Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVIII, págs. 327-341 (artículo completo)
Cristian García Bauza, Pablo Lotito, Lisandro Parente, Marcelo Vénere (Eds.)
Tandil, Argentina, 3-6 Noviembre 2009

APLICACIÓN COMPUTACIONAL ORIENTADA AL ANÁLISIS DEL COMPORTAMIENTO DE GEOMATERIALES

Fernando Albarracín[a], **Victorio Sonzogni**[b], **Lía Orosco**[a]

[a]*Facultad de Ingeniería e Informática, Universidad Católica de Salta, Campo Castañares, 4400 Salta, Argentina, fjavier.gtk@gmail.com*

[b]*Universidad Nacional del Litoral, Bv. Pellegrini 2750, 3000 Santa Fe, Argentina,*
sonzogni@intec.unl.edu.ar

Palabras claves: Suelo, Acción Dinámica, Elementos Finitos, Linux.

Resumen. La aplicación computacional que aquí se presenta y que actualmente se encuentra en fase de desarrollo, es continuación de una anterior programada originalmente para sistemas operativos Windows y que inicialmente solo abarcaba el análisis elástico lineal en el plano y en el espacio. Posteriormente, su mismo autor efectuó la "migración" del código fuente hacia plataformas con núcleo Linux. Específicamente se trabaja con herramientas GNU/Linux, empleándose en la actualidad las distribuciones Debian GNU/Linux y Ubuntu Linux de 64 bits.

El objetivo de este código será aplicarlo para el estudio de la respuesta global de la estructura de suelo en el Valle de Lerma, en la región donde se asienta la Ciudad de Salta, ante la acción dinámica de ondas producidas por movimientos sísmicos intensos. Este es un valle sedimentario constituido principalmente por mantos cuaternarios de distinto espesor. Se deberá desarrollar entonces, un programa de elementos finitos y elementos de borde (boundary elements) adecuado a ser aplicado a geomateriales (elementos apropiados, modelos constitutivos, etc) y modelar los efectos de cuenco o vasija que se operan en el Valle de Lerma de Salta durante eventos sísmicos, efectuando estudios bidimensionales, en principio. Si bien éste es el objetivo principal, el programa computacional no se circunscribe a este problema en particular, siendo su aplicación de carácter mucho más general.

Esta aplicación se desarrolla en C/C++ y consta de un entorno para escritorios gráficos como Gnome, KDE y otros, que permite la introducción de datos de forma gráfica (CAD), trabajando por planos o directamente en el espacio, con herramientas de mallado aún en desarrollo (Pre-Procesador). Este mismo entorno permite la visualización de resultados obtenidos, ya sea como texto (script) o de forma gráfico/cromática, pudiendo hacer uso de tecnología OpenGL (Pos-Procesador).

El Procesador de cálculo se encuentra actualmente "empotrado" en el entorno gráfico único, solo por conveniencia de desarrollo. Finalmente constituirá un motor de cálculo externo para obtener el máximo rendimiento de los recursos disponibles.

El desarrollo de este Procesador, permitió el estudio del comportamiento de elementos curvos isoparamétricos lineales, cuadráticos y cúbicos de las familias de Lagrange y los denominados Serendípitos, así como la evaluación de los procesos de integración numérica asociados y otros.

Se quiere con esta presentación, mostrar las características de diseño del entorno gráfico de soporte, incluidas barras de herramientas por ejemplo, el sistema de script/CAD de ingreso de datos y visualización de resultados, así como algunos ejemplos analizados con fines académicos, volcando conclusiones obtenidas en la evaluación del comportamiento de los elementos finitos programados a la fecha.

1 INTRODUCCIÓN

El Valle de Lerma en la Provincia de Salta, es un valle sedimentario constituido principalmente por mantos cuaternarios de distinto espesor. El estudio de la respuesta global de la estructura de suelo en la región donde se asienta la Ciudad de Salta, ante la acción dinámica de ondas producidas por movimientos sísmicos intensos, constituye un proceso de análisis que se decidió abordar con software desarrollado dentro de la misma Facultad de Ingeniería e Informática de la Universidad Católica de Salta. Esta decisión surge por el hecho de contar con experiencia previa en programación y desarrollo del método de elementos finitos dentro de la mencionada Facultad, procurando continuar esta línea de estudio y formación de futuros investigadores.

Abordaremos en esta presentación, las instancias del desarrollo de esta aplicación hasta la fecha, haciendo hincapié en las instancias de decisión sobre las tecnologías elegidas: sistema operativo, lenguaje de programación, compilador, etc; en la estructura general adoptada para el desarrollo del programa; y en partes integrantes del código genérico.

2 LENGUAJE DE PROGRAMACIÓN

Esta aplicación está desarrollada con el lenguaje C++. Éste, es un lenguaje de programación orientado a objetos, constituyendo una versión expandida del lenguaje C.

El lenguaje C, es un potente y flexible lenguaje de programación, considerado por muchos especialistas como de “nivel medio”, lo que significa que posee características de lenguajes de bajo nivel (ensamblador) y de alto nivel (FORTRAN, BASIC, etc).

El poseer características similares al ensamblador, convierte a C en un lenguaje sumamente potente, capaz de manipular bits, bytes, palabras, punteros y direcciones de memoria, lo que no está permitido con los lenguajes de alto nivel. Se suele emplear el término “alto nivel”, para señalar un notable “alejamiento” del lenguaje con respecto a los elementos básico de la computadora. Esto le hace particularmente adecuado a C/C++ para la programación de sistemas operativos, por ejemplo, lo que no es posible con lenguajes de alto nivel.

Por otro lado, C también es un lenguaje estructurado. La característica distintiva de un lenguaje estructurado es la compartimentalización de código y datos. Se trata de la capacidad de un lenguaje de seccionar y esconder del resto del programa toda la información y las instrucciones necesarias para llevar a cabo una determinada tarea. Por ejemplo, dos funciones distintas pueden tener declaradas en su interior variables locales del mismo tipo y del mismo nombre sin que unas interfieran con otras. Esto hace posible que el código posea ninguna o pocas variables globales (variables conocidas en todo el programa). El uso excesivo de variables globales, puede provocar que los errores proliferen en el programa, al permitir efectos secundarios no deseados. Esto es lo ocurre en lenguajes como BASIC o FORTRAN. Los lenguajes modernos tienden a ser estructurados puesto que son más fáciles de mantener y expandir.

