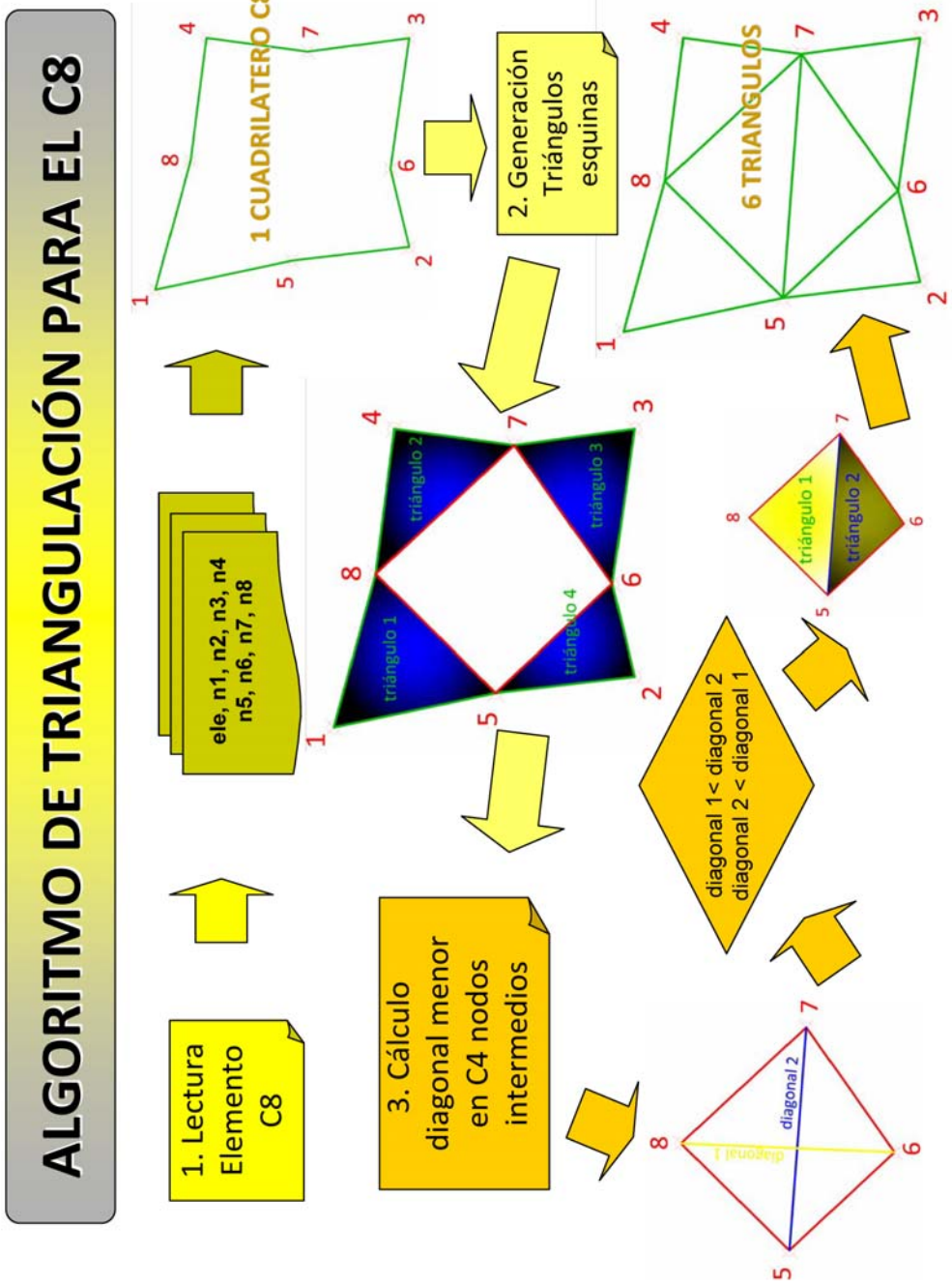


Fig 20.- Diagrama del algoritmo de triangulación para el C8

ALGORTIMO DE TRIANGULACIÓN DEL ELEMENTO TIPO HEXAEDRO DE 20 NODOS (H20)

1. Lectura del elemento y de los veinte nodos que lo componen
2. Determinación de los nodos que componen las caras del hexaedro.
3. Generación de los cuatro triángulos de las esquinas de cada cara.
4. Cálculo de la diagonal menor del C4 formado por los nodos intermedios de cada cara del H20.
5. Generación de los dos triángulos según sea la diagonal menor de cada C4 de los nodos intermedios por cada cara del H20 cumpliendo la condición de Delanuay.

Es importante destacar que el algoritmo aplicado al H20 es una aplicación del algoritmo del caso C8 ejecutado en cada una de las caras del H20, donde a su vez dentro del algoritmo del C8 esta aplicado el algoritmo del C4, por lo tanto se deduce que el algoritmo C4 se aplica seis veces por cada hexaedro ya que un hexaedro posee seis caras, también se deriva que por cada cara entonces se generan 4 triángulos en las esquinas y 2 en el C4 de los nodos intermedios, es decir que por cada cara del H20 de generan un total de 6 triángulos, es decir, por cada H20 se generan 36 triángulos.

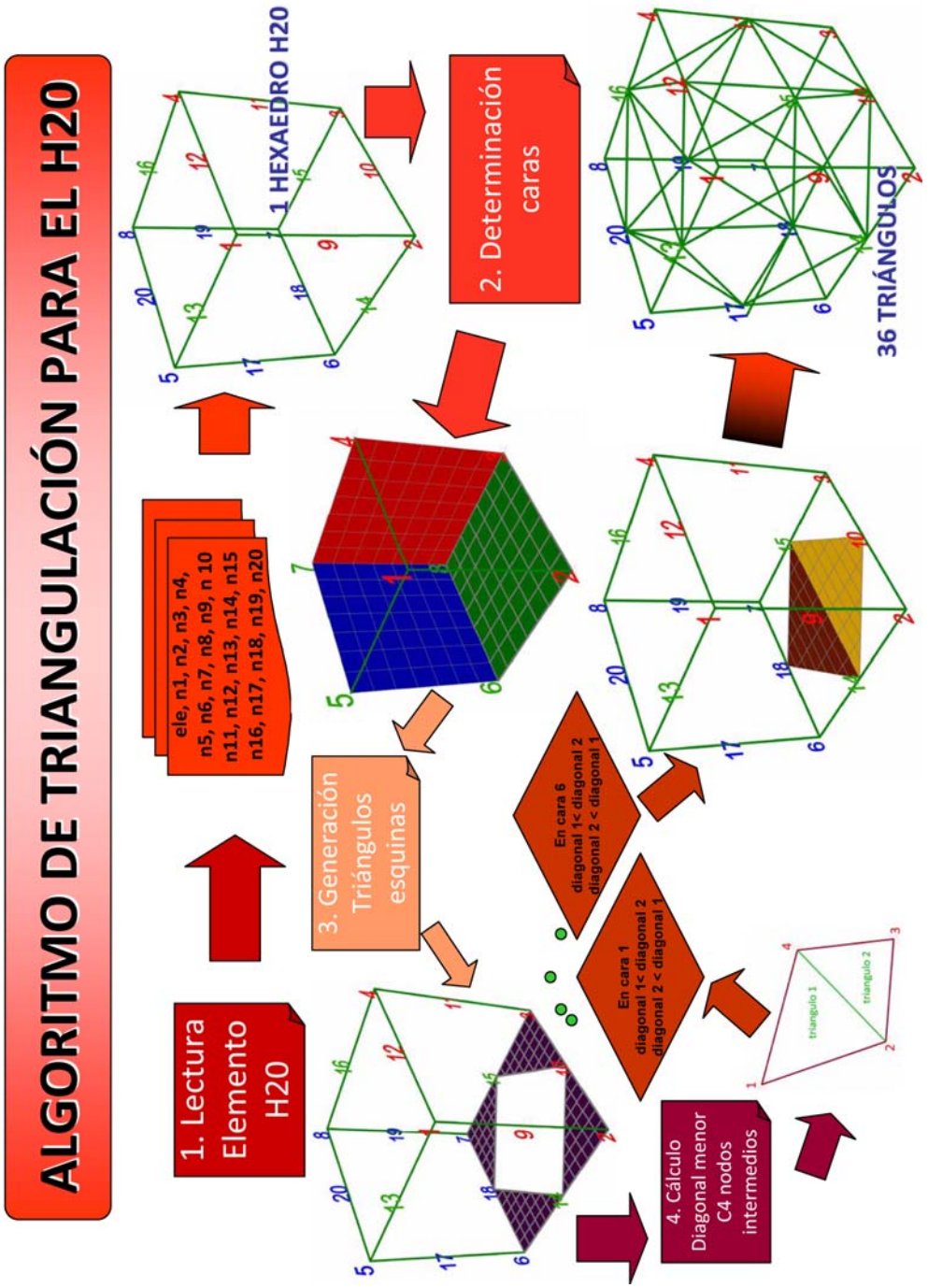


Fig 21.- Diagrama del algoritmo de triangulación del H20

MEJORAS REALIZADAS EN EL PROGRAMA

Una vez desarrollado los fundamentos de la expansión del programa de visualización, se debe entonces aplicar la misma al código original del programa. Para lograr esto es necesario hacer ciertas modificaciones en las rutinas y funciones del visualizador y agregar funciones adicionales necesarias para cumplir con la expansión necesaria.

Es importante mencionar que se depuraron varios de los archivos contenidos en el programa visualizador. Se agrego un único archivo que lleva por nombre *triangulación*.cpp el cual contiene las funciones necesarias para la triangulación de las mallas recibidas por el programa en el formato antes descrito.

Para llevar un orden secuencial se comentarán a continuación las modificaciones y rutinas agregadas al código en el orden en que se mostró al momento de explicar el funcionamiento general del visualizador antes de realizar la expansión.

ARCHIVO main.cpp

En primera instancia la llamada a las bibliotecas del c y la declaración de variables permanece sin alteraciones, de igual manera la llamada a la función *openfiles* permanece sin cambios. Paso siguiente a este llamado se introduce el llamado a una función nueva que lleva por nombre *triangulame_esto* la cual se explicara con más detalles mas adelante y que es la encargada de triangular los elementos recibidos por el visualizador.

La función *display* permanece sin alteraciones. La función *render* es modificada a través de la introducción de una etiqueta para el tipo de elemento que deba manejar. Esta etiqueta varía según la siguiente tabla:

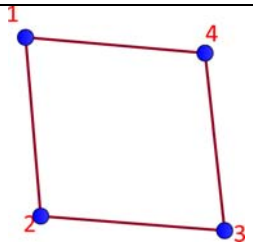
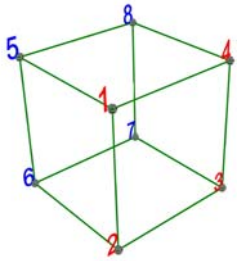
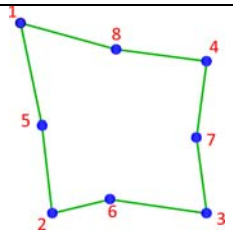
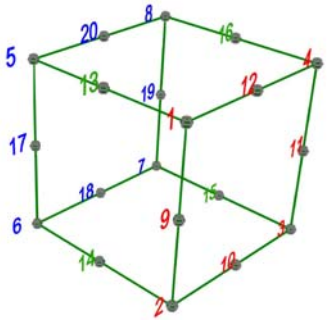
ETIQUETA	NOMBRE	IMAGEN
Cuadri4nod	Cuadrilátero de 4 nodos	
Hexa8nod	Hexaedro de 8 nodos	
Cuadri8nod	Cuadrilatero de 8 nodos	
Hexa20nod	Hexaedro de 20 nodos	

Fig 22.-Tabla de etiquetas implementadas en el visualizador *VisualMEF*

Según sea el tipo de elemento que reciba el dibujador entonces la función redireccionará la línea de ejecución a la función *dibuja2D* o *draw3D* para completar la rutina inicial del *main*, nótese que las etiquetas *Cuadri4nod* y *Cuadri8nod* son casos bidimensionales lo cuales se redireccionaran a la función *dibuja2D* y las etiquetas *Hexa8nod* y *Hexa20nod* son casos tridimensionales los cuales se redireccionaran a la función *draw3D*. Las funciones *reshape, init* y *displaysubwindow* permanecen sin cambio.

ARCHIVO openfiles.cpp

La función contenida dentro de este mismo archivo y que lleva el mismo nombre en principio no se altera, ya que la declaración de variables no se cambia y la lectura y almacenamiento de los primeros datos del archivo uno de datos tampoco sufre cambios. Aquí también se introduce la etiqueta antes descrita para cada tipo de elemento.

Según cada caso se almacena los datos en variables tipo estructuras especializadas para cada elemento, el cálculo de la normal en cada nodo permanece igual. De igual manera la lectura y almacenamiento de los esfuerzos y desplazamientos en los nodos desde el segundo archivo de datos también permanece sin cambio.

ARCHIVO dibuja2d3d.cpp

En la función *dibuja2D* ubicada en este archivo se configuro de tal manera que dibuje por defecto todos los elementos triangulados, haciendo uso

de las etiquetas desarrolladas para cada caso, el resto del código permanece sin alteraciones.

ARCHIVO draw3d.cpp

De igual manera para la función *draw3D* ubicada en este archivo se configuro para que por defecto se dibujen todos los elementos triangulados. Aquí también se usan las etiquetas para clasificar cada caso según el tipo de elemento. Es importante destacar que para los casos de C8 y H20 fue necesario la creación de nuevas estructuras para el manejo de los datos y el orden de los nodos en estos casos. El resto del código permanece sin alteraciones.

En los archivos *raton.cpp, teclado.cpp, estructuras.h,. defines.h, paletas.h, protofuncs.h, variab_extern.h,* no hubo mayores modificaciones salvo la declaración de algunas variables necesarias para la triangulación.

ARCHIVO triangulacion.cpp

Este es el archivo que posee todas las instrucciones necesarias para triangular todos los elementos que recibe el visualizador, en el encontraremos los siguientes procesos:

a) Llamada a bibliotecas de funciones propias del lenguaje de programación
b) Declaración de variables locales.

c) Definición de la función *triangulame_esto* encargada de triangular todos los tipos de elementos que reciba el visualizador haciendo uso de las etiquetas antes descritas.
d) Calculo del vector normal en cada uno de los nodos de los triángulos.
e) Definición de la función *diagonal*, encargada de calcular la menor diagonal de un C4 haciendo uso de la función *distanciapp*
f) Definición de la función *distanciapp*, encargada de medir la distancia entre dos puntos
g) Definición de la función *triangulac8*, encargada de triangular los tipos de elementos C8, esto lo hace generando los triángulos en las esquinas del C8, e invocando la función *diagonal* para calcular los dos triángulos que se generan en el C4 derivado de los 4 nodos intermedios del C8.

