MÉTODOS COMPUTACIONAIS EM ENGENHARIA
Lisboa, 31 de Maio - 2 de Junho, 2004

IMPLEMENTACIÓN DE UN ALGORITMO DE REFINAMIENTO/DESREFINAMIENTO PARA MALLAS DE TETRAEDROS

J.M. González-Yuste [*], R. Montenegro, J.M. Escobar, G. Montero, y E. Rodríguez

[*] Instituto Universitario de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería
Universidad de Las Palmas de Gran Canaria
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria
e-mail: jmgonzalezy@pas.ulpgc.es, es, web http://www.iusiani.ulpgc.es

Palabras clave: Elementos Finitos, Mallas Adaptativas, Refinamiento, Desrefinamiento, Programación Orientada a Objetos, Estructuras de Datos.

Resumen. *Para resolver problemas formulados en EDP mediante el método de elementos finitos es necesario en primer lugar establecer una buena discretización del dominio. Una vez definida una malla base que aproxima su geometría, ésta debe adaptarse a las singularidades de la solución numérica. Utilizando técnicas de refinamiento/desrefinamiento que introducen o eliminan elementos en zonas del dominio, podemos construir una partición que determine la solución numérica con una precisión deseada. Estas técnicas son especialmente útiles en problemas evolutivos donde la posición de las singularidades de la solución varían con el tiempo. Este trabajo presenta una implementación en C++ de un algoritmo de refinamiento/desrefinamiento para mallas de tetraedros. La elección del lenguaje se ha realizado atendiendo a las facilidades que ofrece para la gestión de memoria y modelización de entidades mediante las clases. El algoritmo de refinamiento está basado en la subdivisión en 8 subtetraedros, ver [1]. Se ha elegido este algoritmo por su sencillez y aplicabilidad, así como porque el algoritmo de desrefinamiento se puede definir directamente como su inverso. Presentamos también pseudocódigos de implementación de los algoritmos, así como diferentes propuestas de aplicación.*

1. INTRODUCCIÓN

En la actualidad, la mayor parte de los programas que utilizan el método de elementos finitos se apoya en técnicas adaptables basadas en una estimación del error cometido con nuestra solución numérica, o al menos en indicadores de error fiables que nos señalen los elementos que deben ser refinados o desrefinados en la malla.

En generación de mallas adaptables podemos considerar dos aspectos diferentes: la discretización del dominio atendiendo a su geometría o a la solución numérica. Existen muchas formas de abordar estos aspectos. La primera cuestión es: ¿mallas estructuradas o no estructuradas?. En este sentido, está claro que el uso de mallas no estructuradas nos proporciona más flexibilidad a la hora de mallar geometrías complejas utilizando un número óptimo de nodos. En este caso, los métodos más clásicos para la obtención de triangulaciones tridimensionales se basan fundamentalmente en algoritmos de avance frontal [6] o en algoritmos basados en la triangulación de Delaunay [9] y [7]. Una vez que se ha discretizado la geometría del dominio, la malla debe adaptarse atendiendo a las singularidades de la solución numérica. Este proceso implica la introducción (refinamiento) o eliminación (desrefinamiento) de nodos de la malla actual. Los cambios pueden afectar a la malla actual de forma local o global, dependiendo del método de triangulación elegido. Diferentes estrategias de refinamiento han sido desarrolladas para triangulaciones en 2-D, y han sido generalizadas a 3-D. Si se ha optado por un refinamiento que afecte localmente a la malla actual, cabe plantease otra cuestión: ¿mallas encajadas o no encajadas?. La respuesta en este caso no es tan clara. El uso de mallas encajadas tiene varias ventajas importantes. Podemos conseguir familias de secuencias de mallas encajadas en un mínimo tiempo de CPU. Además, se puede aplicar más fácilmente el método multimalla para resolver el sistema de ecuaciones asociado al problema. Por otra parte, se puede controlar automáticamente la suavidad y la degeneración de la malla, y el mantenimiento de las superficies definidas en el dominio, en función de las características de la malla inicial. Si el dominio posee una geometría compleja, un buen modo de proceder es obtener la malla inicial empleando un generador de mallas no estructuradas y, posteriormente, aplicar una técnica de refinamiento y desrefinamiento local de mallas encajadas atendiendo a un indicador de error apropiado al problema. Además, si tratamos de resolver un problema evolutivo, podemos aproximar automáticamente cualquier solución inicial definida en el dominio. Con la técnica de refinamiento y desrefinamiento conseguimos un óptimo soporte de interpolación a trozos capaz de aproximar esta solución con la precisión deseada. En general, podría aplicarse esta técnica para cualquier función definida en el dominio de forma discreta o analítica.