En el lenguaje C++, a todas estas importantes características del lenguaje C : potencia, portabilidad, flexibilidad, estructuración, etc., se suman las prestaciones de la orientación a objetos. Dependiendo del proyecto, un programa de 25,000 a 100,000 líneas se vuelve difícil de gestionar debido a que no es fácil tratarlo como un todo, por lo tanto en estos casos es imperativo el empleo de la programación orientada a objetos.

C++ es un superconjunto de C, y como todo programa orientado a objetos posee tres características sobresalientes: objetos, polimorfismo, herencia. Un objeto es simplemente una entidad lógica que contiene datos y un código que manipula esos datos, que proporciona un

significativo nivel de protección contra modificaciones accidentales o contra un uso incorrecto. El enlazado de código y dato de esta forma se denomina frecuentemente encapsulación. El polimorfismo, permite usar un nombre para varios propósitos relacionados pero ligeramente diferentes. La herencia es el proceso por el cual un objeto puede adquirir las propiedades de otro objeto (Herbert Schildt, 1994).

Más adelante se mostrarán otras características particulares que se incorporaron en estos últimos años de desarrollo a este potente lenguaje de programación llamado C++.

3 SISTEMA OPERATIVO

Inicialmente desarrollada para el sistema operativo Windows, esta aplicación fue migrada durante el año 2008 a sistemas operativos GNU/Linux y por lo tanto compatible con cualquiera que posea núcleo (kernel) Linux, UNIX o derivados de éste último, como ser BSD.

Si bien no entra en discusión en este trabajo la evaluación ni comparación entre distintos sistemas operativos, la decisión de migrar hacia plataformas con núcleo Linux fue tomada en base a múltiples parámetros, entre ellos la velocidad de procesamiento y gestión de memoria. Ambas resultaron superiores en plataformas Linux bajo las condiciones requeridas para el objetivo final de esta aplicación.

Se optó por emplear principalmente Debian GNU/Linux, uno de los principales sistemas operativos universales, que junto con las características comunes a toda plataforma Linux (y que si bien no restringe el uso de software privativo), nos permite el uso de software cien por ciento libre, incluidos compiladores C/C++ y Entorno Integrado de Desarrollo (IDE, del inglés Integrated Development Environment), garantizando cumplir en todo momento con el uso de la Licencia Pública General de GNU (GNU GPL, del inglés GNU General Public License). Esto es de suma importancia en este caso, ya que se trata, hasta la fecha, de software no comercial orientado a la investigación en la ingeniería.

3.1 Compilador C/C++ y entorno integrado de desarrollo

Se emplea como compilador C/C++ al potente gcc de GNU, las librerías libres de Qt y el entorno integrado de desarrollo KDevelop. Si bien Qt y KDevelop generan entornos gráficos de trabajo para el escritorio KDE principalmente, la aplicación corre igualmente en distintos escritorios de trabajo para Linux, como ser el GNOME, que es el escritorio desarrollado por GNU. Cabe recordar, que un escritorio gráfico es un entorno multitareas que nos permite hacer uso fácilmente de los recursos gráficos y otros disponibles, de manera “amigable”.

El empleo de un entorno gráfico de tareas tiene que ver con la estructura general de esta aplicación, como veremos a continuación.

4 ESTRUCTURA GENERAL DE LA APLICACIÓN

Está estructurada genéricamente en tres partes: Pre-procesador, Procesador, y Pos-procesador, que nos permitirán respectivamente, ingresar datos generales y geométricos; efectuar el cálculo y análisis obteniendo resultados; y finalmente, observar y analizar los resultados obtenidos.

Estas tres partes se encuentran totalmente integradas en un entorno gráfico y único de tareas, que con menús, barras de herramientas y ventanas de distintos tipos, permite un trabajo más fluido y cómodo (ver Figura 1).

Tanto el Pre-procesador como el Pos-procesador, hacen uso de recursos gráficos de manera significativa. De allí la decisión de trabajar sobre escritorios gráficos.