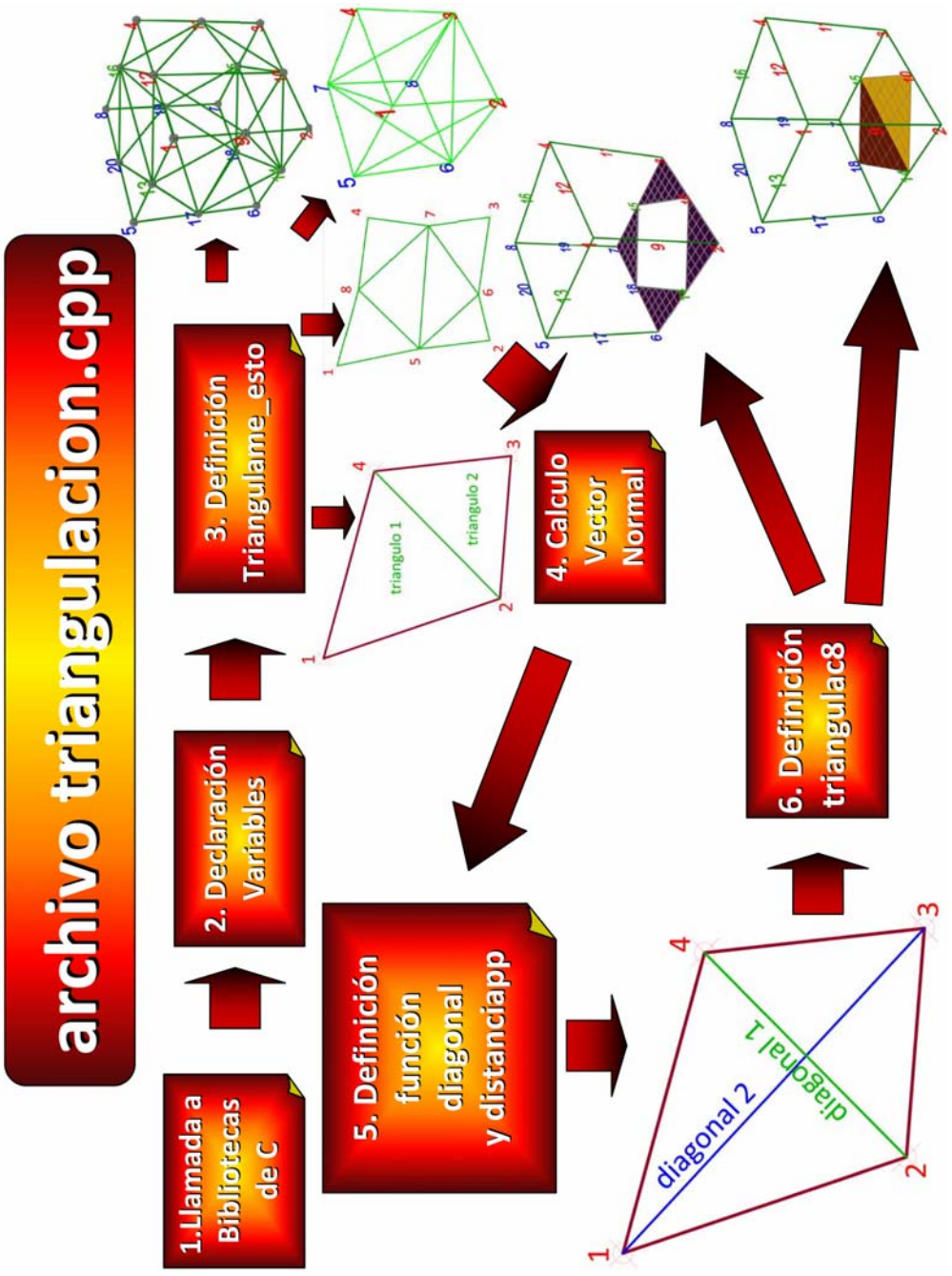


Fig 23.-Diagrama de bloque del archivo *triangulación.cpp*

CAPÍTULO V

CALIBRACIÓN Y ESTUDIO DE PIEZAS DESARROLLADAS

Es necesario verificar el desempeño del programa utilizando casos de calibración que permitan apreciar el mismo. Para esto utilizamos 4 piezas una por cada tipo de elemento, cuadriláteros de 4 y 8 nodos y hexaedros de 8 y 20 nodos.

En todos los casos de calibración se procedió a crear los valores de esfuerzo de cada archivo de resultados manualmente con el fin de mostrar toda la potencialidad del visualizador en lo que a color se refiere, sin tomar en cuenta las cargas del archivo de datos.

Para elementos cuadriláteros de 4 nodos tenemos el archivo *lamielas.dat*:

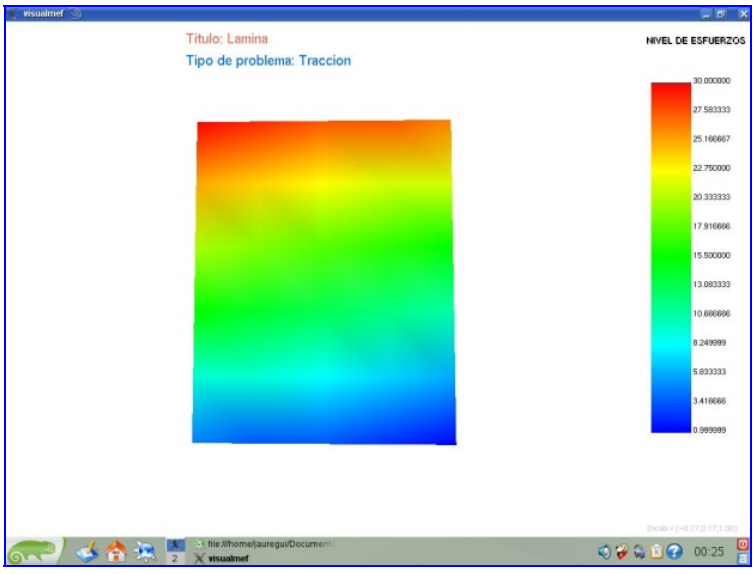


Fig. 24.- Ejercicio de calibración *lamielas* superficie triangulada.

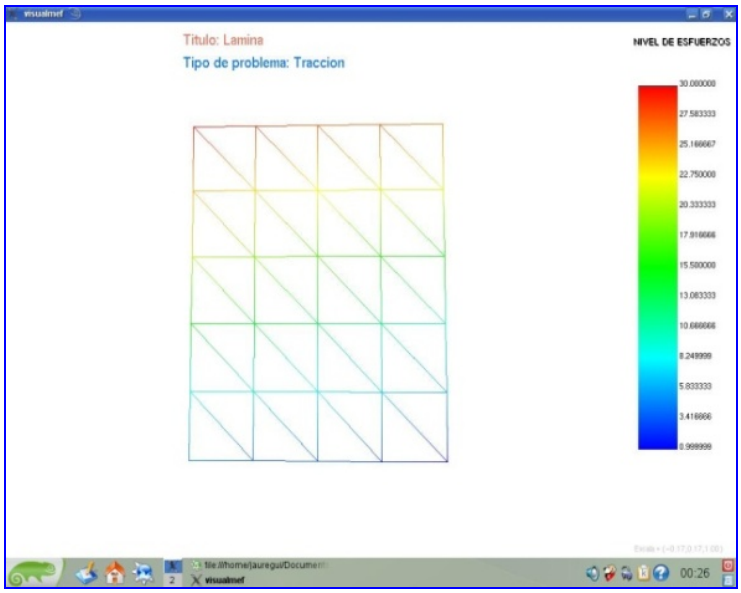


Fig. 25.- Ejercicio de calibración *lamielas* malla triangulada.

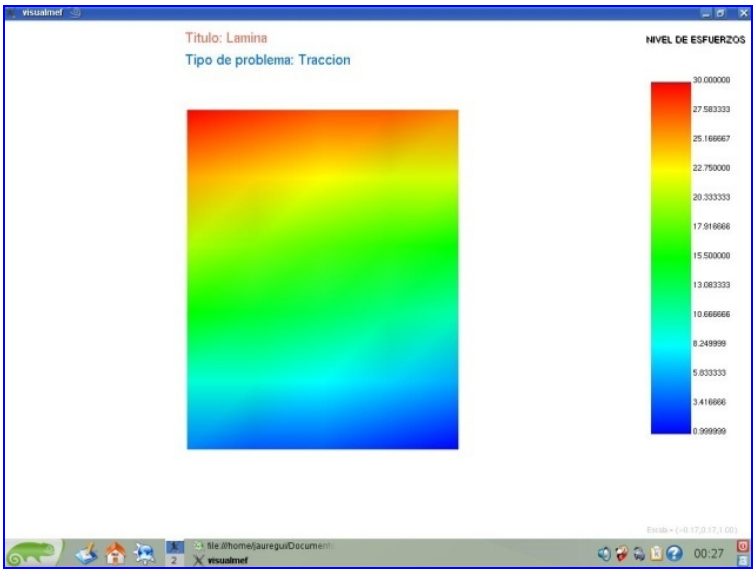


Fig. 26.- Ejercicio de calibración *lamielas* superficie original.

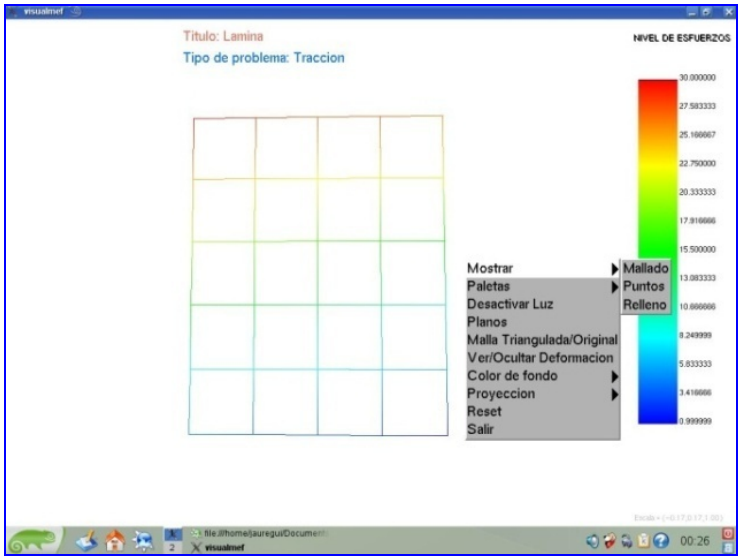


Fig. 27.- Ejercicio de calibración *lamielas* malla original.

El ejercicio de calibración para cuadriláteros de 8 nodos es el archivo c8:

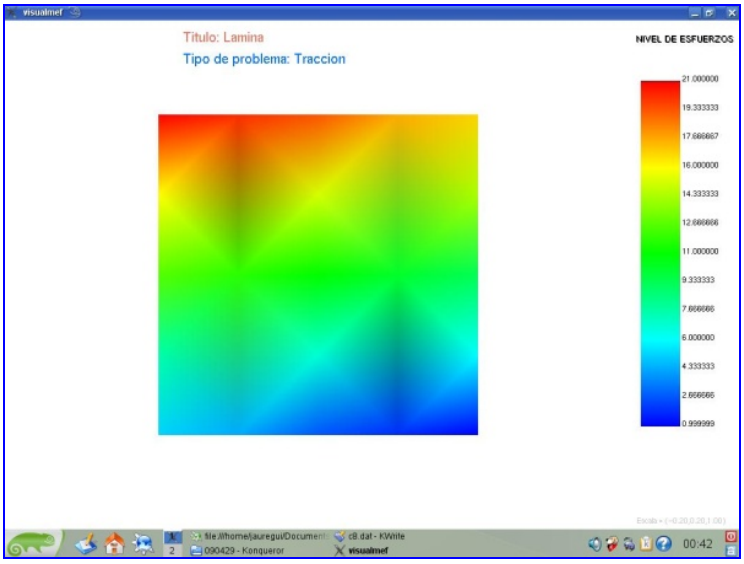


Fig. 28.- Ejercicio de calibración *c8* superficie triangulada.

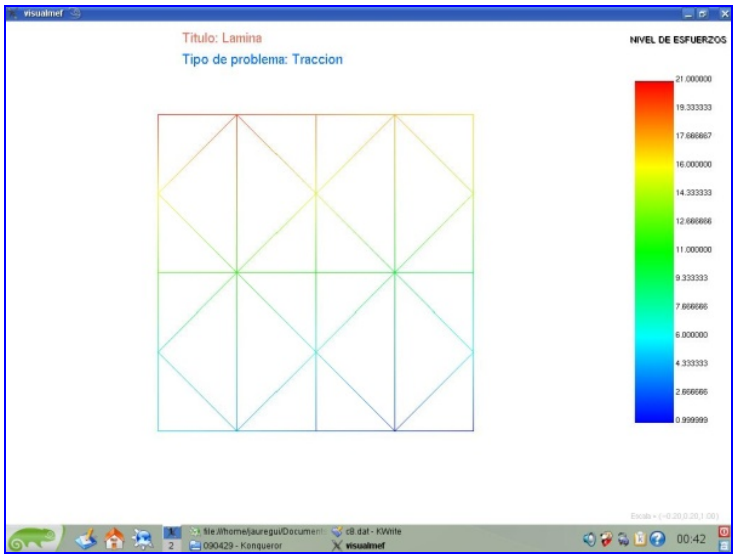


Fig. 29.- Ejercicio de calibración c8 malla triangulada.

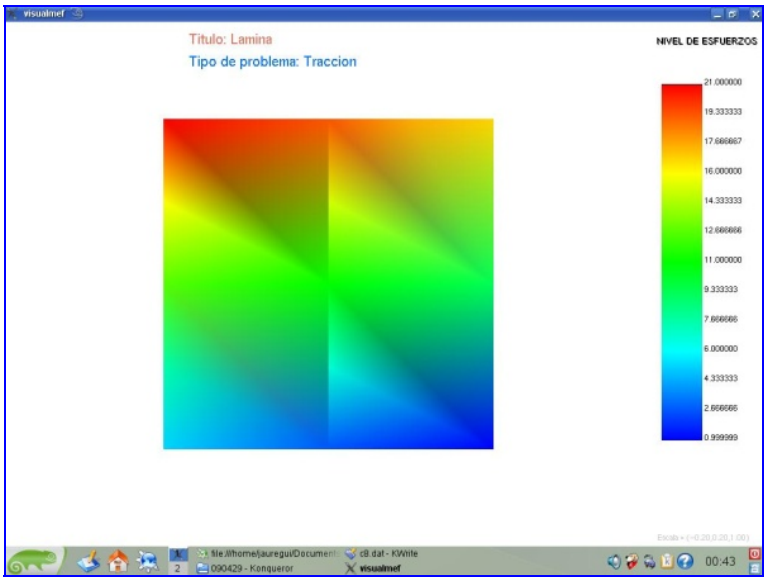


Fig. 30.- Ejercicio de calibración *c8* superficie original

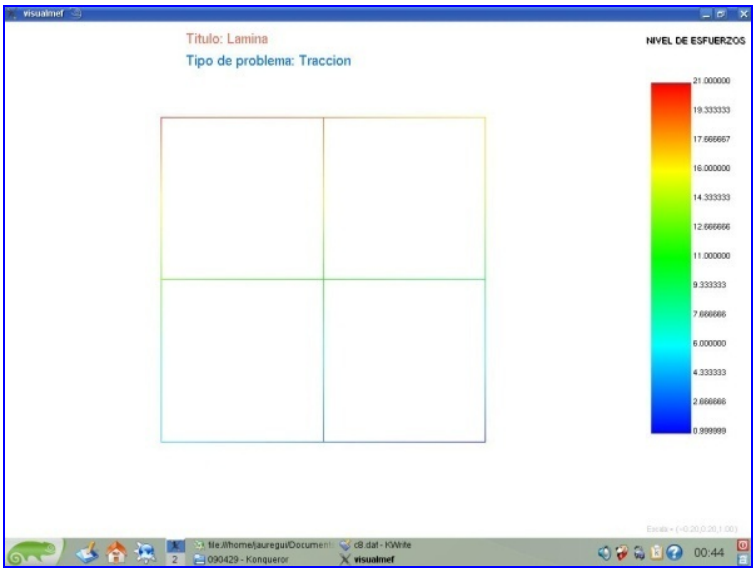


Fig. 31.- Ejercicio de calibración *c8* malla original.