Con estas ideas, anteriormente se desarrollaron técnicas adaptables en 2-D obteniendo buenos resultados en diferentes problemas estacionarios y evolutivos, véase por ejemplo [8], [10], [14]. En estos trabajos se utilizó una versión del algoritmo de refinamiento local 4-T de Rivara [12]; todos los triángulos que deben ser refinados, atendiendo al indicador de error, se dividen en cuatro subtriángulos mediante la introducción de un nuevo nodo en los centros de sus lados y uniendo el nodo introducido en el lado mayor con el vértice opuesto y los otros dos nuevos nodos. La elección particular del algoritmo de refinamiento es muy importante, puesto que el

algoritmo de desrefinamiento puede entenderse como el inverso del algoritmo de refinamiento. El algoritmo de refinamiento 4-T de Rivara posee buenas propiedades en cuanto a la suavidad y degeneración de la malla. Además de esto, el número de posibilidades que aparecen en la relación entre un elemento padre y sus hijos es menor que con otros algoritmos de refinamiento en 2-D, tras asegurar la conformidad de la malla. Por ejemplo, sería más complicado desarrollar un algoritmo de desrefinamiento, acoplado con el algoritmo de refinamiento local propuesto en [3]; todos los triángulos que deben ser refinados, atendiendo al indicador de error, se dividen en cuatro subtriángulos mediante la introducción de un nuevo nodo en los centros de sus lados y uniéndolos entre sí.

En 3-D, el problema es diferente. Aunque parezca paradógico, la extensión de un algoritmo adaptable que sea más simple que otro en 2-D, no tiene porqué ser también más simple en 3-D. Así, entre los algoritmos de refinamiento desarrollados en 3-D podemos mencionar los que se basan en la bisección del tetraedro [15], [13], [11], y los que utilizan la subdivisión en 8-subtetraedros [4], [5], [2]. En concreto, el algoritmo desarrollado en [11] se puede entender como la generalización a 3-D del algoritmo 4-T de Rivara, que a su vez está basado en la bisección del triángulo por su lado mayor. El problema que se produce en esta extensión a 3-D es el gran número de casos posibles en los que puede quedar dividido un tetraedro, respetando las diferentes posibilidades de la división 4-T en sus cuatro caras, durante el proceso de conformidad de la malla. Sin embargo, los algoritmos analizados en [4], [5], [2], que a su vez generalizan a 3-D la partición en cuatro subtriángulos propuesta en [3], son más sencillos debido a que el número de particiones posibles de un tetraedro es mucho menor que en el caso de la generalización del algoritmo 4-T. Por otra parte, puesto que la calidad de la malla está asegurada en todos estos casos, hemos optado por implementar una versión del algoritmo que utiliza la subdivisión en 8-subtetraedros, y que será presentada en la segunda sección de este trabajo. En la tercera sección se comenta el algoritmo de desrefinamiento asociado al refinamiento propuesto. En la cuarta sección veremos diversos aspectos de la elección del lenguaje de programación y las estructuras de datos. En las secciones quinta y sexta se muestra como han sido implementados los algoritmos. Finalmente veremos una aplicación del algoritmo y lineas futuras de investigación enmarcadas en la generación de mallas no estructuradas para el método de elementos finitos. La aplicación del algoritmo de refinamiento se realiza sobre sobre mallas tridimensionales generadas a partir de una versión del método de triangulación de Delaunay introducida en [7]. Este método de triangulación es bastante popular debido a la calidad de las mallas que produce. Sin embargo, este método presenta serios problemas, especialmente en 3-D, debidos a los errores de redondeo que resultan en el ordenador y que se acentúan por problemas inherentes al propio método de triangulación. En [7] se presenta un procedimiento para construir una triangulación tridimensional cercana a la triangulación de Delaunay que resuelve estos problemas. Por otra parte, puesto que la triangulación de Delaunay se construye a partir de una nube de puntos situados en la frontera e interior del dominio, puede suceder que la malla resultante no contenga todas las aristas y caras principales definidas en la frontera o interior dominio. Este aspecto constituye un problema abierto y ha sido estudiado por numerosos autores, aunque las soluciones proporcionadas son complejas.

2. ALGORITMO DE REFINAMIENTO

En esta sección presentaremos el algoritmo de refinamiento local que hemos desarrollado a partir de la subdivisión en 8-subtetraedros introducida en [2]. Consideremos una triangulación inicial τ_0 del dominio formada por un conjunto de n_0 tetraedros $t_1^0, t_2^0, ..., t_{n_0}^0$. Nuestro objetivo es construir una secuencia de $m+1$ niveles de mallas encajadas $T = \{\tau_0 < \tau_1 < ... < \tau_m\}$, tal que el nivel τ_{j+1} se obtiene mediante un refinamiento local del nivel anterior τ_j. Si suponemos conocido el indicador de error η_i^j asociado al elemento $t_i^j \in \tau_j$, decidimos que este elemento debe ser refinado si $\eta_i^j \geq \gamma \eta_{\text{máx}}^j$, siendo $\gamma \in [0,1]$ el parámetro de refinamiento y $\eta_{\text{máx}}^j$ el máximo valor de los indicadores de error de los elementos de τ_j. Desde un punto de vista constructivo, plantearemos inicialmente la obtención de τ_1 partiendo de la malla base τ_0, atendiendo a las siguientes consideraciones:

a) *Subdivisión en 8-subtetraedros*. Decimos que $t_i^0 \in \tau_0$ es un tetraedro de *tipo I* si se verifica que $\eta_i^0 \geq \gamma \eta_{\text{máx}}^0$. Este conjunto de tetraedros serán posteriormente subdivididos en 8 subtetraedros según la figura 1(a); se introducen 6 nuevos nodos en el punto medio de sus aristas y se subdividen cada una de sus cuatro caras en cuatro subtriángulos según la división propuesta por Bank [3], cuatro subtetraedros quedan determinados a partir de los cuatro vértices de t_i^0 y las nuevas aristas, y los otros cuatro subtetraedros se obtienen al unir los dos vértices opuestos más cercanos del octoedro que resulta en el interior de t_i^0. Esta sencilla estrategia es la que se propone en [2] y [4], frente a otras basadas en transformaciones afines a un tetraedro de referencia, como la analizada en [5], que aseguran la calidad de los tetraedros resultantes. Bornemann et al. [4] afirma que con ambas estrategias obtiene resultados comparables en sus experimentos numéricos, y Löhner y Baum [2] asegura que nuestra elección produce el menor número de tetraedros distorsionados en la malla refinada. Evidentemente, siempre se podría determinar la mejor de las tres opciones existentes para la subdivisión del octoedro interior, mediante el analisis de la calidad de sus cuatro subtetraedros, pero esto aumentaría el coste computacional del algoritmo.

Una vez que se ha definido el tipo de partición de los tetraedros *tipo I*, de cara a asegurar la conformidad de la malla, nos podemos encontrar con tetredros vecinos que pueden tener 6, 5,..., 1 ó 0 nuevos nodos introducidos en sus aristas. Analizamos a continuación cada uno de estos casos. Hay que tener en cuenta que en todo este proceso únicamente estamos marcando las aristas de los tetraedros de τ_0 en las que se ha introducido un nuevo nodo, y en base al número de aristas marcadas se clasifica el correspondiente tetraedro. Es decir, hasta que no está asegurada la conformidad de τ_1, simplemente marcando aristas, no se procederá a la definición de esta nueva malla.

b) *Tetraedros con 6 nuevos nodos*. Aquellos tetraedros que por razones de conformidad tengan marcadas sus 6 aristas pasan automáticamente al conjunto de tetredros *tipo I*.

c) *Tetraedros con 5 nuevos nodos*. Aquellos tetraedros que tengan 5 aristas marcadas pasan también al conjunto de tetraedros *tipo I*. Previamente, habrá que marcar la arista en la que no había sido introducido ningún nuevo nodo.

d) *Tetraedros con 4 nuevos nodos*. En este caso, se marcan las dos aristas restantes y pasa a

ser considerado de *tipo I*.

Hay que tener en cuenta que al proceder de la forma indicada en (b), (c) y (d) debido al refinamiento global considerado en (a) atendiendo al indicador de error, mejoramos la calidad de la malla y simplificamos considerablemente el algoritmo. Puede criticarse que este procedimiento puede aumentar la zona refinada, pero hay que tener en cuenta que sólo se introducirán 1 ó 2 nuevos nodos sobre un total de 6. Obsérvese que esta proporción es igual o inferior a la que surge en el refinamiento en 2-D del algoritmo 4-T de Rivara para un triangulo, en el que la probabilidad de encontrarnos con un nodo introducido en el lado mayor es $1/3$. Este fenómeno se acentúa en el algoritmo propuesto como su generalización a 3-D.

e) *Tetraedros con 3 nuevos nodos*. En este caso, hay que distinguir dos situaciones:

e.1) Si las correspondientes 3 aristas marcadas no están sobre la misma cara, entonces se marcan las 3 restantes y el tetraedro se introduce en el conjunto de tetraedros *tipo I*. Aquí podemos hacer la consideración antes mencionada, al comparar este paso con otros algoritmos basados en bisección por el lado mayor.

En los siguientes casos, ya no marcaremos ninguna nueva arista, lo cual implica que no se introducirá ningún nuevo nodo en el tetraedro que se pretende hacer conforme. Se procederá a subdividirlos de la forma que se indica a continuación, creando subtetraedros que llamaremos *transitorios*, ya que podrán desaparecer en posteriores etapas de refinamiento para asegurar que la malla no degenere.

e.2) Si las 3 aristas marcadas están sobre la misma cara del tetraedro, entonces se crearán 4-subtetradros transitorios como se muestra en la figura 1(b); se definen nuevas aristas uniendo entre sí los tres nuevos nodos y conectando éstos con el vértice opuesto a la cara que los contiene. Los tetraedros de τ_0 con estas características se englobarán en el conjunto de tetraedros de *tipo II*.