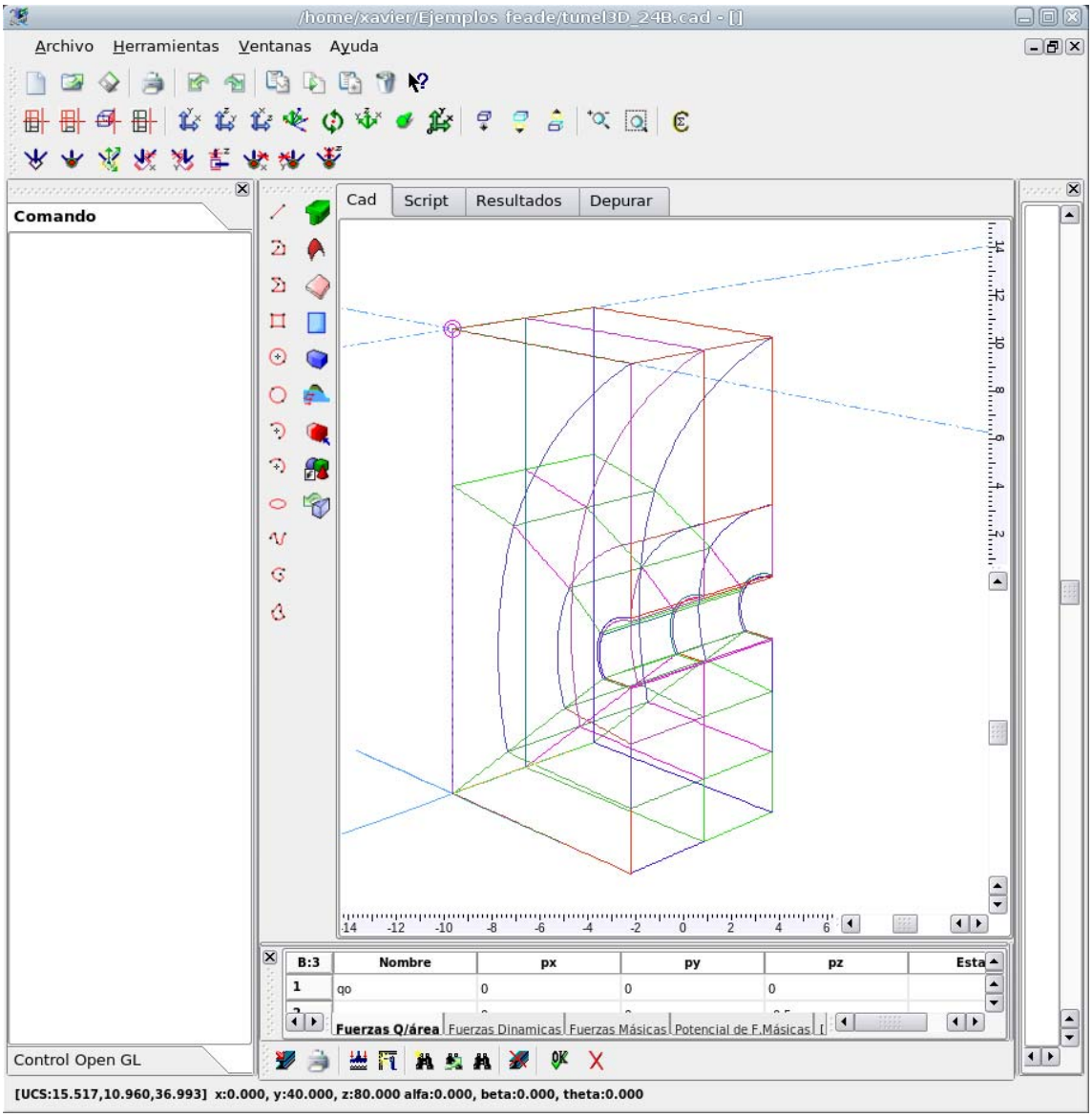

Figura 1. Entorno gráfico único – CAD en primer plano en trazado 3D

4.1 Entorno gráfico de trabajo

Posee los elementos típicos de la mayoría de los entornos para aplicaciones que se ejecutan bajo escritorios gráficos: barra de menú principal, barras de herramientas y ventanas de trabajo.

La ventana correspondiente a hojas de cálculo y tablas dinámicas, poseen elementos de diseño particulares que fueron diagramados y programados por el autor independientemente de las librerías Qt. La ventana gráfica de salida en la cual se empotra es parte de estas librerías.

Las hojas de cálculo poseen usos múltiples, entre ellos el ingreso de datos en forma de tablas dinámicas. Permite incorporar datos de distintos formatos y fórmulas, que pueden evaluarse inmediatamente o depender de variables que tomarán valor en el proceso que llame a la fórmula; por ejemplo, durante la integración temporal de la ecuación general de

movimiento (método de Newmark, HHT y otros). También pueden emplearse para mostrar resultados.

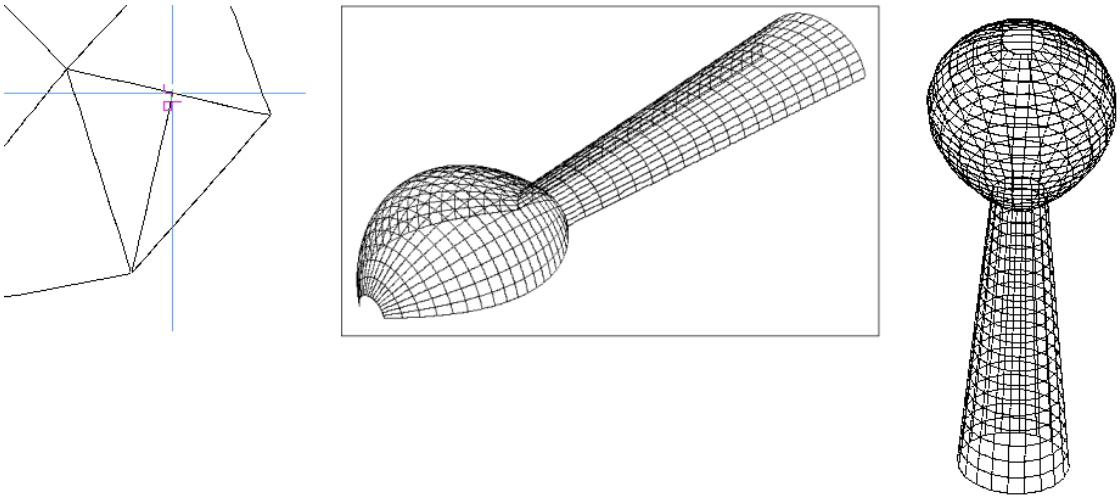

Figura 2. a) Trazado de línea perpendicular a otra por rastreo. b) y c) Mallado automático en 3D

4.2 Pre-procesador

Está constituido por tres herramientas de apoyo para el ingreso de parámetros y datos generales y geométricos: editor de texto; hojas de cálculos y tablas dinámicas; y principalmente, un sistema de diseño asistido por computadora (CAD, del inglés Computer Asisted Design), orientado a la discretización por elementos finitos.

Tanto las hojas de cálculo y tablas dinámicas, como el CAD, fueron totalmente desarrollados y programados por el autor de la aplicación, haciendo uso solamente de las herramientas y librerías básicas aportadas por el IDE. El editor de texto, es parte de las herramientas aportadas por las librerías de Qt.

Una particularidad del diseño asistido implementado, es que se puede interactuar entre el CAD y el editor de texto permutando entre una y otra ventana de asistencia. La regeneración o refresco de los datos al pasar de un sistema a otro se efectúa automáticamente y en tiempo real.

El CAD permite, entre otras cosas, el trazado automático o manual de mallas, empleando ratón y/o línea de comandos para herramientas. Las herramientas y otras acciones pueden elegirse desde barras de botones. Se han ido desarrollando un cierto número de malladores 2D y 3D, que permiten el discretizado de un dominio, en elementos curvos isoparamétricos (lineales, cuadráticos y cúbicos), de las familias lagrangiana y serendípita. El mallado se efectúa a partir de unos pocos elementos generatrices, si el dominio no puede representarse adecuadamente a través de funciones matemáticas. Estos malladores son aún objeto de estudio y desarrollo (ver Figura 2).

Para poder manipular los elementos gráficos en el CAD, se efectúa el trazado de las curvas, por ejemplo el borde de un elemento finito, a través de alguna función de forma y la correspondiente transformación de coordenadas (Zienkiewicz O.C., 1982, Capítulos 7 y 8); de esta manera se obtiene punto a punto dicho elemento gráfico básico. Es decir, los elementos finitos que se visualizan en pantalla, son exactamente los mismos que se integrarán durante el proceso de cálculo.

Naturalmente que las formas del dominio distintas a las polinómicas, serán aproximadas a

través de estas funciones de forma, pero puede lograrse la máxima aproximación antes del mallado final, representando los elementos generatrices del dominio con curvas apropiadas. Para ello, la aplicación incorpora funciones de forma circulares por ejemplo, para representar dichas fronteras del dominio. La aplicación emplea estas funciones particulares solo para representar los elementos generatrices, pero no queda descartado en un futuro, su estudio para generar elementos finitos.