El ejercicio de calibración para hexaedros de 8 nodos es el archivo *mefet*:

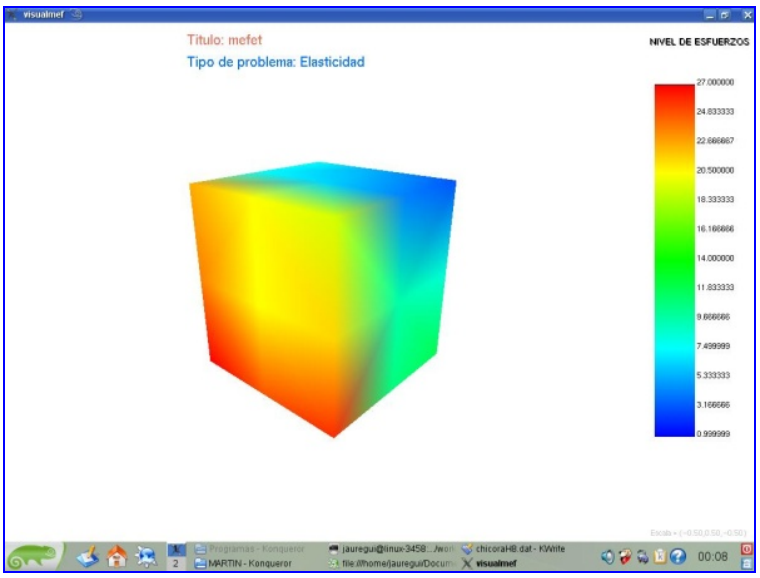


Fig. 32.- Ejercicio de calibración *mefet* superficie triangulada.

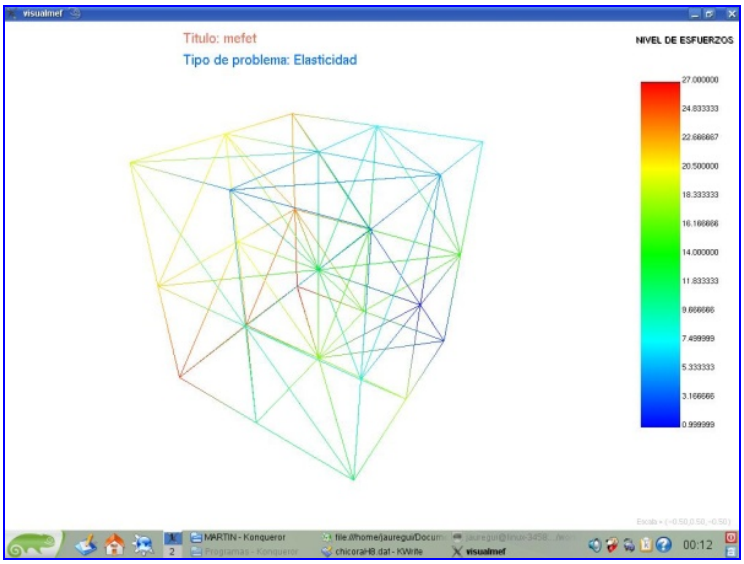


Fig. 33.- Ejercicio de calibración *mefet* malla triangulada.

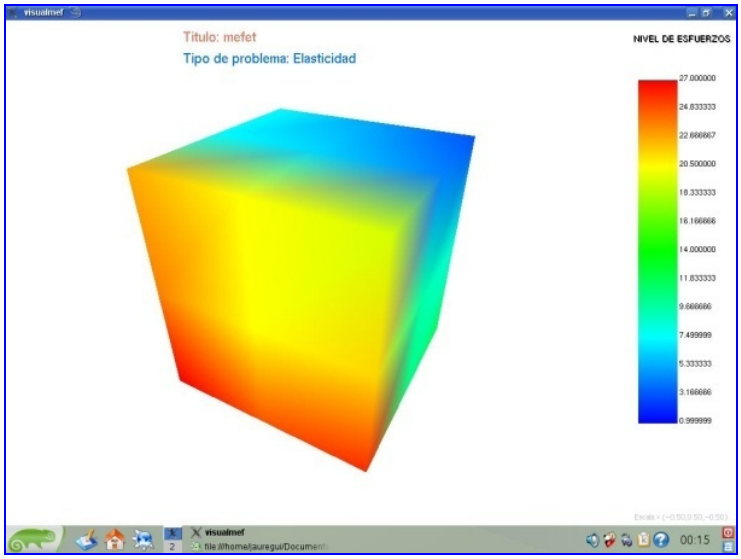


Fig. 34.- Ejercicio de calibración *mefet* superficie original.

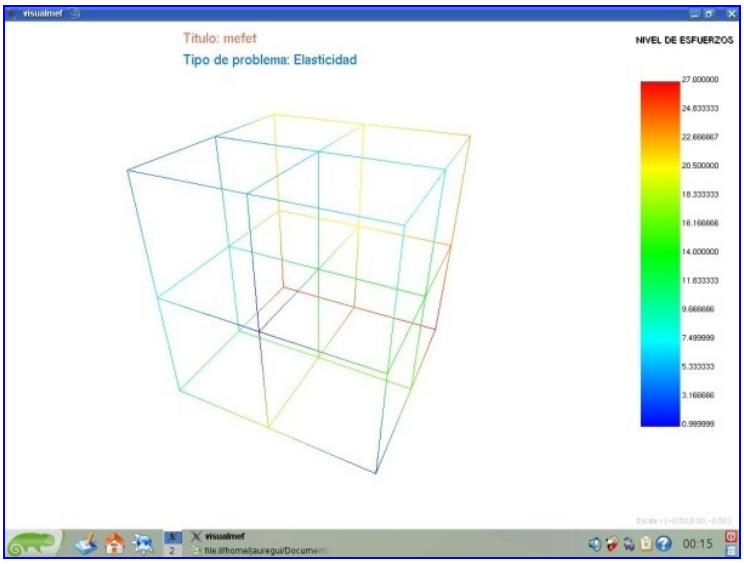


Fig. 35.- Ejercicio de calibración *mefet* malla original.

El ejercicio de calibración para hexaedros de 20 nodos es el archivo *mefet20*:

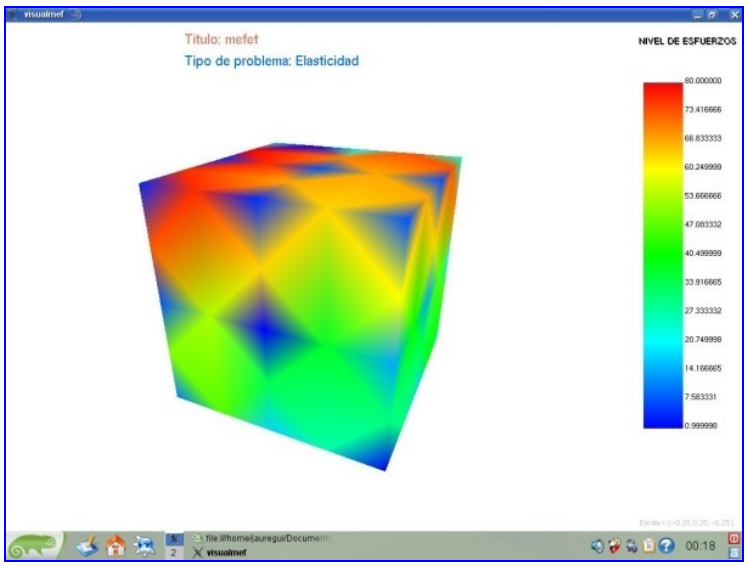


Fig. 36.- Ejercicio de calibración *mefet20* superficie triangulada.

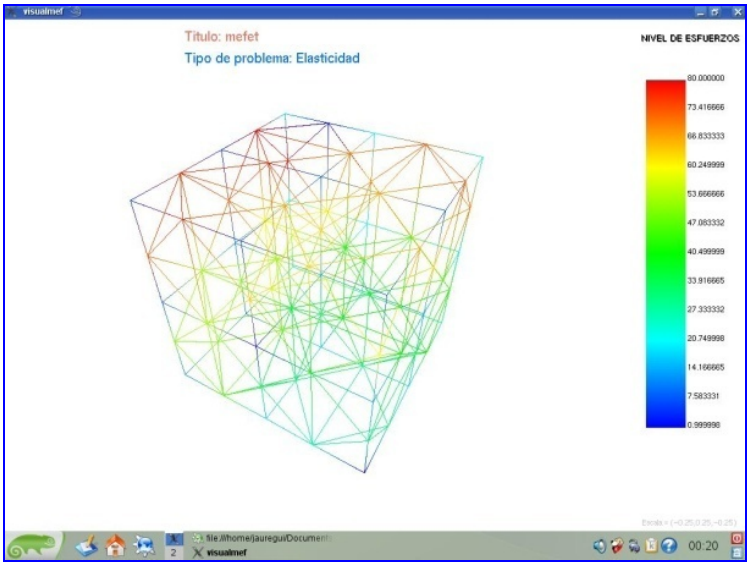


Fig. 37.- Ejercicio de calibración *mefet20* malla triangulada.

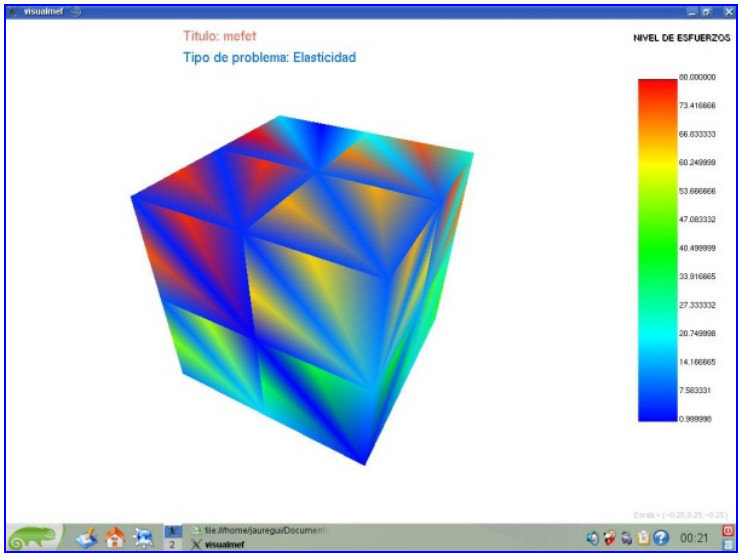


Fig. 38.- Ejercicio de calibración *mefet20* superficie original.

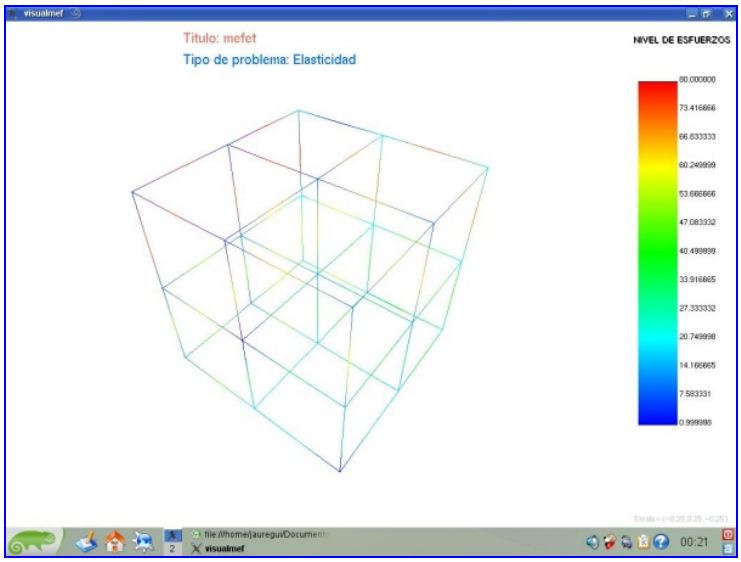


Fig. 39.- Ejercicio de calibración *mefet20* malla original.

Para este trabajo se aplicó el análisis numérico a piezas reales que se utilizaron durante el desarrollo del programa visualizador original [10], estas piezas son la lanza empujadora (cuadriláteros de 4 nodos) y la mesa para cilindros (hexaedros de 8 nodos), y se pudo visualizar el archivo *placa_aguj* (cuadriláteros de 8 nodos) con la finalidad de realizar comparaciones con el programa actual.

Descripción problema lanza: para esta pieza se realizó un análisis de esfuerzo a compresión en dos dimensiones cargado en dirección Y, la pieza fue restringida de moviendo en el borde externo desde el elemento 1 hasta el 8. [10].

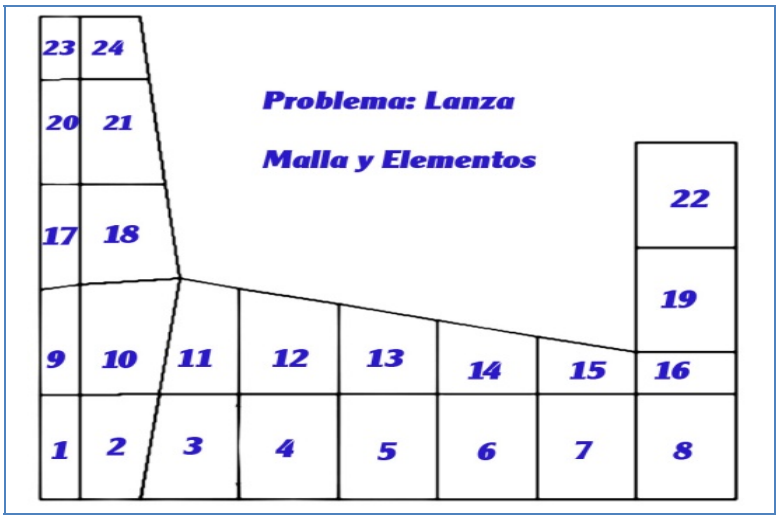


Fig. 40.- Malla de lanza.