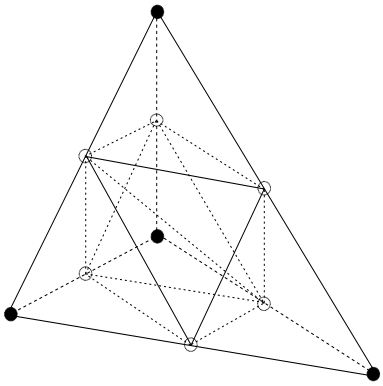
f) *Tetraedros con 2 nuevos nodos*. También distinguiremos dos situaciones:

f.1) Si las dos aristas marcadas no están sobre la misma cara, entonces se construirán 4-subtetraedros transitorios como se presenta en la figura 1(c), definidos a partir de las aristas que conectan los dos nuevos nodos y a éstos con los vértices opuestos de las dos caras que los contienen. Los tetraedros que se encuentren en esta situación se denominan de *tipo III.a.*

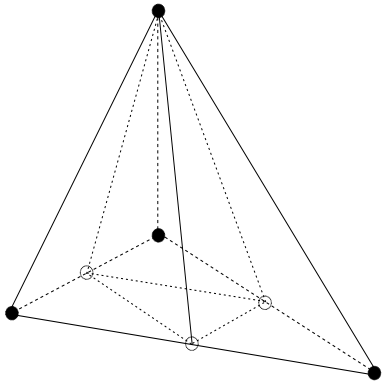
f.2) Si las dos aristas marcadas están sobre las misma cara, entonces se crearán 3 subtetraedros transitorios según se expone en la figura 1(d); se divide en tres subtriángulos la cara definida por las dos aristas marcadas, conectando el nuevo nodo situado en la arista mayor de éstas dos con el vértice opuesto y con el otro nuevo nodo, tal que estos tres subtriángulos y el vértice opuesto a la cara que los contiene definen los tres nuevos subtetraedros. Se destaca que de las dos posibles eleciones, se toma como referencia la mayor arista marcada para aprovechar en algunos casos las propiedades de la bisección por el lado mayor. Los tetraedros que se encuentren en esta situación se denominan de *tipo III.b.*

g) *Tetraedros con 1 nuevo nodo*. Se crearán dos subtetraedros transitorios según la figura 1(e); se une el nuevo nodo con los otros dos que no pertenecen a la correspondiente arista marcada. Este conjunto de tetraedros se denomina de *tipo IV*.

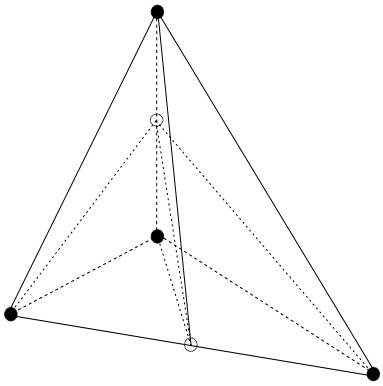
h) *Tetraedros con 0 nuevos nodos*. Estos tetraedros de τ_0 no se dividen y serán heredados a



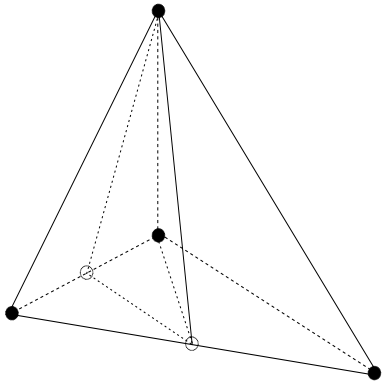
(a) Tipo I



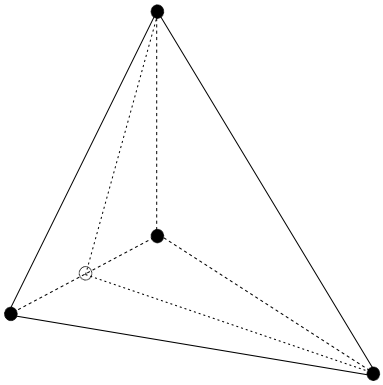
(b) Tipo II



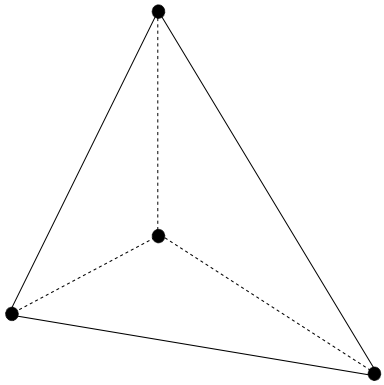
(c) Tipo III.a



(d) Tipo III.b



(e) Tipo IV



(f) Tipo V

Figura 1. Clasificación de las subdivisiones de un tetraedro en función de los nuevos nodos indicados con círculo blanco.

la malla refinada τ_1. Los denominaremos de *tipo V* y se representan en la figura 1(f).

Este proceso de clasificación de los tetraedros de τ_0 se realiza simplemente marcando sus aristas. La conformidad de la malla se va asegurando por vecindad entre los tetraedros que contienen una nueva arista marcada. Esto hace que, al finalizar el recorrido de este subconjunto de tetraedros de *tipo I*, la malla resultante sea conforme y refinada localmente. Además, el proceso resulta de un bajo coste computacional, ya que el proceso de estudio local finaliza cuando nos encontramos con tetraedros en los que no se tiene que marcar ninguna nueva arista.

En general, cuando queremos refinar el nivel τ_j en el que ya existen tetraedros *transitorios* se procederá de igual forma que en el paso de τ_0 a τ_1, con la siguiente variante: desde el momento en que se tenga que marcar una arista de un tetraedro transitorio, bien porque tenga que ser refinado atendiendo al indicador de error o bien por razones de conformidad, entonces se eliminan (proceso de borrado) todos los tetraedros transitorios "hijos" de su tetraedro "padre", marcamos todas las aristas de éste "padre" y se introduce en el conjunto de tetraedros de *tipo I*.