Esta manera de generar curvas implica que pueden leerse o "rastrearse" punto a punto cualquiera de ellas y que solo se emplean las librerías de Qt para el trazado elemental de puntos y líneas entre puntos generados. La distancia entre puntos generados depende de la precisión de salida gráfica y rastreo, y puede modificarse desde la aplicación.

Todos los parámetros así como las propiedades constitutiva de los materiales y otros datos, se incorporan al problema a través de tablas dinámicas (y hojas de cálculo) y editor de texto. La asignación de propiedades puede efectuarse directamente desde el CAD con las herramientas de selección y asignación disponibles. Incluso pueden copiarse y pegarse propiedades entre elementos. Si el usuario desea, puede incorporar todo el problema (propiedades, generación de elementos, asignaciones, etc) directamente con el editor de texto (Script) o interactuando CAD-Script-tablas dinámicas.

4.3 Procesador

El procesador puede resolver problemas de elasticidad lineal en dos y tres dimensiones, y obtener modos y frecuencias naturales. Incluye elementos sólidos, planos, de placas, de cáscaras y de barras, empleándose exclusivamente integración numérica. Esto también incluye a los elementos de barra, ya que se emplean para ello elementos de tres nodos con dos puntos de integración. Estos pueden modelar barras rectas o de curvatura constante con unos pocos elementos.

Actualmente se encuentra en etapa de evaluación, la implementación desarrollada de los métodos de Newmark y HHT para la integración temporal de la ecuación general de movimiento. La etapa actual de desarrollo es la incorporación del análisis no lineal (geométrico y constitutivo) y plasticidad. Esta etapa abarca el estudio particular de modelos constitutivos para suelos.

Hablando de la estructura y organización del procesador, el motor de cálculo se encuentra "empotrado" en el entorno gráfico de trabajo. Esto obedece exclusivamente a razones de programación/evaluación de la aplicación, pero el proyecto del desarrollo de la misma, prevé como etapa final: a) que el procesador de análisis y cálculo sea independiente del entorno gráfico de pre-pos procesamiento; b) preparar el código para el procesamiento en paralelo.

La razón de esta decisión es la optimización en el uso de recursos y hardware: el procesador lee un archivo en disco con los datos del problema generados en el entorno gráfico, entrega los resultados en otro archivo en disco, que posteriormente podrán pos-procesarse en el entorno gráfico. Si la capacidad de hardware es suficiente para el problema, simplemente se ejecuta el procesador desde el entorno gráfico. Esto lo decidirá el usuario.

Recordemos que los escritorios gráficos en plataformas Linux, son aplicaciones destinadas a facilitar el uso de recursos de hardware y software. Cualquier distribución Linux (y esto incluye al sistema operativo Debian GNU/Linux) puede arrancar con un núcleo o kernel específico que se incorpora a la lista del gestor de arranque. Más aún, este núcleo puede ser compilado directamente en el sistema o hardware a utilizarse, creando una versión altamente especializada a la cual se le puede añadir solo lo necesario: todos los recursos pueden quedar disponibles para una única aplicación. Esta también fue otra de las razones que decidió la migración hacia plataformas Linux.

```
{
…
for(_e = 0; _e < n_E; _e++) {
  for(_n = 0; _n < n_N; _n++) {
   for(_k = 0; _k < n_K; _k++) {

    // Calcular la matriz "jacobiana" en el punto(_e,_n,_k) de Gauss
    Jacobiano=(this->*jacobiana[TipoFE])( feplate->pArist, FEMplano::lpnode,
                                          feplate->thickness, cfsignos, _k,
                                          (TabGauss[0]+_e)->a, (TabGauss[1]+_n)->a,
                                          (TabGauss[2]+_k)->a,
                                          Jacob, &AntJacobiano );

    if( !Jacobiano ) return FALSE; // error

    // Calcular la "Inversa" de la matriz jacobiana
    SystemEc::InversaGJ( Jacob, FilsJacob, InvJ );

    // Nodo i
    // Obtener dNoi/dx, dNoi/dy, dNoi/dz (vector dNoi[3])
    // Obtener dN1i/dx, dN1i/dy, dN1i/dz (vector dN1i[3])
    // Obtener Noi, N1i (vector Ni[3])
        (this->*dN01[TipoFE])( feplate->pArist, cinode, nodoI, feplate->thickness, isign, _k,
                  (TabGauss[0]+_e)->a, (TabGauss[1]+_n)->a, (TabGauss[2]+_k)->a,
                  InvJ, dNoi, dN1i, Ni, NULL ); // tambien retorna "dNe, dNn"

    // Obtener dV para la integración de Gauss-Legendre
    xCoefPeso_H = (TabGauss[0]+_e)->H * (TabGauss[1]+_n)->H * (TabGauss[2]+_k)->H;
    dV = xCoefPeso_H * Jacobiano;
    // Nodo j
    // Obtener dNoj/dx, dNoj/dy, dNoj/dz (vector dNoj[3])
    // Obtener dN1j/dx, dN1j/dy, dN1j/dz (vector dN1j[3])
    // Obtener Noj, N1j (vector Nj[2])
         (this->*dN01[TipoFE])( feplate->pArist, cjnode, nodoJ, feplate->thickness, jsign, _k,
                                (TabGauss[0]+_e)->a, (TabGauss[1]+_n)->a, (TabGauss[2]+_k)->a,
                                InvJ, dNoj, dN1j, Nj, NULL );

    (this->*AssemblerKij[TipoFE])( dNoi, dN1i, dNoj, dN1j, Ni, Nj, D, kij,
                                   feplate->pArist, FEMplano::lpnode, cfsignos, (nodoI+1), (nodoJ+1) );
…
}
```

La estructura interna definitiva del procesador de cálculo, fue determinada a partir de pruebas preliminares entre algunas opciones. Está claro que existen tantas formas de programar como programadores hay, por lo que también intervino inevitablemente el gusto y subjetividad del autor del código.

Intentando emplear al máximo la potencia de C/C++ se hace uso continuamente de dos herramientas potentes y flexibles: puntero a funciones y objetos; y la estructuración de "objeto orientado a objeto".