Descripción problema mesa: para esta pieza se realizó un análisis de esfuerzo a compresión en tres dimensiones, la pieza fue restringida de moviendo en el borde superior. [10].

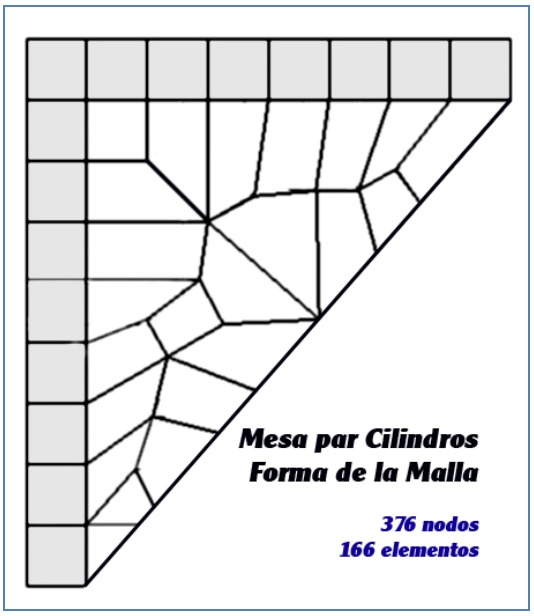
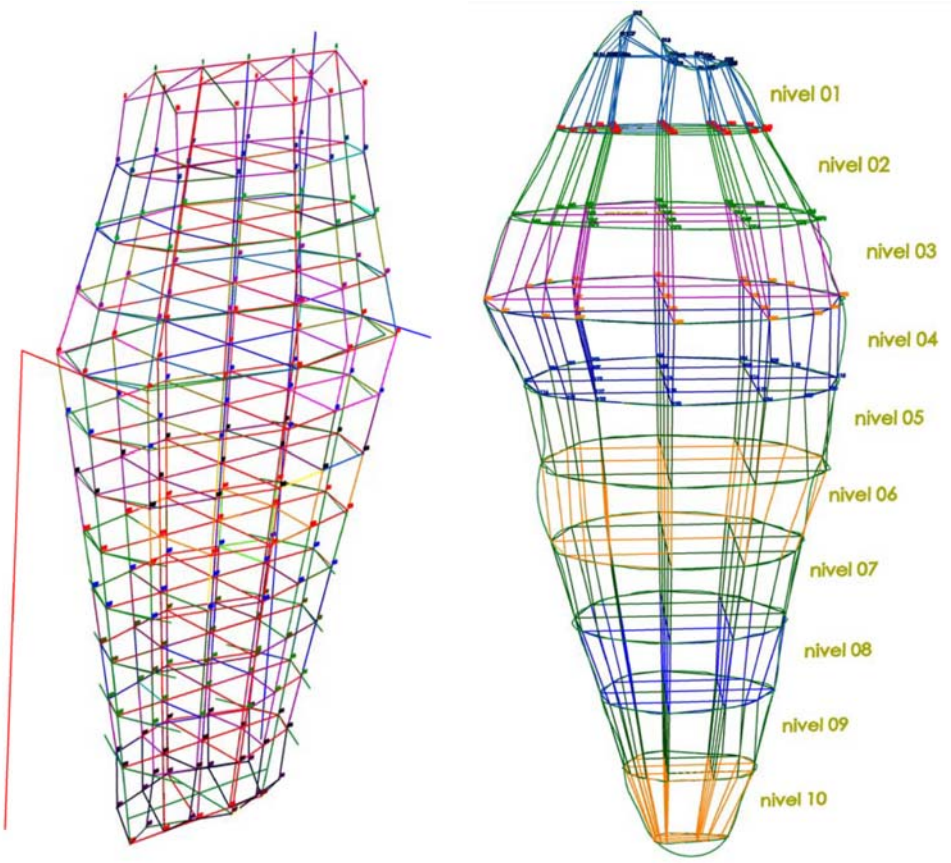


Fig. 41.- Malla de mesa para cilindros.

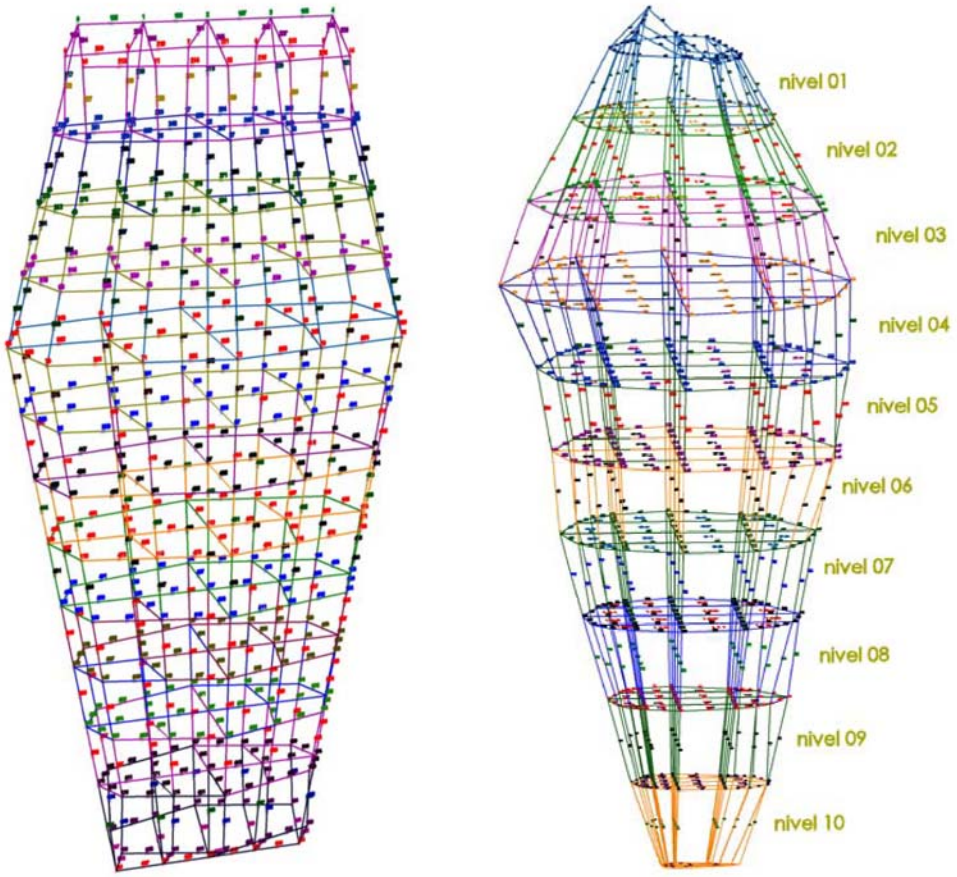
Además con el propósito de emplear el programa VisualMEF con los nuevos elementos se hizo manualmente el mallado de dos piezas, en hexaedros de ocho (8) nodos y de veinte (20) nodos, estas piezas son la chícora y el diente.

Descripción del problema Diente y Chícora: Para estas piezas se procedió a realizar dos tipos de análisis, uno a compresión y otro a flexión, en ambos casos se restringió el movimiento en la periferia de la base de ambas piezas y se colocaron cargas nodales axiales para el caso de estudio a compresión, y cargas laterales para el caso de flexión.



Chícora en formato H8 Diente en formato H8

Fig. 42.-Imagen de mallado formato H8 elaborado manualmente en Autocad



Chicora en formato H20 Diente en formato H20

Fig. 43.- Imagen de mallado formato H20 elaborado manualmente en Autocad

CAPÍTULO VI

VISUALIZACIÓN DE RESULTADOS

Los resultados obtenidos se pueden apreciar en las siguientes imágenes:

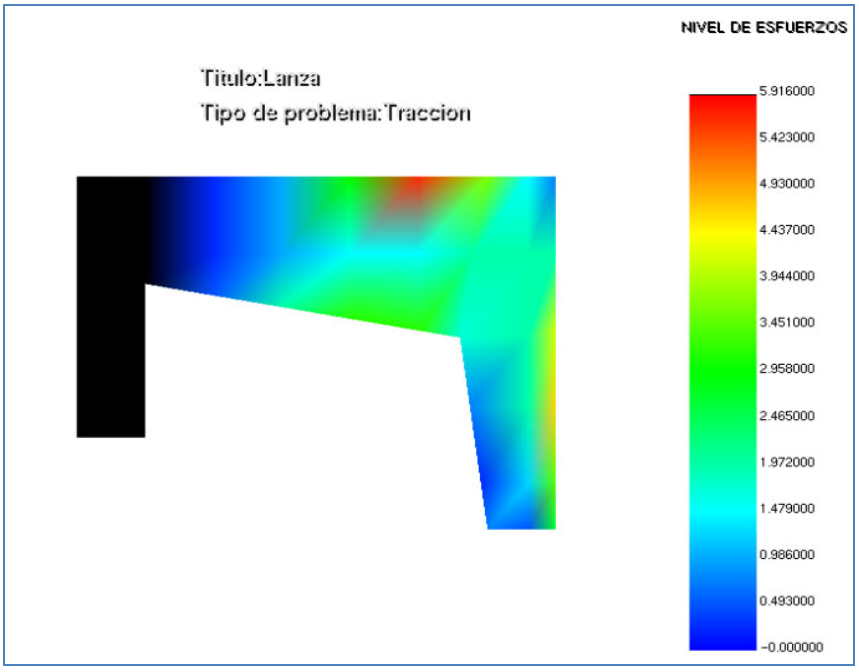


Fig. 44.- Resultados del problema *Lanza* (visualizador original)

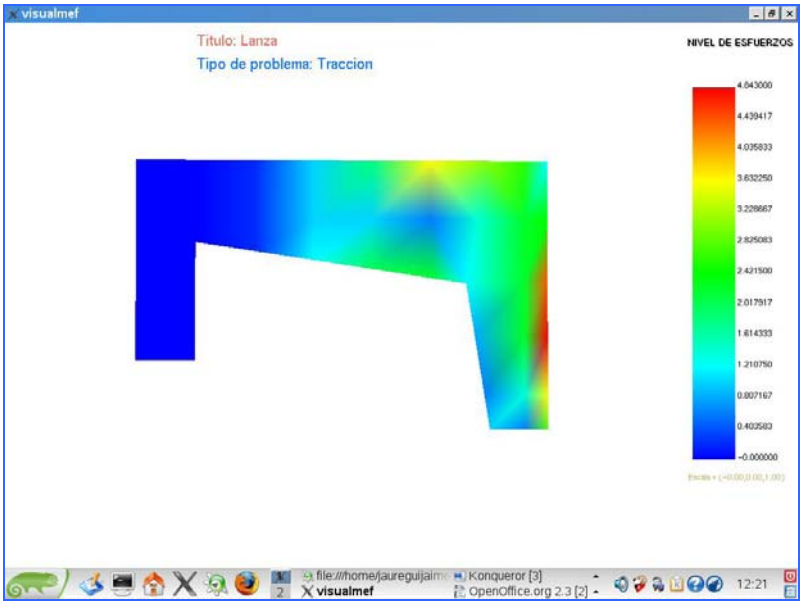


Fig. 45.- Resultados del problema *Lanza* (programa VisualMEF)

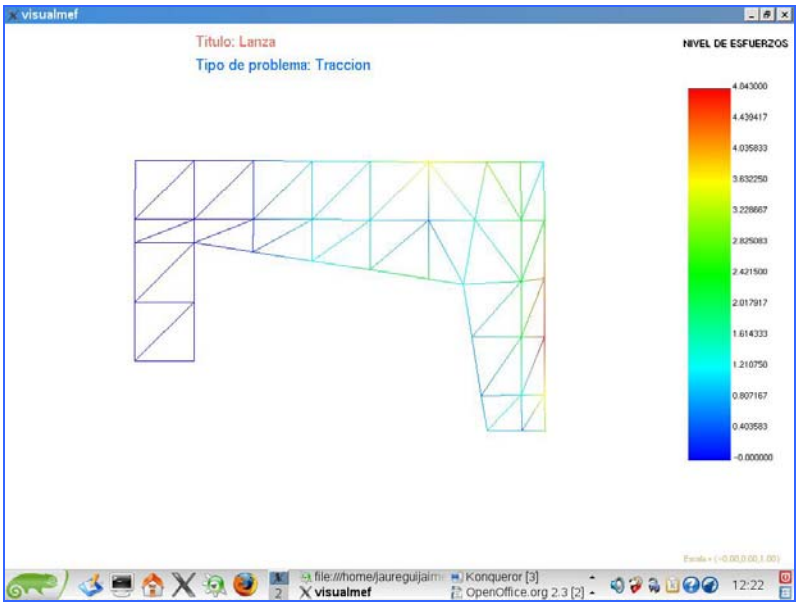


Fig. 46.- Malla triangulada del problema *Lanza* (programa VisualMEF)

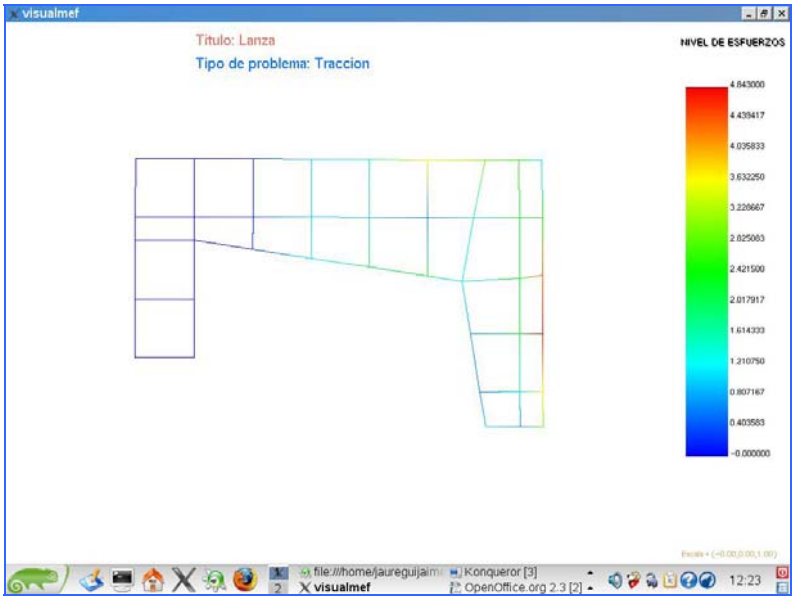


Fig. 47.- Malla original del problema *Lanza* (programa VisualMEF)