3. ALGORITMO DE DESREFINAMIENTO

Presentamos ahora el algoritmo de desrefinamiento planteado como el inverso del algoritmo de refinamiento propuesto. Partiendo de la secuencia de mallas encajadas $T = \{\tau_0 < \tau_1 < \cdots < \tau_m\}$ obtenidas mediante refinamientos sucesivos, se pretende suprimir elementos desde los niveles más profundos de la malla hacia niveles superiores. Así como el refinamiento se basaba en el indicador de error η_i^j asociado al tetraedro $t_i^j \in \tau_j$, una opción para el desrefinamiento planteada en [8] consiste en estudiar la solución numérica en los nodos de la malla. Si un nodo $n_p^{j-1} \in \tau_{j-1}$, al ser una secuencia de mallas encajadas, los nodos pasarán a la siguiente malla τ_j, siendo renombrados como n_p^j. Para determinar si un nodo $n_i^j \in \tau_j$ puede ser eliminado se va a comparar el valor de la solución numérica v_i^j en dicho nodo con la solución interpolada en los nodos n_p^j y n_q^j. Estos nodos componen la arista de τ_{j-1} sobre la que se introdujo n_i^j en el proceso de refinamiento. Diremos que n_i^j es un candidato a ser eliminado si:

$$\left| v_i^j - \frac{v_p^j + v_q^j}{2} \right| < \epsilon \tag{1}$$

El parámetro de desrefinamiento ϵ determina la precisión que desea a la hora fijar la solución en los nodos.

Para estudiar los nodos de la malla se va comenzar por los introducidos en los niveles más profundos. Para que un nodo pueda ser eliminado se debe estudiar todos los tetraedros que comparten la arista de τ_{j-1} delimitada por n_p^j y n_q^j atendiendo a como ha sido dividido cada uno de ellos. Tendremos dos casos:

a) *Tipo I*. Debe tener marcados para desrefinar un número de nodos tal que a la hora de conformar la malla este tetraedro no vuelva a ser marcado como *Tipo I*. Si no fuera el caso y tuvieran que volver a introducirse nodos por conformidad y el tetraedro pasara a *Tipo I* permanecerían todos los nodos candidatos a desrefinar que han sido introducidos en las aristas del tetraedro.

b) *Resto de tipos*. Puesto que siempre tendremos una división adecuada, se puede eliminar cualquier nodo introducido en las aristas de estos tetraedros.

Una vez que se tengan exactamente establecidos los nodos a borrar se procede a la destrucción de los subtetraedros que han generado los nodos introducidos en las aristas del tetraedro padre. El proceso finaliza con la nueva división de los elementos en función de las aristas que hayan quedado marcadas. Tendremos, por lo tanto, un nivel de malla desrefinado y conforme según los criterios del algoritmo, y se podría comenzar a estudiar el nivel inmediatamente superior.

4. PROGRAMACIÓN Y ESTRUCTURAS DE DATOS

La implementación de los algoritmos ha sido realizada en el lenguaje C++. Este lenguaje, aparte de su características propias, incluye todas las ventajas de la programación en C como la gestión dinámica de la memoria, tratamiento de punteros y alta modularidad. C++ añade la programación orientada a objetos. En C existen las *estructuras*, capaces de agrupar información de distinto tipo para un determinado elemento. C++ extiende el concepto de *estructura* al de *clase*. En una clase, además de los datos propios del elemento se pueden especificar que operaciones se pueden realizar con él. Se define, por lo tanto, un tipo de dato.

Las clases nos ofrecen más posibilidades, como el *encapsulamiento* y la *herencia*. Con el *encapsulamiento* se define en cada clase que partes son accesibles al exterior y cuales son restringidas. Esto permite reducir errores en la programación impidiendo accesos a zonas restringidas de la *clase* en donde estarán codificadas sus operaciones fundamentales. La *herencia* consiste en la definición de clases basadas en otras, de manera que las nuevas heredarán todas las características y propiedades de su antecesora. Se puede definir una *jerarquía de clases* partiendo de módulos sencillos hasta llegar a otros mucho más complejos.

Con estas herramientas se ha diseñado una jerarquía para modelizar los diferentes elementos que componen una malla, así como una serie de utilidades para trabajar con ellas. En la figura 2 se puede apreciar la jerarquía definida. En primer lugar se describirán las clases auxiliares. A continuación las clases relacionadas con los elementos de la malla. Finalmente las relacionadas con la solución de problemas.

a) Clases auxiliares. Se ha una clase llamada *Vector* que contiene un array de punteros con ciertas peculiaridades. Principalmente se trata de que no va a contener duplicados. Las operaciones principales consisten en la inserción, borrado y localización de elementos dentro del array. En esta clase se lleva toda gestión de la memoria, y el resto de clases la utilizarán para mantener referencias a otros objetos.