Con los punteros a funciones adecuadamente usados, pueden obtenerse mejoras en el rendimiento de velocidad. Por ejemplo, puede evitarse en muchos casos el uso de "if" dentro

de ciclos. En la Figura 3, se muestra el uso de puntero a funciones dentro del ciclo de generación y ensamblaje de la matriz de rigidez de un elemento (o submatriz). Mediante el uso de puntero a funciones y objetos, el mismo ciclo puede ser empleado para distintos elementos finitos, simplemente cambiando del índice "TipoFE" del ejemplo. Previamente deben declararse las funciones y los punteros a funciones correspondiente, para luego apuntarlos. Esto puede apreciarse en Figura 4 y Figura 5.

```
class FEMplano : public QFrame, public FEMplateNidNi, public FEMderivadas, public SystemEc {
        Q_OBJECT
        public:
…

        DOUBLE  (FEMplano::*jacobiana[MAXFETYPE])( PENTITY *Aristas,
                                LPTMAPNODOS * nodes,
                                ASSIGN * thickness, TYPECHAR * CFsignos, int ko,
                                DOUBLE E, DOUBLE N, DOUBLE K,  // pto de gauss
                                DOUBLE Jacob[9], DOUBLE *AntJacobiano );
                DOUBLE jacobiana0( PENTITY *Aristas, LPTMAPNODOS * nodes,
                                        ASSIGN * thickness, TYPECHAR * CFsignos, int ko,
                                        DOUBLE E, DOUBLE N, DOUBLE K,  // pto de gauss
                                        DOUBLE Jacob[9], DOUBLE *AntJacobiano );
                DOUBLE jacobianaPlano( PENTITY *Aristas, LPTMAPNODOS * nodes,
                                        ASSIGN * thickness, TYPECHAR * CFsignos, int ko,
                                        DOUBLE E, DOUBLE N, DOUBLE K,  // pto de gauss
                                        DOUBLE Jacob[9], DOUBLE *AntJacobiano );
                DOUBLE jacobianaPlaca( PENTITY *Aristas, LPTMAPNODOS * nodes,
                                        ASSIGN * thickness, TYPECHAR * CFsignos, int ko,
                                        DOUBLE E, DOUBLE N, DOUBLE K,  // pto de gauss
                                        DOUBLE Jacob[9], DOUBLE *AntJacobiano );
                DOUBLE jacobiana3D( PENTITY *Aristas, LPTMAPNODOS * nodes,
                                        ASSIGN * thickness, TYPECHAR * CFsignos, int ko,
                                        DOUBLE E, DOUBLE N, DOUBLE K,  // pto de gauss
                                        DOUBLE Jacob[9], DOUBLE *AntJacobiano );

…
};
```

Figura 4. Declaración de funciones y punteros dentro de una clase

De esta manera, se logra compactar el código beneficiando con ellos la evaluación de rutinas, el mantenimiento y la ampliación futura de la aplicación. Como se ve en el ejemplo, es muy simple incorporar algún otro elemento finito a la rutina general de ensamblaje: basta con crear un nuevo puntero, las funciones correspondientes y el índice del nuevo elemento.

Por supuesto que son posibles maneras distintas de lograr el algoritmo, algunas más o menos eficientes que otras. Como en esta aplicación los datos son tomados por las funciones desde listas enlazadas (Gottfried Byron S., 1991), esta organización del algoritmo mediante punteros a funciones y objetos se torna muy eficiente.

La estructuración de "objeto a objeto", es en realidad la forma integral como se decidió organizar la aplicación en conjunto. Este tipo de estructuración es empleada tanto en el entorno gráfico (señales de acciones y elección desde barras de herramientas, por ejemplo) como en los tres componentes de trabajo de la aplicación: pre-procesador, procesador y pos-procesador. Sin embargo es en el procesador de cálculo en donde la eficiencia y potencia del desarrollo de objetos orientados a objetos fue más evidente.

```
{
…
// puntero a las funciones de generación de la Matriz Jacobiana para cada elemento finito
FEMplano::jacobiana[0] = &FEMplano::jacobiana0;
FEMplano::jacobiana[1] = &FEMplano::jacobiana0;
FEMplano::jacobiana[2] = &FEMplano::jacobianaPlano;
FEMplano::jacobiana[3] = &FEMplano::jacobiana0;
FEMplano::jacobiana[4] = &FEMplano::jacobiana3D;
FEMplano::jacobiana[5] = &FEMplano::jacobianaPlaca;
FEMplano::jacobiana[6] = &FEMplano::jacobiana0;
FEMplano::jacobiana[7] = &FEMplano::jacobianaPlano;
FEMplano::jacobiana[8] = &FEMplano::jacobianaPlano;
FEMplano::jacobiana[9] = &FEMplano::jacobiana3D;
…
}
```