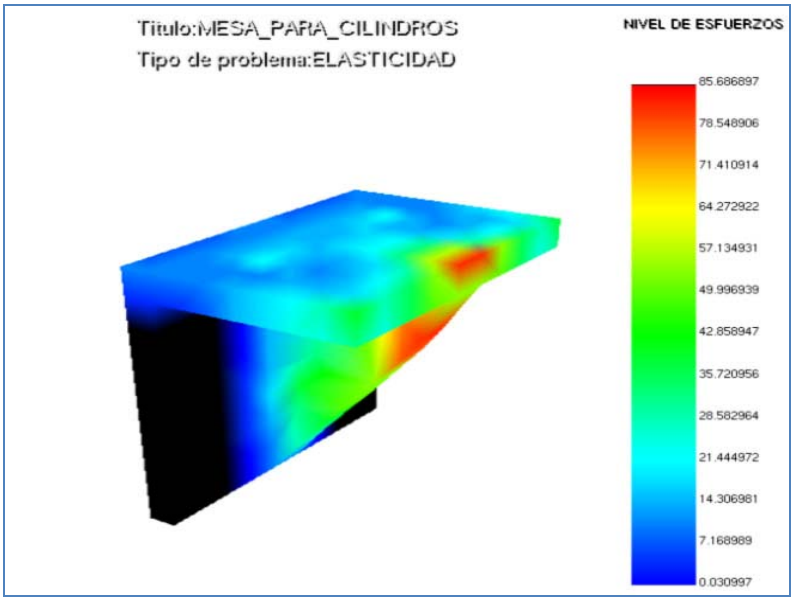


Fig. 48.- Resultados del problema *Mesa* (programa original)

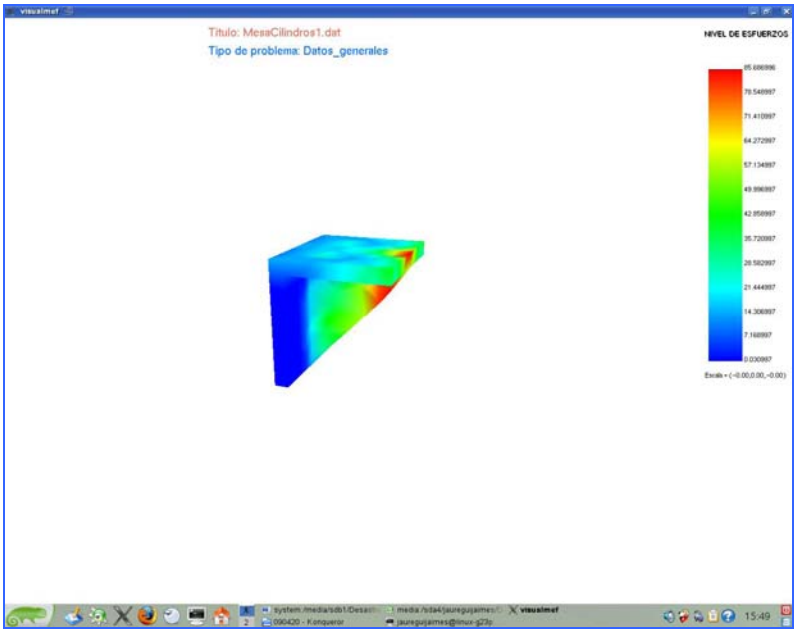


Fig. 49.- Resultados del problema *Mesa* (programa VisualMEF)

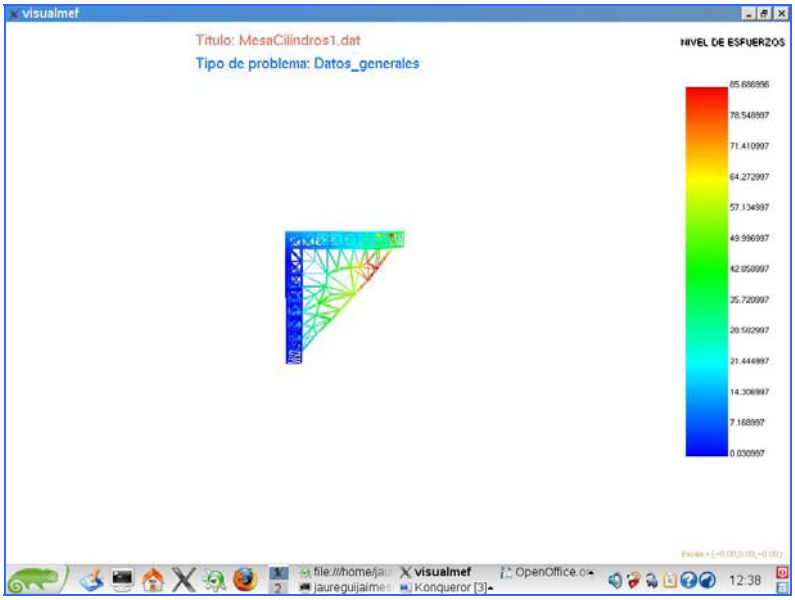


Fig. 50.- Malla triangulada del problema *Mesa* (programa VisualMEF)

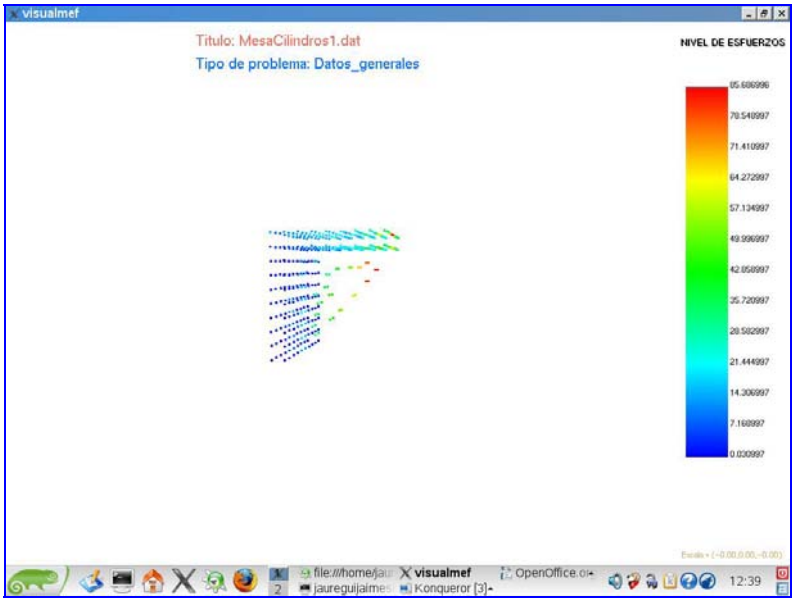


Fig. 51.- Nube de puntos del problema *Mesa* (programa VisualMEF)

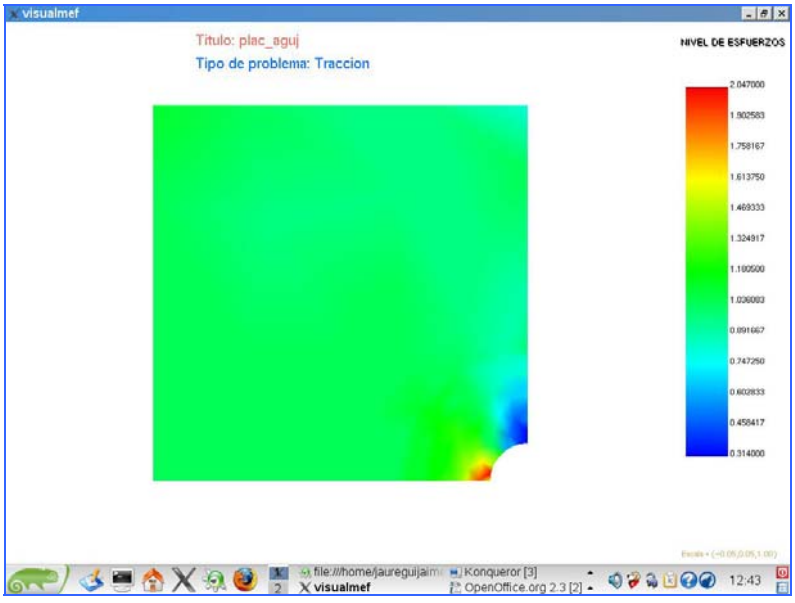


Fig. 52.- Resultados triangulados de *placa agujero* (programa VisualMEF)

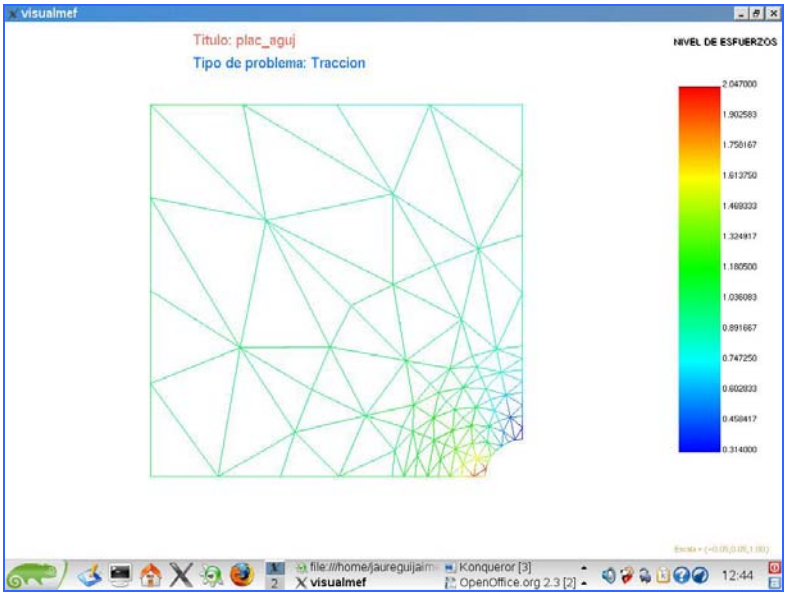


Fig. 53.- Mallado triangulado de *placa agujero* (programa VisualMEF).

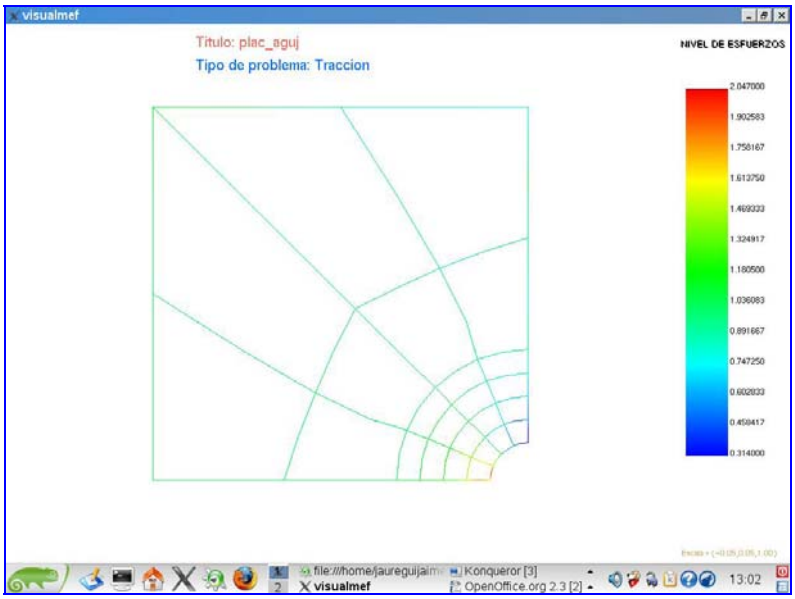


Fig. 54.- Mallado original de *placa agujero* (programa VisualMEF)

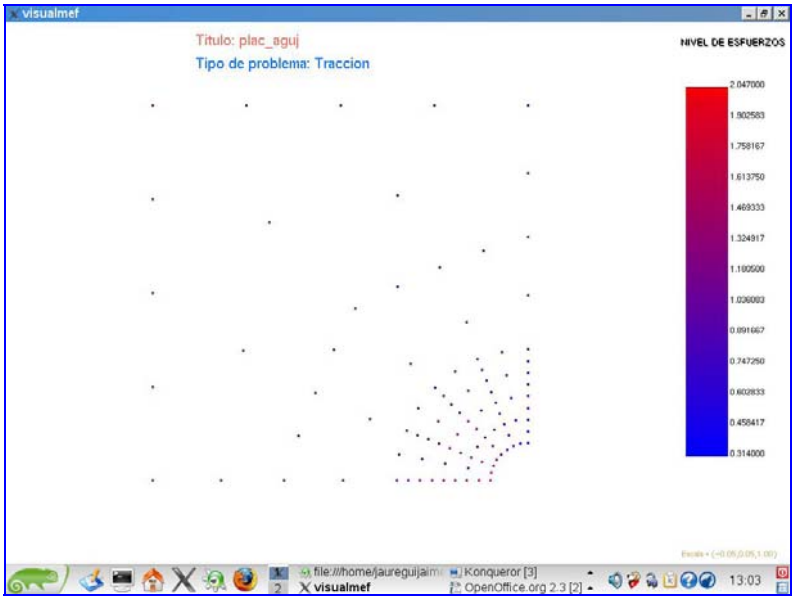


Fig. 55.- Nube de puntos de *placa agujero* (programa VisualMEF)

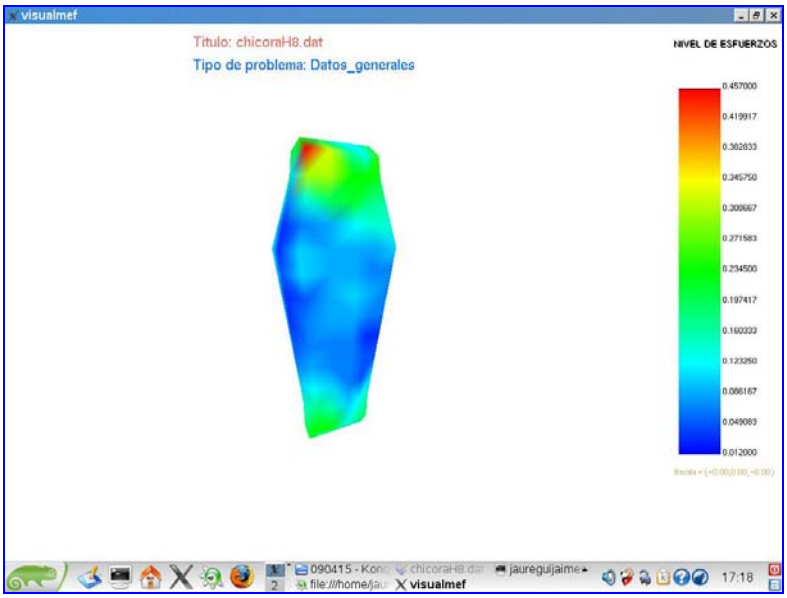


Fig. 56.- *chícoraH8* triangulada. (programa VisualMEF)