De *Vector* se ha heredado la clase *VecIter*. Recoge todas sus características, y añade operaciones de recorrido y acceso rápido sobre los elementos.

b)Punto. En esta define las propiedades básicas en un punto en el espacio. Contiene sus coordenadas (x,y,z) y operaciones de cálculo de módulo y distancia entre dos puntos.

c) Elemento. Es la clase base para todos los elementos de la malla. Es muy simple y guarda dos propiedades: *Referencia* y *Nivel*. La primera contiene un valor propio para el problema que se trata de resolver y la segunda indica la profundidad a la que se encuentra el elemento en la

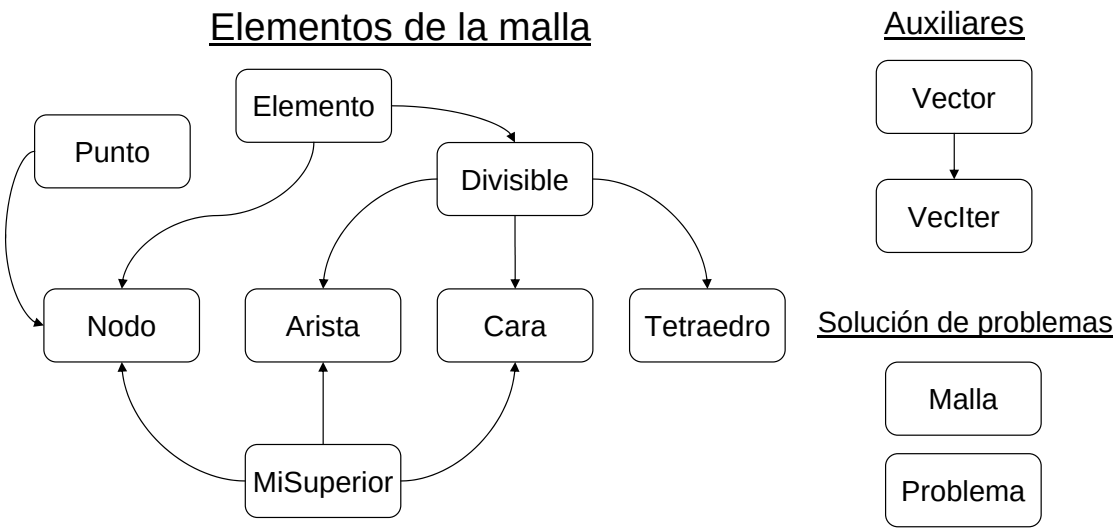


Figura 2. Jerarquía de clases

malla , siendo el valor cero para los elementos de la malla inicial.

d) Divisible. Esta clase hereda de *Elemento* y es la antecesora de todas las que modelan objetos de la malla que puedan dividirse: aristas, caras y tetraedros. Mantiene las referencias genealógicas entre un elemento padre y sus hijos.

e) MiSuperior. Para un elemento dado esta clase contiene referencias a los objetos que compone: aristas que comparten un nodo, caras que comparten una arista y tetraedros que comparten una cara.

f) Nodo. Clase heredada de *Punto*, *Elemento* y *MiSuperior*. Contiene todas la propiedades y operaciones que se pueden realizar con un nodo de la malla.

g) Arista. Clase heredada de *Divisible* y *MiSuperior*. Además de las características propias de una arista, se guardan referencias de marcas recibidas para ser refinada, nodo que divide la arista y operaciones para el cálculo de su longitud.

h) Cara. Al igual que la clase *Arista* se hereda de *Divisible* y *MiSuperior*. En esta clase se guardan también referencias a arista generadas interiormente en su proceso de división.

i) Tetraedro. Solo herea de *Divisible*. Al igual que la clase *Cara* se almacenar referencias a aristas y caras que se puedan generar internamente a la hora de dividirla. También va a contener algunas propiedades como un indicador de refinamiento, de transitorio y otros dependientes del

problema, tal como el valor de la solución.

j) Clases orientadas a la solución de problemas. Se han definido dos clases directamente relacionadas con la resolución de problemas mediante refinamiento/desrefinamiento. Una es la clase *Malla*, la cual contiene una lista de referencias a nodos, aristas, caras y tetraedros. En esta clase se han implententados los algoritmos de refinamiento y desrefinamiento, así como los procesos de división de elementos. La otra clase es la clase *Problema*. Esta clase contiene los elementos de la malla inicial que pasará como referencias a la clase *Malla* para que realice las operaciones necesarias. Puesto que cada elemento mantiene referencias genealógicas sobre los hijos en los que ha sido dividido se forma una estructura de malla encajada. La clase *Malla* informará a *Problema* de cuáles son los elementos que componen la malla final refinada o desrefinada. Se han programado también diferentes formatos de intercambio de información con módulos de resolución de problemas, de manera que se obtenga fácilmente los datos necesarios para llevar a cabo un refinamiento o desrefinamiento.