Como ejemplo, se muestran a continuación en la Figura 6, los algoritmos correspondientes a la implementación de la integración temporal de la ecuación general de movimiento (Klaus-Jürgen Bathe, 1982 y Thomas J. R. Hughes, 1987).

```
// función de creación de la clase. Se inicializan o limpian variables.
Itempo::Itempo() : QFrame (), SystemEc()
{
Itempo::Mconsist = TRUE;
Itempo::damp=0;
Itempo::filas=0;
Itempo::nstep = 1.0;
Itempo::Ki = Itempo::K=NULL;
Itempo::M=NULL;
Itempo::C=NULL;
Itempo::U= Itempo::dU= Itempo::ddU=NULL;
Itempo::Rt=NULL;
Itempo::Rft=NULL;
Itempo::A=0.0; //  (Metodo HHT o metodo alfa) alfa = [-1/3, 0]. Si es =0, se reduce al esquema de Newmark
Itempo::B=0.0;              // beta
Itempo::Y=0.0;              // gama
Itempo::dt=0.0;   // incremento de tiempo
for ( int j=0; j<10; j++ ) Itempo::ca[j] = 0.0;
Itempo::limT = 0.0; // tiempo de analisis
Itempo::tiempo = 0.0;
// Puntero a funciones: indice "damp"
// Matriz de rigideces "efectivas"
Itempo::Kefec[0] = &Itempo::KefecNoDamp;        // Sin amortiguamiento (C nula)
Itempo::Kefec[1] = &Itempo::KefecDamp;
// Vector de cargas "efectivas"
Itempo::Refec[0] = &Itempo::RefecNoDamp;        // Sin amortiguamiento (C nula)
Itempo::Refec[1] = &Itempo::RefecDamp;
}
```

```
// Función de llamada al algoritmo de Integración temporal
bool Itempo::newmark ( DOUBLE * mK, DOUBLE * mM, DOUBLE * mC, DOUBLE *Ft,
                       DOUBLE * u, DOUBLE * du, DOUBLE * ddu,
                       DOUBLE alfa, DOUBLE beta, DOUBLE gama, long int Nstep,
                       DOUBLE deltaTiempo,
                       long int filas )
{
 setnewmark ( mK, mM, mC, Ft, u, du, ddu, alfa, beta, gama, Nstep, deltaTiempo, filas  );
 stepnewmark (); // iniciar análisis dinámico por integración temporal directa
 if( Itempo::K ) delete Itempo::K;
 Itempo::K = NULL;
return TRUE;
}

void Itempo::stepnewmark ()
{
DOUBLE t = dt;

 for ( long int i=1; i <= nstep; i++ ) {

        (this->*Kefec[damp]) (); // calcular las rigideces efectivas
        emit rfnodales ( Rft, t+A*dt, TRUE ); // ...(vector de cargas dinámicas f(t), limpiar)
        (this->*Refec[damp]) (i); // calcular el vector de cargas efectivas
        solve (); // Calcular los desplazamientos 'u' en t+dt
        Ut_dt (i); // calcular du y ddu

        t += dt;  // incrementar el tiempo en dt
} // siguiente incremento de tiempo

return;
}

// retorna un puntero a la zona de memoria asignada al vector de cargas
void Itempo::setnewmark ( DOUBLE * mK, DOUBLE * mM, DOUBLE * mC, DOUBLE *Ft,
                       DOUBLE * u, DOUBLE * du, DOUBLE * ddu,
                       DOUBLE alfa, DOUBLE beta, DOUBLE gama, long int Nstep, DOUBLE
deltaTiempo,
                       long int filas )
{
long int N = filas * (filas+1);  // +1: memoria para almacenar el vector de cargas efectivas

 Itempo::Rft = (Ft+0); // puntero al vector de cargas dinámicas externas
 Itempo::K = new DOUBLE [ N ]; // matriz de rigideces efectivas (+vector de cargas al final) (O DE
MASAS EFECTIVAS en el método alfa)
 Itempo::Rt = (K + (filas*filas)); // vector de cargas efectivas
 for ( long int i=0; i< filas; i++ ) Itempo::Rt[i] = 0.0;

 Itempo::Ki = mK; // matriz de rigidez
 Itempo::M = mM; // matriz de masa consistente
 Itempo::C = mC; // matriz de amortiguamiento
 Itempo::damp = Itempo::C ? 1 : 0; // índice a los punteros a funciones con o sin amortiguamiento
 Itempo::U = u;
 Itempo::dU = du;
 Itempo::ddU = ddu;
 Itempo::filas = filas;
```

```
 Itempo::A = alfa;              // alfa = [-1/3, 0]. Si alfa=0, se reduce al Método de Newmark
 Itempo::B = !beta ? 0.25 * pow(1.0-A, 2.0) : beta;
 Itempo::Y = !gama ? 0.50 * (1.0-2.0*A) : gama;

 // Tiempo
 Itempo::nstep = Nstep;
 Itempo::dt = deltaTiempo;
 Itempo::limT = (DOUBLE)(Nstep) * dt;

 setCintegra (); // Constantes

return;
}

// contantes
void Itempo::setCintegra ()
{
 ca[0] = 1.0 / (B * dt * dt );
 ca[1] = ( 1.0 + A ) * Y / ( B * dt );
 ca[2] = 1.0 / ( B * dt );
 ca[3] = 1.0 / ( 2.0 * B ) - 1.0;
 ca[4] = Y / B * ( 1.0 + A ) - 1.0;
 ca[5] = ( 1.0 + A ) * 0.5 * dt * ( Y / B - 2.0 );
 ca[6] = dt * ( 1.0 - Y );
 ca[7] = Y * dt;
 ca[8] = (1.0 + A);
 ca[9] = A;
}

// calcular la matriz de rigideces "efectivas"
void Itempo::KefecNoDamp () // sin amortiguamiento
{
DOUBLE N = filas * filas;
 for ( long int i=0; i < N; i++ ) K[i] = ca[0] * M[i] + ca[8] * Ki[i];
}

void Itempo::KefecDamp () // con amortiguamiento
{
DOUBLE N = filas * filas;
 for ( long int i=0; i < N; i++ ) K[i] = ca[0] * M[i] + ca[1] * C[i] + ca[8] * Ki[i];
}

// Calcular el vector de cargas efectivas
void Itempo::RefecNoDamp ( long int step ) // en t+dt. Sin amortiguamiento
{
long int ant = filas * (step-1); // anterior

 for ( long int i=0; i< filas; i++ ) {
  Rt[i] = Rft[i];
  for ( long int j=0; j< filas; j++ ) {
            Rt[i] += (*(M+filas*i+j)) * ( ca[0] * (*(U+ant+j)) +
                        ca[2] * (*(dU+ant+j)) + ca[3] * (*(ddU+ant+j)) ) + (*(Ki+filas*i+j)) * ca[9];
  }
 }
return;
}
```

```
void Itempo::RefecDamp ( long int step ) // en t+dt. Con amortiguamiento
{
long int ant = filas * (step-1); // anterior

 for ( long int i=0; i< filas; i++ ) {
  Rt[i] = Rft[i];
  for ( long int j=0; j< filas; j++ ) {
            Rt[i] += (*(M+filas*i+j)) * ( ca[0] * (*(U+ant+j)) + ca[2] * (*(dU+ant+j)) +
                        ca[3] * (*(ddU+ant+j)) ) + (*(C+filas*i+j)) * ( ca[1] * (*(U+ant+j)) +
                        ca[4] * (*(dU+ant+j)) + ca[5] * (*(ddU+ant+j)) ) + (*(Ki+filas*i+j)) * ca[9];
  }
 }
return;
}

// calcular los desplazamientos en el tiempo t+dt y sus derivadas
void Itempo::Ut_dt ( long int step ) // step: 1 a nstep. El índice "0" es para las condiciones iniciales
{
long int ant = filas * (step-1); // anterior
long int ind = filas * step; // actual

 for ( long int i=0; i < filas; i++ ) {
            *(U+(ind+i)) = Rt[i]; // t+dt
            *(ddU+(ind+i)) = ca[0] * ( (*(U+(ind+i))) - (*(U+(ant+i))) ) - ca[2] * (*(dU+(ant+i))) –
                                    ca[3] * (*(ddU+(ant+i))); // t+dt >> (t+dt - t), t, t
            *(dU+(ind+i)) = (*(dU+(ant+i))) + ca[6] * (*(ddU+(ant+i))) + ca[7] * (*(ddU+(ind+i)));
 }
return;
}
```