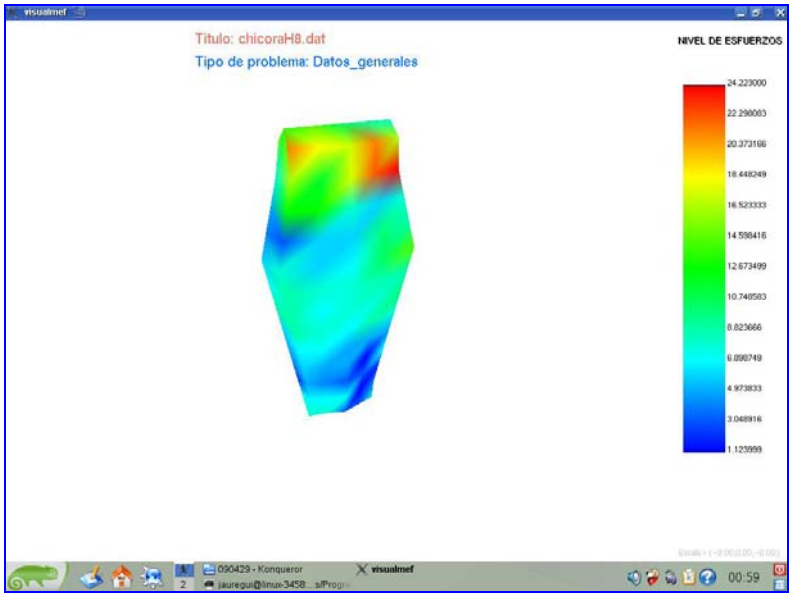


Fig. 57.- *chícoraH8* no triangulada. (programa VisualMEF)

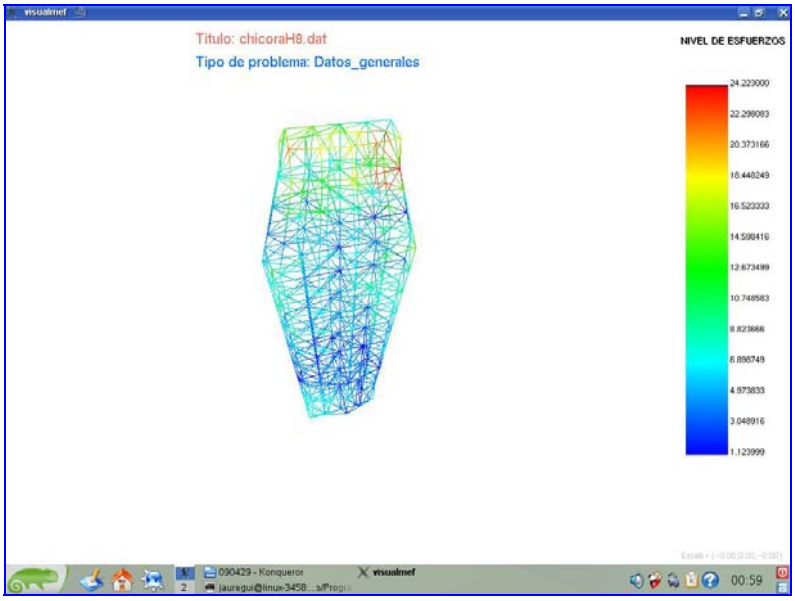


Fig. 58.- *chícoraH8* malla triangulada. (programa VisualMEF)

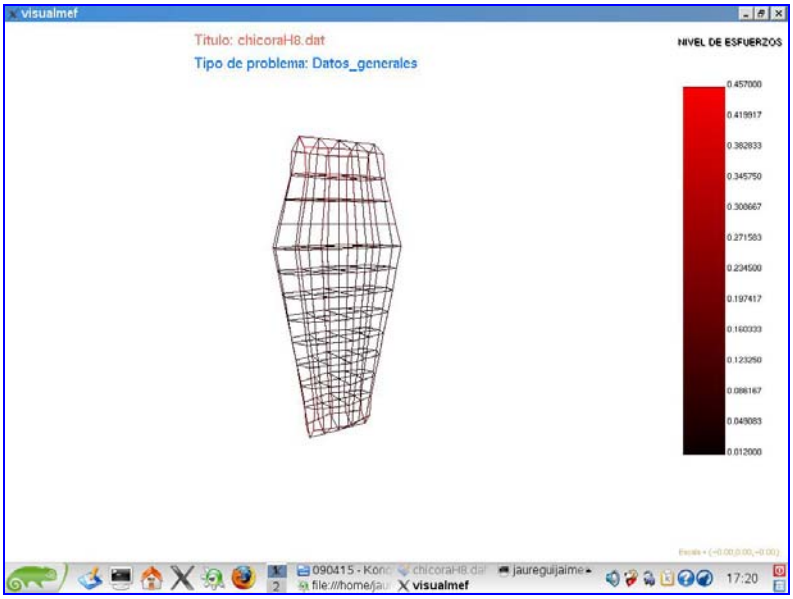


Fig. 59.- *chícoraH8* malla original(programa VisualMEF)

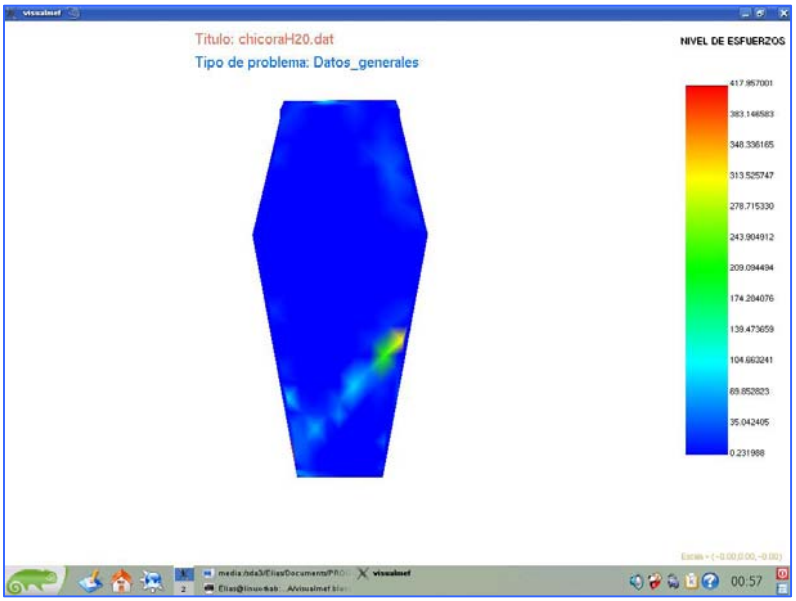


Fig. 60.- *chicoraH20* triangulada. (programa VisualMEF)

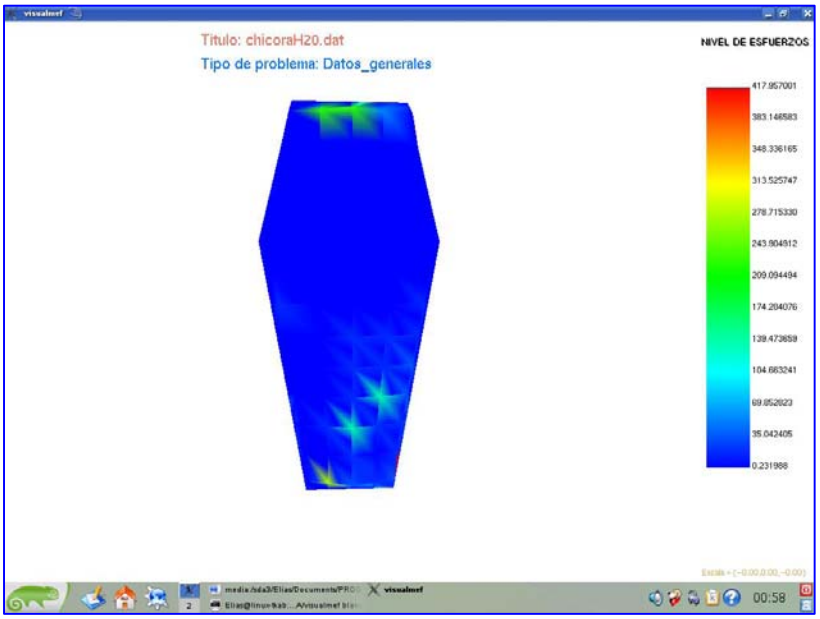


Fig. 61.- *chicoraH20* no triangulada. (programa VisualMEF)

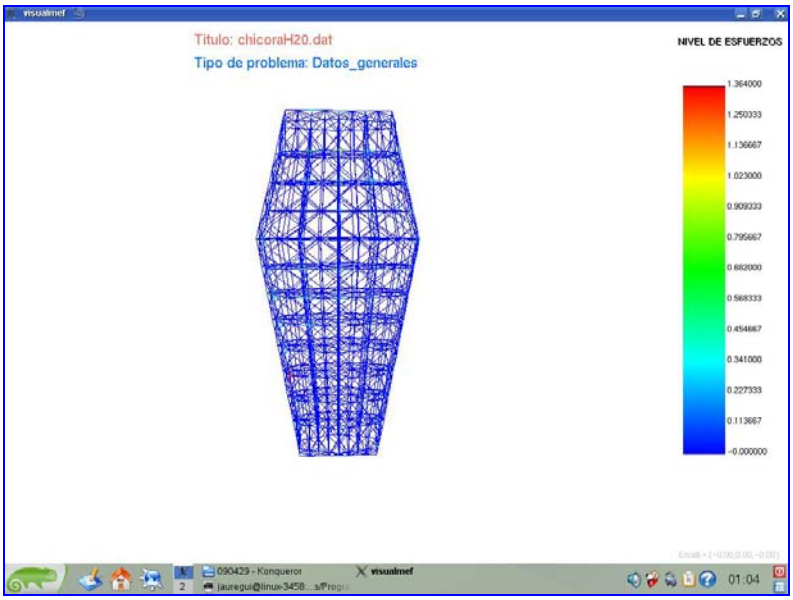


Fig. 62.- *chicoraH20* malla triangulada. (programa VisualMEF)

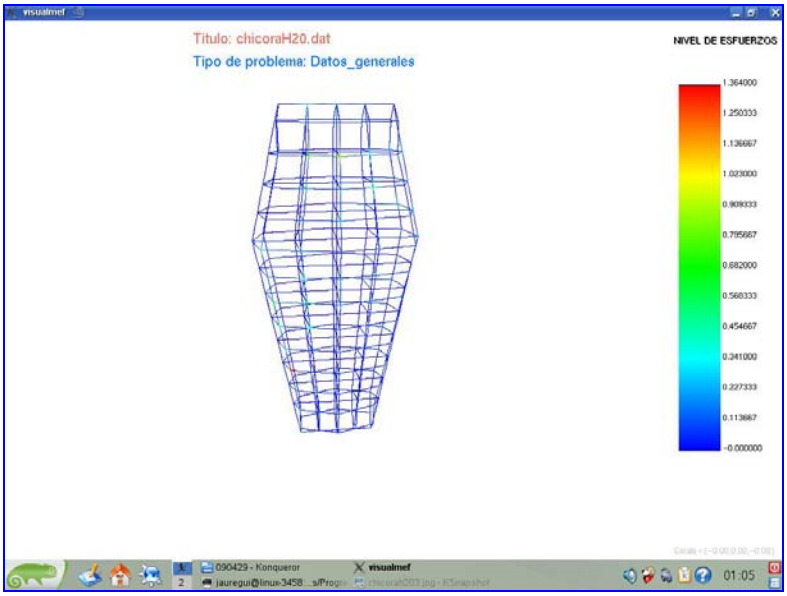


Fig. 63.- *chicoraH20* malla original. (programa VisualMEF)

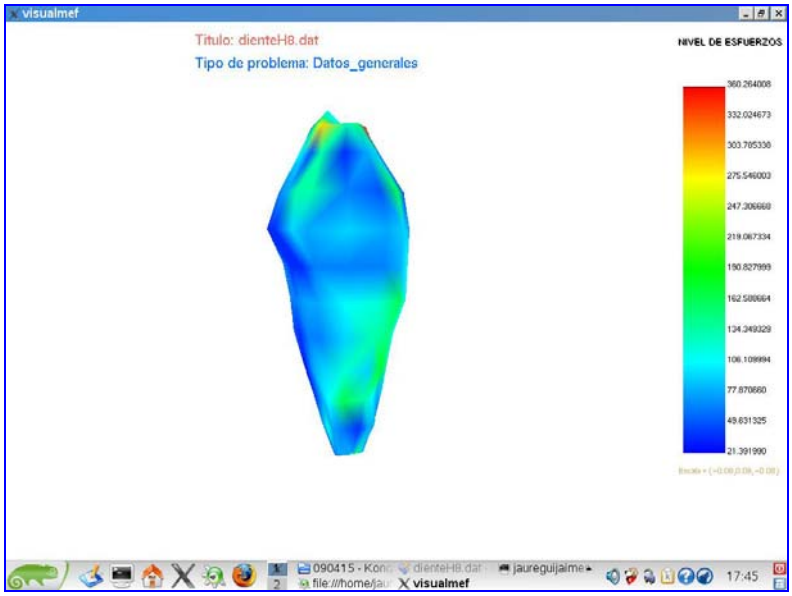


Fig. 64.- diente*H8* triangulado (programa VisualMEF)

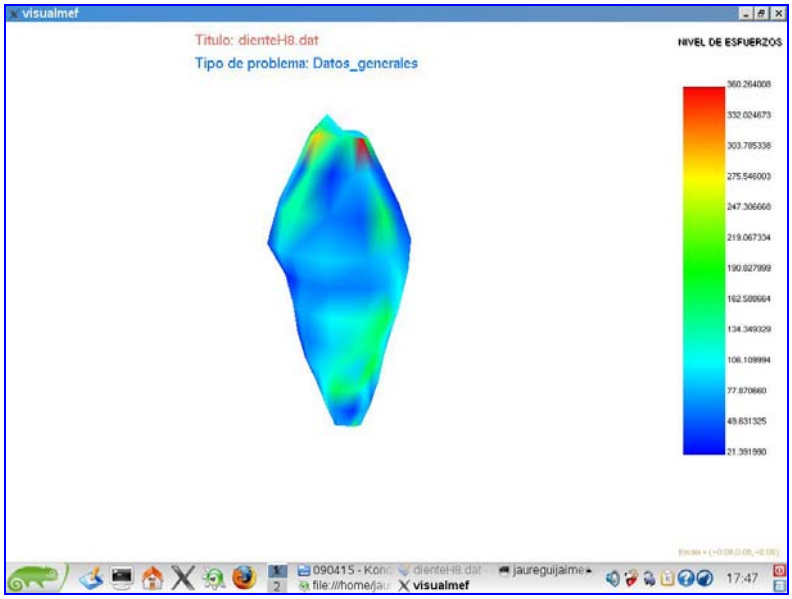


Fig. 65.- diente*H8* no triangulado (programa VisualMEF)