5. IMPLEMENTACIÓN DEL ALGORITMO DE REFINAMIENTO

Apoyándonos en las estructuras de datos mencionadas se va a crear una lista con todos los tetraedros de la malla que deben ser revisados. Cuando un tetraedro es estudiado se elimina de la lista. Según sea un tetraedro transitorio o no podemos encontrarnos los siguientes casos:

- Tetraedro permanentes (no transitorios). Si el indicador de error establece que el tetraedro debe refinarse o por conformidad tendrá que dividirse como *Tipo I* se marcan todas sus aristas que no estuvieran previamente marcadas. Además, se añade a la lista de tetraedros pendientes de estudiar todos aquellos que comparten las aristas se han marcado.
- Tetraedros transitorios. Si tuviera que refinarse por su indicador de error o hubiera recibido una marca debido que un vecino suyo deba refinarse, entonces:

  1. Se elimina de la lista todos los tetraedros “hermanos” del tetraedro en estudio.
  2. Se marcan todas las aristas del tetraedro padre que no estuvieran marcadas.
  3. Se procede a la división del tetraedro padre como *Tipo I*.
  4. Los tetraedros resultantes de la división se añaden a la lista de tetraedros pendientes de estudiar

Al finaliza el proceso tendremos una malla marcada y conforme según el algoritmo propuesto. A continuación se puede proceder a la división de los elementos de la malla. Un código ilustativo sería el que se puede ver en la figura 3.

El proceso de eliminación de transitorios no sólo afecta al tetraedro de estudio sino a todos los vecinos por cada una de sus caras. Al introducir nuevas marcas en el tetraedro una vez eliminada su división interna, se recorrerán todos sus vecinos por caras, excepto por aquella que esté dividida en cuatro, puesto que ésta queda permanente al no poder recibir más marcas. Siempre que se vaya a eliminar una división de una cara se lanza un proceso recursivo sobre los tetraedros que la comparten para eliminar sus divisiones internas. Hay que destacar que ésto

```
Refinar(Malla τ_i)
  Generar lista l_i con tetraedros de τ_i
  Mientras haya tetraedros en l_i
    Tomar un t_j y eliminarlo de l_i
    Si t_j es transitorio
      Si t_j es refinable o está marcado
        Obtener t_p padre de t_j
        Eliminar de l_i los hijos de t_p
        DeshacerTransitorios(t_p)
        MarcarAristas(t_p)
        Dividir t_p como Tipo I
        Añadir hijos de t_p a l_i
    si no
      Si t_j es Tipo I
        MarcarAristas(t_j)

MarcarAristas(t_i)
  Para cada a_j arista no marcada de t_i
    Poner marca en a_j
    Añadir a l_i todos los tetraedros que comparten a_j

DeshacerTransitorios(t_k)
  Si t_i no dividido retornar
  Eliminar división interna de t_k
  Para cada cara c_j de t_k
    DeshacerDivisionCaras(c_j)
  Añadir t_k a la l_i

DeshacerDivisionCaras(c_i)
  Si c_i dividida en 4 retornar
  Para cada tetraedro t_j que comparte c_i
    DeshacerTransitorios(t_j)
  Eliminar división de c_i
```

Figura 3. Código del refinamiento

solo se realizará sobre vecinos que nunca estarán divididos como *Tipo I* o *Tipo II*, puesto que el criterio de parada, la división de una cara en cuatro, solo la tienen los tipos de tetraedros mencionados.

Finalmente destacar que la propagación del refinamiento se realiza a partir de las aristas no marcadas, mientras que la propagación de la eliminación de transitorios se hace a partir de caras marcadas de forma no permanente.

6. IMPLEMENTACIÓN DEL ALGORITMO DE DESREFINAMIENTO

La implementación del desrefinamiento se va a basar en el estudio de los nodos de la malla. Inicialmente se marcan todos los nodos como desrefinables, excepto aquellos de la malla inicial, puesto que nunca van a ser eliminados. A continuación se evaluará la condición de desrefinamiento para cada nodo, y se desmarcarán aquellos que no la cumplan. Cada vez que se desmarca un nodo se lanza un proceso recursivo por el que se van a desmarcar los nodos de la arista sobre la que se ha introdcido el nodo. El pseudocódigo asociado se puede ver en la figura 4.

Una vez concluido el recorrido se generan unas listas de nodos marcados para desrefinar

```
PropagarNoDesrefinable(Nodo n_i)
  Desmarcar n_i
  Localizar n_p y n_q, nodos que componen la arista que divide n_i
  Si n_p es desrefinable entonces PropagarNoDesrefinable(n_p)
  Si n_q es desrefinable entonces PropagarNoDesrefinable(n_q)
```

Figura 4. Código de la propagación del desrefinamiento

según el nivel de profundidad de la malla en el que se encuentre cada nodo. El siguiente proceso va a determinar qué nodos pueden ser realmente eliminados y se realizará sobre las listas de los niveles más profundos hacia los más superficiales. Para cada nodo de la lista se estudian los tetraedros que comparten la arista que el nodo divide, y tendremos dos casos:

- Si el tetraedro ha sido dividido en transitorios no se realizará ninguna acción. Si el nodo es finalmente eliminado siempre se podrá rehacer la división de este tetraedro, puesto que pasaría de ser *Tipo II*, *Tipo III* o *Tipo IV* a *Tipo III*, *Tipo IV* o *Tipo V*.
- Si el tetraedro ha sido dividido en 8 (*Tipo I*) se estudia si al eliminar todos los nodos candidatos a ser desrefinados pasaría a ser de otro tipo. Si no es así, no se va a eliminar el nodo, y se lanzaría el proceso `PropagarNoDesrefinable` para cada nodo candidato que haya en las aristas de este tetraedro.