Figura 6. Implementación de los métodos de Newmark y Hilber-Hughes-Taylor

Como se observa en esta implementación, también se hace uso de punteros a funciones. Si observamos en la función “stepnewmark()”, la orden “**emit**” envía una señal hacia otro objeto o clase. Esta señal es interpretada en tiempo de ejecución pero en el momento de conectarla o activarla, por lo que realmente es veloz y eficiente. Para efectuar la conexión de esta señal, se emplea la función “**connect**”.

```
{...
Itempo itempo;
...
  connect( &itempo, SIGNAL( rfnodales( DOUBLE *, DOUBLE , bool ) ), this,
                                SLOT( Rfnod( DOUBLE *, DOUBLE , bool ) ) );
  itempo.newmark ( RR, MM, NULL, RF, u, du, ddu, alfa, beta, gama, Nstep, deltaTiempo, filas );
  disconnect( &itempo, SIGNAL( rfnodales( DOUBLE *, DOUBLE , bool ) ), this,
                                SLOT( Rfnod( DOUBLE *, DOUBLE , bool ) ) );
... }
```

Figura 7. Conexión de una señal en estructuración con objetos orientados a objetos

En la Figura 7, se muestra como puede efectuarse la conexión de esta señal para este

ejemplo. La función “Rfnod()” será llamada dentro de la clase “ITempo” cuando en alguna parte de esta clase, se emita la señal “rfnodales()” a través de “**emit**”. “Rfnod()”no es conocida en la clase “ITempo”, sino en la clase que solicita a la función “newmark()”, sin embargo devolverá el vector de “cargas dinámicas” para el tiempo t+dt a la clase “ITempo”. Finalmente se desconecta esta señal con “**disconnect**”. Esto permitirá volver a conectar la señal con la misma u otra función apropiada a un determinado caso para la evaluación del vector de “cargas dinámicas”.

De esta misma y sencilla manera podrá evaluarse tiempo a tiempo por ejemplo, la matriz de rigidez para casos no lineales.

4.4 Pos-procesador

Las herramientas de pos-procesamiento y análisis de resultados son las mismas que las empleadas durante el pre-procesamiento. Solo cambia la manera en que se emplean dentro del código, para lo cual se hace uso de los recursos ya descriptos: punteros a funciones y objetos; y objetos orientados a objetos.

Los resultados pueden visualizarse en editor de texto, tablas dinámicas y/o mapeo cromático haciendo uso del CAD (Figura 8) o de tecnología OpenGL (Open Graphics Library) a través de su implementación de código abierto “Mesa 3D” (Figura 9 y Figura 10).

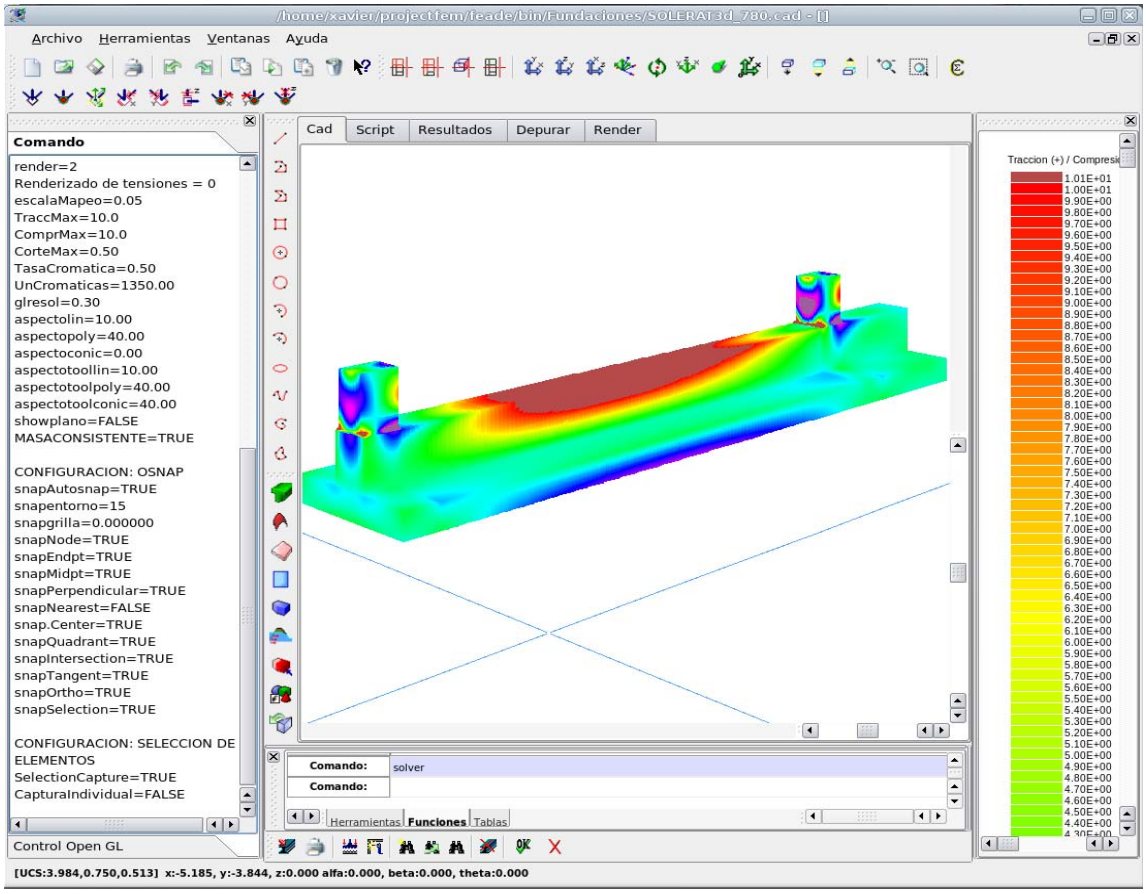

Figura 8. Mapeo cromático en el espacio CAD

La manera de visualización dependerá del tipo de datos o resultado y/o de los requerimientos y necesidades del usuario.