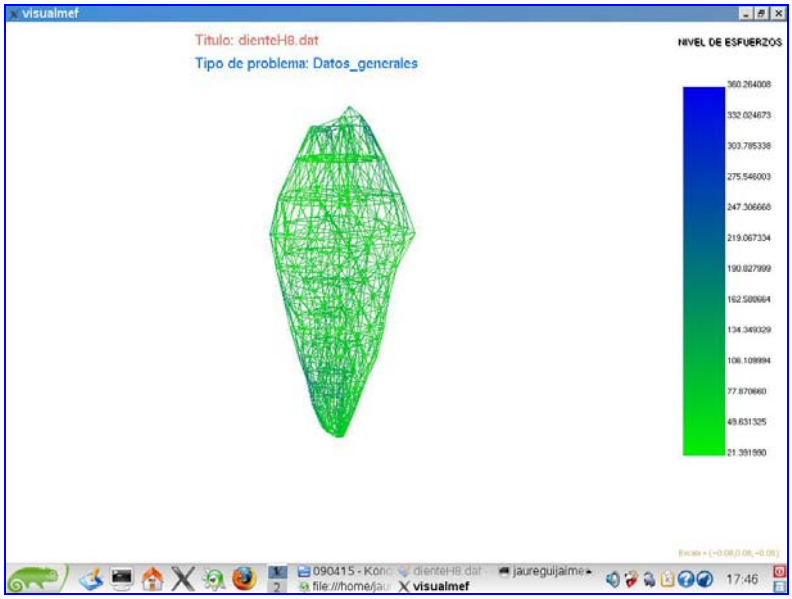


Fig. 66.- Mallado triangulado *dienteH8* (programa VisualMEF)

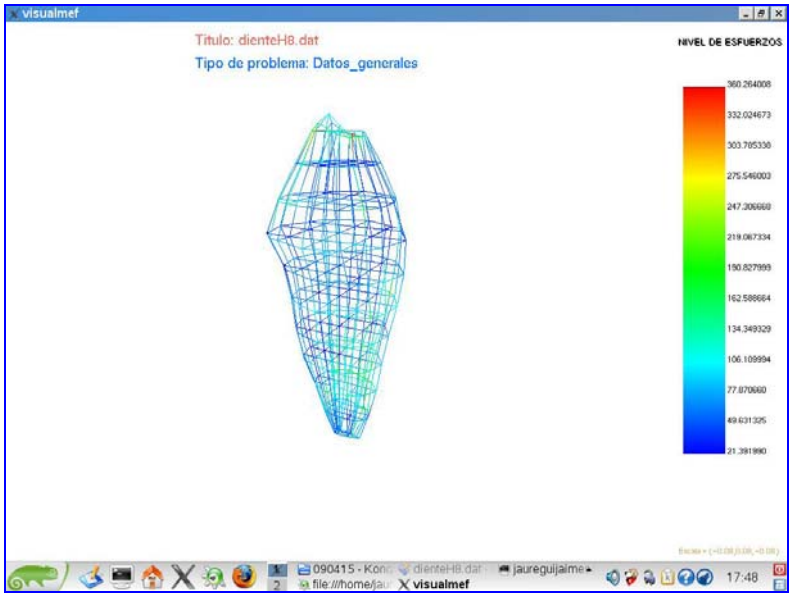


Fig. 67.- Malla original *dienteH8* (programa VisualMEF)

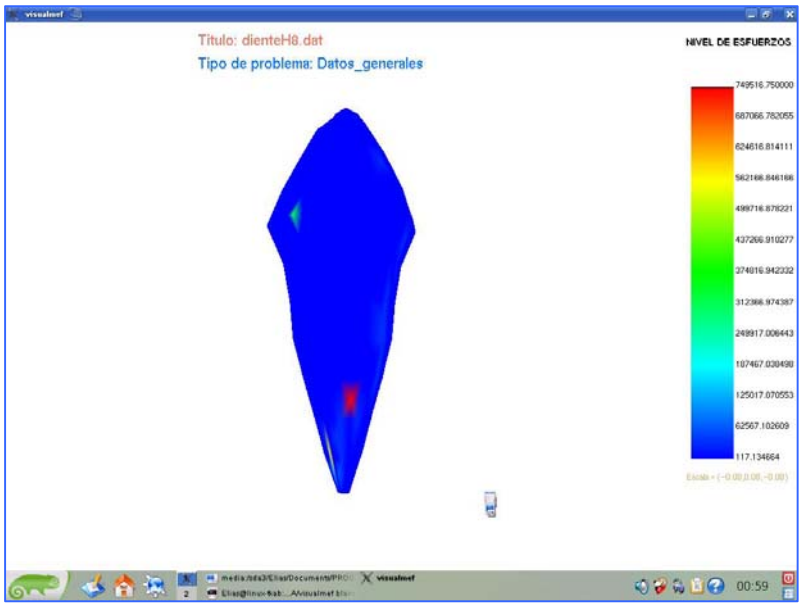


Fig. 68.- *dienteH20* triangulado (programa VisualMEF)

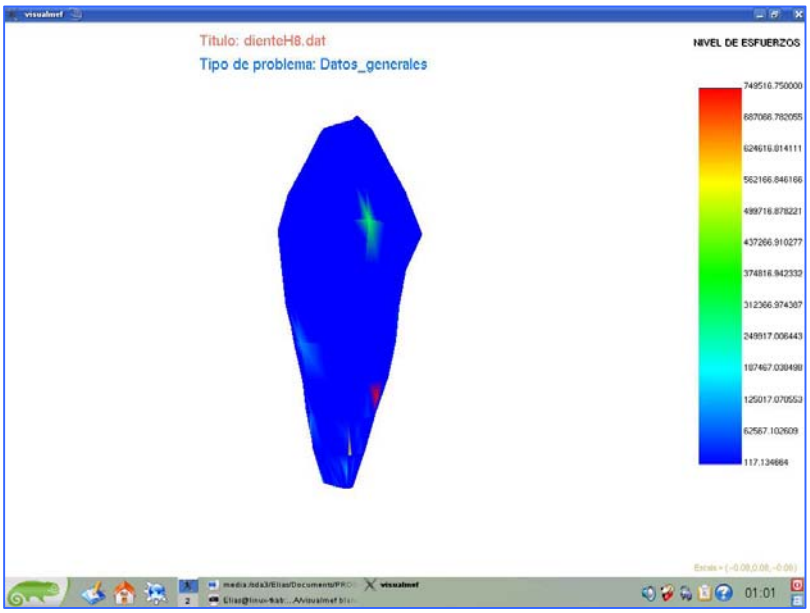


Fig. 69.- *dienteH20* no triangulado (programa VisualMEF)

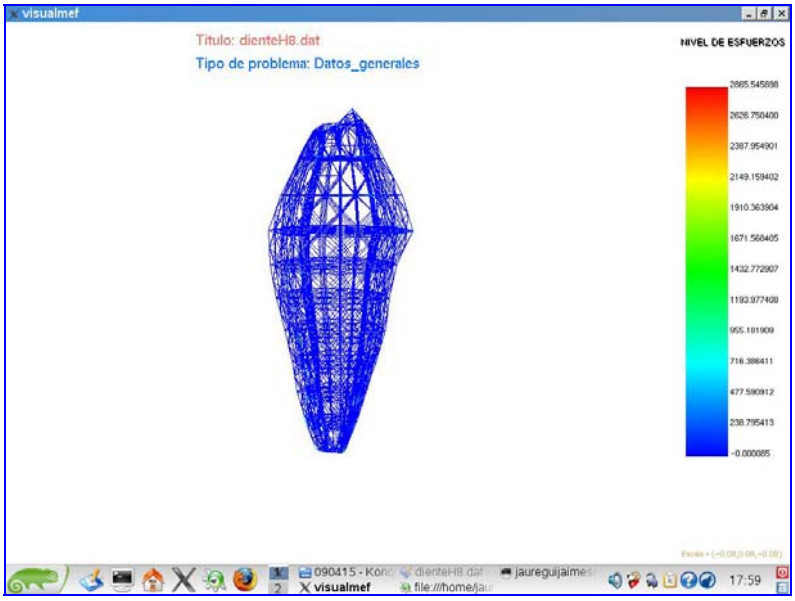


Fig. 70.- Mallado triangulado diente*H20* (programa VisualMEF)

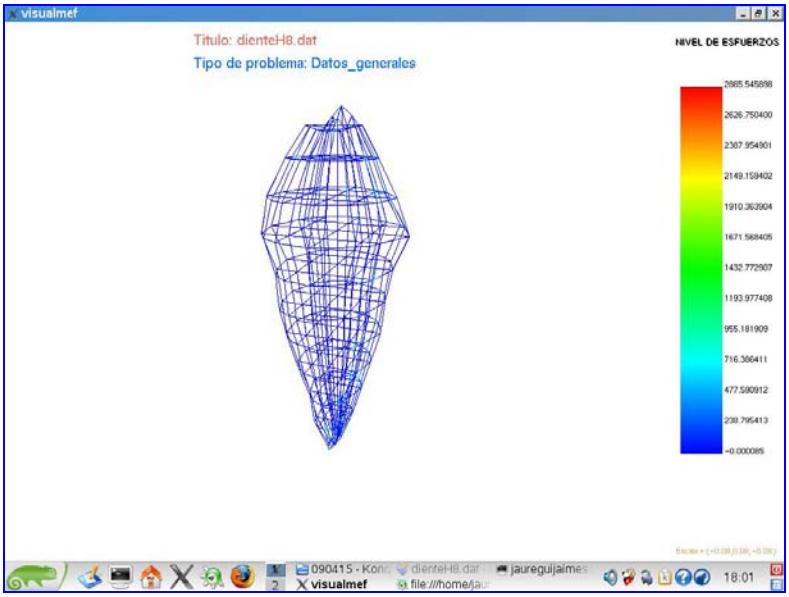


Fig. 71.- Mallado original *dienteH20* (programa VisualMEF)

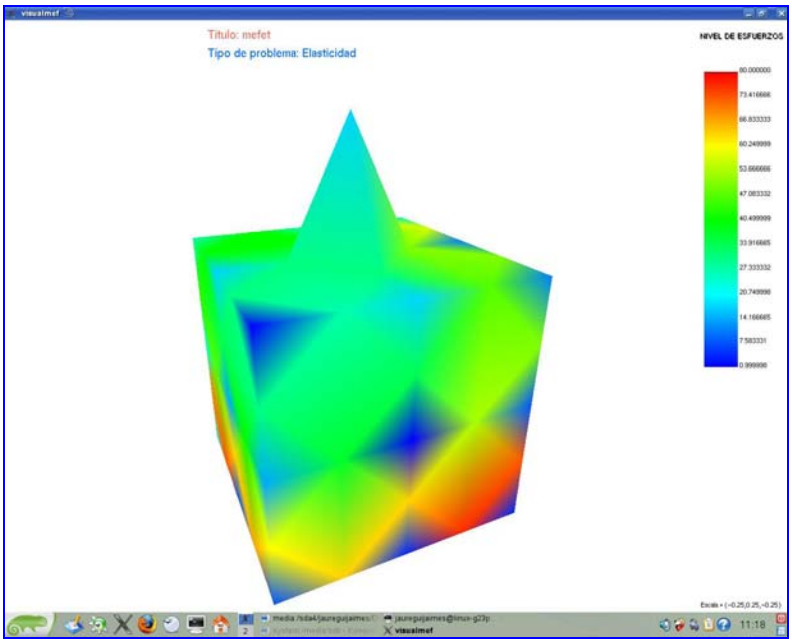


Fig. 72.- *mefet20 con pirámide* triangulado

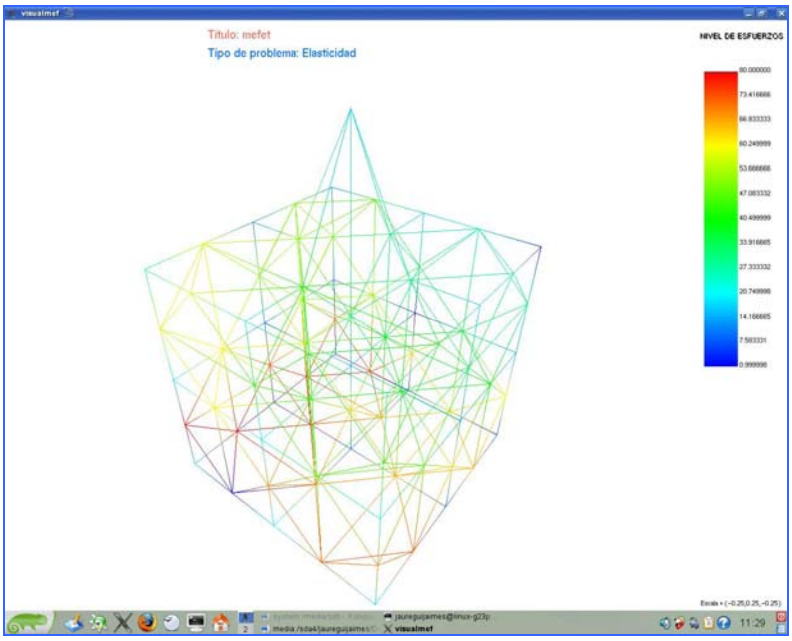


Fig. 73.- Malla triangulada *mefet20 con pirámide*

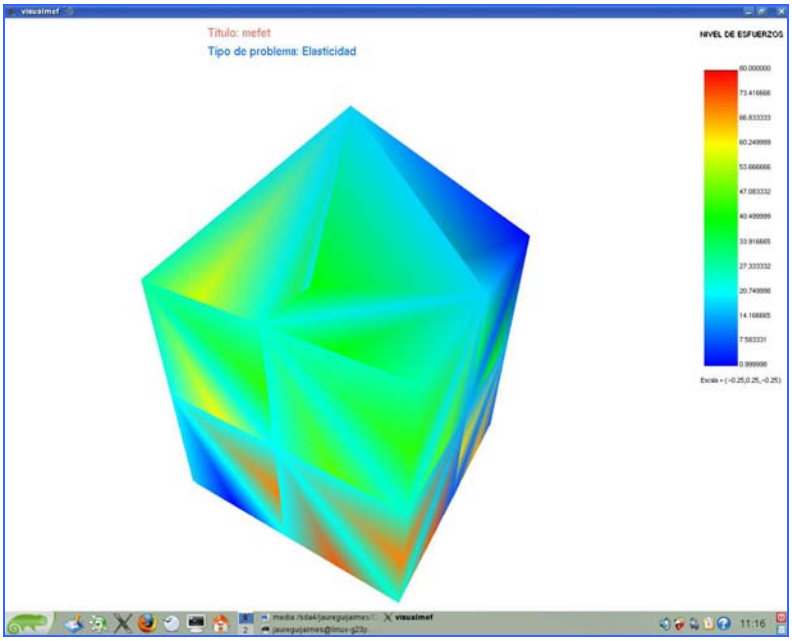


Fig. 74.- *mefet20 con pirámide* sin triangular

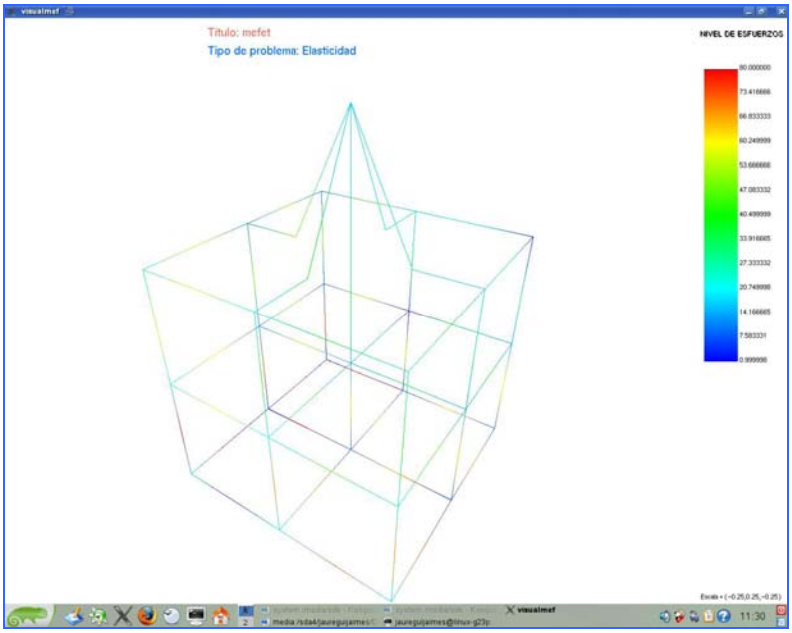


Fig. 75.- Malla original *mefet20 con pirámide*

CAPÍTULO VII
ANÁLISIS Y DISCUSIÓN DE RESULTADOS

- El programa VisualMEF puede graficar una data obtenida del análisis numérico, particularmente todos los tipos de elementos con los que cuenta el programa MEFET actualmente.