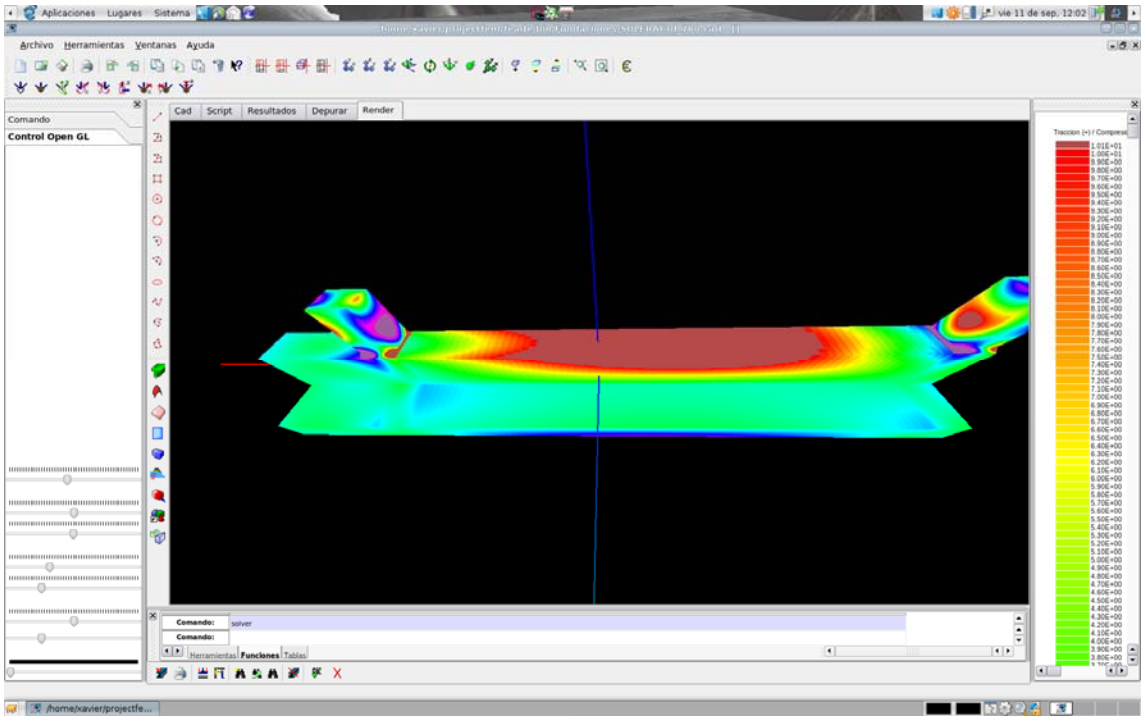

Figura 9. Mapeo cromático con Mesa 3D (OpenGL)

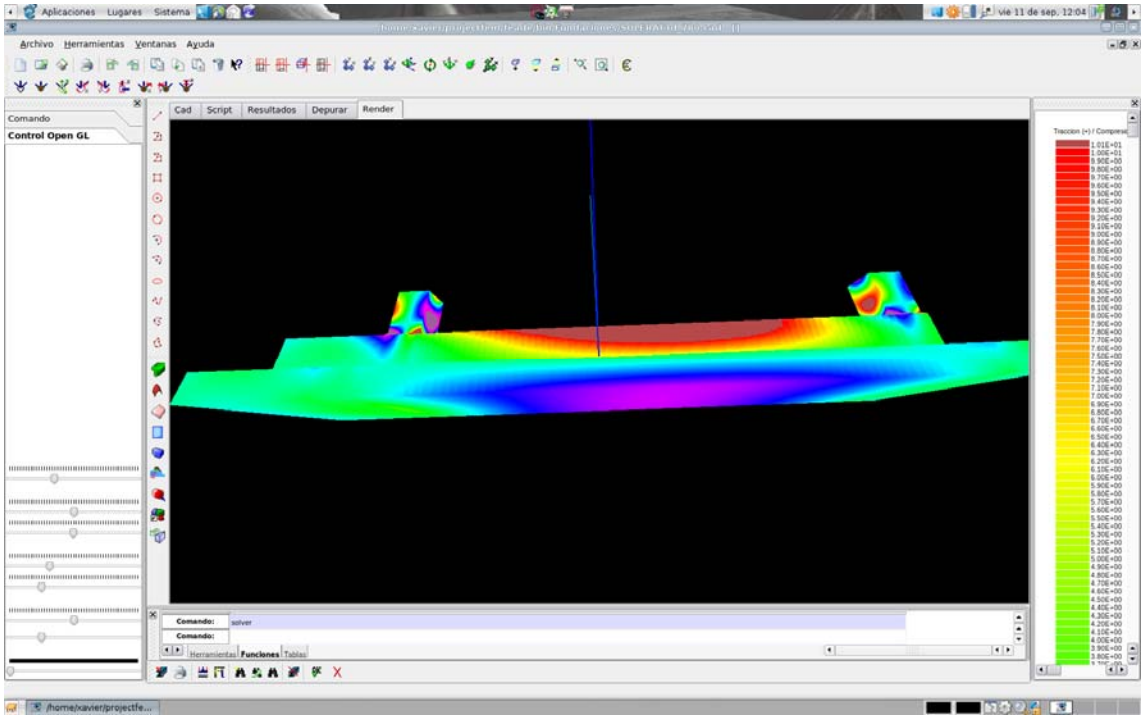

Figura 10. Mapeo cromático con Mesa 3D

Durante el procesamiento, los valores buscados se calculan en los puntos de Gauss para la cuadratura de Gauss-Legendre. Para el mapeo cromático, los elementos finitos de cálculo se subdividen en volúmenes y áreas cuya cantidad dependerá de la precisión deseada de mapeo.

Los valores en el resto del volumen de cada elemento se obtienen por interpolación de los valores en los puntos de Gauss, calculándose en los contornos comunes a elementos, el valor medio (Zienkiewicz O.C., 1982).

5 CONCLUSIONES

El desarrollo de esta aplicación implicó el estudio y aprendizaje en distintas áreas: uso de nuevos sistemas operativos, implementación de técnicas de cálculo numérico, estudio e investigación de distintas familias de elementos finitos, y actualmente el análisis de propiedades y modelos constitutivos de suelos adecuados al objetivo final de esta aplicación, etc.

La aplicación en el estado actual, se emplea de forma didáctica para el aprendizaje de alumnos de grado en distintas materias como: Estabilidad III, Fundaciones, etc, y es empleado por su autor en su trabajo profesional.

Esperamos que estas últimas etapas del proceso de desarrollo que se inician, sean tan fructíferas para los involucrados y alumnos como lo ha sido hasta el presente.

AGRADECIMIENTOS

Los autores agradecen a la Universidad Católica de Salta, por el apoyo brindado para este proyecto y en especial a la Dra. Alicia Pérez.

REFERENCIAS

Zienkiewicz, O.C., *El método de los elementos finitos*. Editorial Reverté, S. A., 1982.

Gottfried Byron S., *Programación en C*. McGraw Hill, 1991.

Herbert Schildt, *Turbo C/C++. Manual de referencia*. Osborne/McGraw Hill, 1992.

Klaus-Jürgen Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Inc., 1982. *Englewood Cliffs, New Jersey 07632*.

Thomas J. R. Hughes, *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis*, Prentice-Hall, Inc., 1987. *Englewood Cliffs, New Jersey 07632*.

Barbat Alex H. y Canet Juan M., *Estructuras sometidas a acciones sísmicas*. Segunda Edición. CIMNE, Barcelona, 1994.