- Se visualizan en pantalla los gráficos con sus superficies trianguladas, se observa la diferencia respecto al programa original. Aunque el programa por defecto triangula las superficies, se mantiene la opción de graficar los datos con su malla sin triangular.

- Se notó durante la visualización de piezas desarrolladas con elementos hexaedros de 20 nodos, *chicoraH20* y *dienteH20*, que al ser procesados con el programa MEFET y visualizados no se obtenían los resultados gráficos esperados.

- El ejercicio *mefet20 con pirámide* es un caso interesante ya que al cambiar de triangulado a malla original la superficie que contiene la pirámide se muestra muy distinta al mallado triangulado.

- El programa durante su ejecución despliega en pantalla una ventana principal que contiene la pieza y una ventana donde esta contenida una paleta de colores con valores de la variable de estudio intercalados a lo largo de esta.

- Al hacer clic derecho con el ratón se despliega en pantalla un menú de opciones que permite cambiar la vista de la pieza a mallado, nube de puntos, relleno. Además permite activar o desactivar la luz, cambiar el color de la paleta , mostrar planos coordenados, el tipo de proyección, color de fondo, mostrar y ocultar deformaciones y la forma en que se dibuja la malla, triangulada u original.

- Cuando se visualiza con el programa **V*isualMEF*** se observan funciones de control a través del teclado, cuyas opciones son explicadas en el texto de ayuda que se muestra en la consola. Hay que destacar la opción de zoom que se realiza solo por este medio.

CAPÍTULO VIII

CONCLUSIONES Y RECOMENDACIONES:

Conclusiones

- Como resultado del presente trabajo se expandieron las capacidades del programa visualizador de resultados obtenidos mediante un programa de simulación numérica aplicando el método de los elementos finitos.

- Se realizó un análisis a fondo de la estructura y funcionamiento del programa visualizador de resultados original que permitió realizar la expansión del mismo.

- El programa desarrollado es capaz de graficar imágenes de piezas en dos (2D) y tres dimensiones (3D), triangulando la superficie de los elementos sin modificarlos. La triangulación de los elementos permite una mejora sustancial en la calidad de la distribución de los colores sobre la superficie.

- Se descubrió a través de los ejercicios con elementos H20 que no se obtuvo el resultado de visualización esperado, esto debido a que los nodos intermedios se observan con un color correspondiente a un valor de esfuerzo igual a cero (azul), al revisar los archivos de datos se detectó que los nodos intermedios del archivo de resultados (archivo 2 de datos) poseen un desplazamiento y un esfuerzo de valor igual acero, mientras que en los nodos principales el valor es distinto a cero, razón por la cual se origina una visualización en la que no se aprecia una degradación de

color progresiva en nuestro programa VisualMEF, de manera tal que se concluye que el programa MEFET presentaba una irregularidad en el procesado de este tipo de elemento, este inconveniente con el programa MEFET fue corregido.

- En el programa se ampliaron los tipos de elementos que se pueden visualizar, cumpliendo con la tipología de:

    - Cuadriláteros de cuatro (4) nodos
    - Cuadriláteros de ocho (8) nodos
    - Hexaedros de ocho (8) nodos
    - Hexaedros de veinte (20) nodos

- Con este programa se pueden visualizar todos los resultados obtenidos del programa de simulación MEFET , mostrando en pantalla la pieza estudiada y los valores de la variable de estudio mediante una escala de colores, hay siete (7) escalas de colores diferentes.

- Las modificaciones realizadas al código fuente fueron completamente desarrolladas en lenguaje C++, conservando el esquema de trabajo, el cual es estructurado y no orientado a objetos. Se utilizó las bibliotecas de OpenGL y GLUT, las cuales permiten el manejo de información gráfica, además de las bibliotecas estándar de entrada y salida de datos y manejo de archivos.

- El programa conserva las herramientas y facilidades del programa original, como el uso del teclado y/o el ratón con la ayuda de menús gráficos en la pantalla, así como las opciones disponibles mediante el

teclado presentadas en la consola paralela a la ventana de graficación de OpenGL.

- El programa fue probado con ejercicios de calibración para todos los elementos, tanto para dos (2) como para (3) dimensiones, además de aplicarse análisis numérico a piezas desarrolladas en este trabajo.

- Se hicieron modificaciones sobre el formato del archivo de datos y resultados destinados a facilitar su uso tanto para este programa como para el simulador.

- El fenómeno ocurrido en el ejercicio *mefet20 piramide* es un error conocido como flooding o inundación de color. Esto ocurre debido a que al exagerar el desplazamiento de un nodo de la cara del *mefet20 se* genera una superficie no contemplada en el dibujo, motivado a que el dibujador al usar polígonos como por ejemplo el cuadrilatero de ocho nodos ocurre lo descrito anteriormente, lo cual queda superado al usar la triangulación ya que el triángulo es mucho más simple y se adapta mejor a la deformación antes descrita.

Recomendaciones

- Aún es posible seguir expandiendo las capacidades de este programa, algunas de estas pueden ser:
    - Crear una rutina para que el visualizador pueda mostrar solo la malla original sin la necesidad de un archivo de resultados.
    - Desarrollar una escala de valores constante para estudiar las deformaciones de manera independiente.
    - Implementar la posibilidad de guardar, imprimir, modificar los datos de entrada, etc. que permitan mejorar el uso del programa.
    - Mostrar en pantalla las condiciones a la que esta sometida la pieza, como esfuerzos y restricciones.
    - Capacitar al programa para el manejo de resultados de análisis dinámicos, es decir que muestre los resultados como animaciones.

- Puede utilizarse este programa como visualizador de otros tipos de análisis numéricos,por ejemplo en el área de mecánica de fluidos, transferencia de calor, termodinámica, entre otras, solo hay que realizar una rutina de lectura de datos adecuada al formato de los archivos de datos a visualizar.

- Al realizar un análisis de simulación numérica sería aconsejable incluir material informativo adicional que permita una mejor interpretación de las condiciones del problema, esto para evitar confusiones de archivos y su comprensión en un futuro, este material pudiera contener los siguientes datos:

- Una descripción general de la pieza, descripción de los materiales que la componen y unidades del modulo de Young que correspondan a cada material.
- Unidades de las dimensiones de la pieza y de las coordenadas de los nodos.
- Una descripción explícita de las fuerzas aplicadas para la cargas de estudio de la pieza así como su magnitud dirección y sentido. De igual manera hacer una misma descripción de las condiciones de borde de la pieza.

REFERENCIAS

(1) Akin J. y Gray W. (1977). Contouring On Isoparametric Surfaces. International Journal for Numerical Methods in Engineering, Vol 11.

(2) Akin, J. y Gray, W. (1979). An Improved Methods For Contouring On Isoparametric Surfaces. International Journal for Numerical Methods in Engineering, Vol 14.

(3) Akin, J., Gray, W. y Zhang, Q. (1984). Coloring isoparametric contour, Engineering Computers, Vol 1.

(4) Alonzo, A. y Oramas, M. (1991). Elaboración e implementación de un sistema de representación gráfica de entidades geométricas bi y tridimensionales en computadores personales, con aplicación al diseño mecánico y a la bioingeniería, Universidad Central de Venezuela, Facultad de Ingeniería, EIM.

(5) Barragán, A. (1991). Implantación de técnicas de postprocesamiento en la aplicación del método de los elementos finitos. 1ra Conferencia de Visualización Científica, Intevep.

(6) Foley J. y otros (1994). Introduction to Computer Graphics.

(7) Jaen, H. y López, J. (1990). Desarrollo de modelos computacionales de representación gráfica para problemas de elasticidad y aplicación en el análisis de esfuerzos en órganos dentarios mediante el método de

elementos finitos, Universidad Central de Venezuela, Facultad de Ingeniería, EIM.

(8) Kenneeth, A. y Cross, K. (1989). Smootin Outdata Display Error. Mechanical Engineering,

(9) Lynes, J. y Asquith, H. (1985) A Simple Plotting Program For Finite Elements Output. Advance Engineering Software, Vol 5, Nº 1, pag 23-31

(10) Martín, M. (2008). Desarrollo e implementación de un programa de visualización de resultados para programas de simulación por elementos finitos y elementos de contorno. Universidad Central de Venezuela, Facultad de ingeniería, EIM.

(11) Nakamae, E. y otros. (1983).Color Computer Graphics in Magnetic Field análisis by means of finite elements method. Comput & Graphics,

(12) Noya, J. y Rabinovich, N. (1990). Implementación de modelos de cálculo para problemas no lineales mediante el método de los elementos finitos y aplicación al cálculo de esfuerzos de contacto en un órgano dentario, Universidad Central de Venezuela, Facultad de Ingeniería, EIM.

(13) Ordáz, E. y Pulido, O. (1992). Resolución de problemas geométricos asociados a la generación automática de mallas tridimensionales de elementos finitos en piezas de forma libre, Universidad Central de Venezuela, Facultad de Ingeniería, EIM.

(14) Puig – Pey, J. (1985). A simple algorithm for the graphical representation of surfaces. Microsoftware For Engineers, Vol 1, Nº 1.

(15) Quiroz, L. y Dufeo, E. (1989). Un postprocesador gráfico independiente de los dispositivos. Revista internacional de métodos numéricos para cálculo y diseño en ingeniería, vol 5.

(16) Stelzer, F. (1984). A Simple But Effective Methods to produce color in F.E.M., Results Presentation. Engineerin Computers, Vol 1.

(17) Stelzer, F. y Welzel, R. (1987). Plotting of contour in a natural way. International Journal for Numerical Method in Engineering, Vol 24.

(18) Yeo, M. (1984). An Interactive contour plotting program. Engineering Computers, Vol 1.

BIBLIOGRAFÍA

Alcocer R. (2001). Flat Shading usando OpenGL, Vectores Normales a una Cara. [Documento en línea]. Disponible: http://www.geocities.com/valcoey/nor-males.html [Consulta: 2008 octubre 3].

Cervantes H. (2008). Taller de Visualización 3D. Una introducción a la programación de aplicaciones en las estaciones de trabajo Silicon Graphics. [Taller en línea]. Universidad Autónoma Metropolitana, Unidad Iztapalapa, México. Disponible:http://www.humbertocervantes.net/homepage/itzamna/TUTORIAL/tutorial.html [Consulta: 2008, septiembre 4].

Foley J. y otros (1994). Introduction to Computer Graphics. Estados Unidos. Addison Weslley Publishing Company, Inc. 557 pág.

Gottfried, B. (1997). Programación en C, 2da ed. Madrid, España. Mac Graw Hill. 659 pág.

Kernighan B. y Ritchie D. (1991) . El Lenguaje de Programación C. Segunda Edición. México. Editorial Pretince Hall. 294 pág.

Marcos M. y García A (2004). Luces y Lámparas [Apuntes en línea]. Universidad de Salamanca, Grupo de Investigación, España. Disponible: http://gsii-.usal.es/~corchado/igrafica/descargas/temas/Tema08.ppt [Consulta: 2008, septiembre 4].

Martín, M. (2008) Desarrollo e Implementación de un Programa de Visualización de Resultados para Programas de Simulación por elementos finitos y elementos de contorno. Trabajo especial de grado. Inédito. Universidad Central de Venezuela, Facultad de Ingeniería, EIM, Caracas.

Mata S. y otros (2004). Gráficos y Visualización 3D. Iluminación: transparencias [Apuntes en línea]. División de Arquitectura de Computadores de la Universidad Rey Juan Carlos, España. Disponible: http://cgfiunam.-googlepages.com/2008-2Phong.ppt [Consulta: 2008, septiembre 4].

Menendez C. (2006). Tipo de Datos & Estructura de datos [Documento en linea]. Disponible: http://www.geoavila.com/blog/descargablesblog/2006/03/TIPOS-%20DE%20DATOS%20C.doc [Consulta: 2008, septiembre 11].

Mikaty M., Padrón M. y Figuera L. (2005). Instructivo para la Presentación de Tesis de Pregrado, Postgrado, Doctorado y Trabajos de Ascenso. Universidad Central de Venezuela, Subcomisión de Bibliotecas para la Normalización de la Presentación de Trabajos Especiales de Grado y Subcomisión de Deposito Legal.

Neider J., Davis T. y Woo M. (1993). OpenGL Programing Guide, The Official Guide to Learning OpenGL, Release 1. Estados Unidos. Addison Weslley Publishing Company, Inc. 516 pág.

Pozo S. (2003) Curso de C++. [Libro en línea]. Con Clase, Cursos en la Red, España. Disponible: http://c.conclase.net/curso/index.php [Consulta: 2008, septiembre 11].

Universidad Complutense Madrid (2005). Librería de Utilidades (glu.h). [Documento en línea]. Facultad de Informática, Universidad Complutense Madrid, España. Disponible:http://www.fdi.ucm.es/profesor/segundo/Documentacion/GLu.htm
[Consulta: 2008 octubre 20].

Universidad Complutense Madrid. (2005). Librería GL. [Documento en línea]. Facultad de Informática, Universidad Complutense Madrid, España. Disponible:
http://www.fdi.ucm.es/profesor/segundo/Documentacion/GL.htm
[Consulta: 2008 octubre 17].

Villanueva A. (2002). Una Aproximación a OpenGL. [Documento en línea]. Universidad de la Coruña, Departamento de Tecnologías de la Información y las Comunicaciones, España. Disponible: http://rnasa.tic.udc.es/gc/Tutorial-%20OpenGL/files/tutorial-opengl.pdf
[Consulta: 2008 octubre 3].