Una vez concluido el recorrido por la lista se lanzaría el proceso de refinamiento para rehacer todas las divisiones de los tetraedros que hayan perdido sus divisiones internas. La malla quedaría nuevamente conformada y se pasaría a estudiar el nivel inmediatamente anterior. El código completo quedaría como muestra la figura 5

```
Desrefinar(Malla τ_i)
  Marcar todos los nodos de τ_i como desrefinables
  Para cada nodo n_j de τ_i
    Si n_j no cumple criterio de desrefinamiento
      PropagarNoDesrefinable(n_j)
  Defimos m como el nivel más profundo donde hay un nodo marcado para desrefinar
  Para k desde m hasta 1
    Generar l_k, una lista con los nodos del nivel k desrefinables
    Mientras fin = FALSE
      fin = TRUE
      Para cada nodo n_j en l_k
        Para cada tetraedro t_q que comparte la arista que divide n_j
           Si t_q dividido en 8
             Si t_q fuera Tipo I con sus nodos no-desrefinables
               Eliminar n_j de l_k
               PropagarNoDesrefinable(n_j)
               fin = FALSE
// aquí quedaría en l_k los nodos definitivamente eliminables
    Para cada nodo n_j en l_k
      Eliminar tetraedros que comparten la arista que divide n_j
    Refinar nivel de malla k
```

Figura 5. Código básico del desrefinamiento

7. APLICACIONES Y LÍNEAS FUTURAS

Para obtener resultados de la implementación se ha empleado una geometría consistente en un cubo con una gaussiana en una de sus caras, la malla τ_0. Se ha definido una serie de puntos sobre la superficie de la curva y para obtener la siguiente τ_i se ha realizado un proceso de tres refinamientos locales y un desrefinamiento en torno al cada punto. Se puede observar en la figura 6 como se va desplazando la zona refinada a lo largo de la base de la geometría.

En próximos trabajos se pretende aplicar estos algoritmos en problemas evolutivos, especiamente sobre temas medioambientales para la difusión de contaminantes.

AGRADECIMIENTOS

Este trabajo ha sido desarrollado en el marco del proyecto REN2001-0925-C03-02/CLI, subvencionado por el Ministerio de Ciencia y Tecnología y FEDER.

REFERENCIAS

[1] J.M. González-Yuste, R. Montenegro, J.M. Escobar, G. Montero and E. Rodríguez, *Local Refinement Triangulations Using Object-Oriented Methods,* Advances in Engineering Software, aceptado (2004).

[2] R. Löhner y J.D. Baum, *Adaptive h-refinement on 3D unstructured grids for transient problems*, Int. J. Num. Meth. Fluids, **14**, 1407-1419 (1992).

[3] R.E. Bank, A.H. Sherman y A. Weiser, *Refinement algorithms and data structures for regular local mesh refinement*, in Scientific Computing IMACS, Amsterdam, North-Holland, 3-17 (1983).

[4] F. Bornemann, B. Erdmann y R. Kornhuber, *Adaptive multilevel methods in three space dimensions*, Int. J. Numer. Meth. Eng., **36**, 3187-3203, (1993).

[5] A. Liu y B. Joe, *Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision*, Mathematics of Comput., **65**, 215, 1183-1200 (1996).

[6] R. Löhner y P. Parikh, *Three-dimensional grid generation by advancing front method*, Int. J. Num. Meth. Fluids, **8**, 1135-1149 (1988).

[7] J.M. Escobar y R. Montenegro, *Several aspects of three-dimensional Delaunay triangulation*, Advances in Engineering Software, **27**, 1/2, 27-39 (1996).

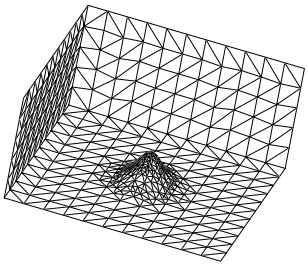
[8] L. Ferragut, R. Montenegro y A. Plaza, *Efficient refinement/derefinement algorithm of nested meshes to solve evolution problems*, Comm. Num. Meth. Eng., **10**, 403-412 (1994).

[9] P.L..George, F. Hecht and E. Saltel, *Automatic mesh generation with specified boundary*, Comp. Meth in Appl. Mech and Eng., **92**, 269-288, (1991).

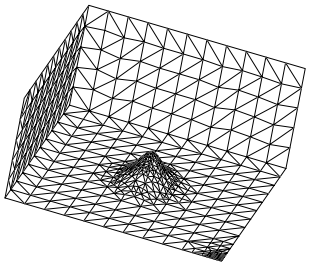
[10] R. Montenegro, A. Plaza, L. Ferragut e I. Asensio, *Application of a nonlinear evolution model to fire propagation*, Nonlinear Analysis, Th., Meth. App., **30**, 5, 2873-2882 (1997).

[11] A. Plaza y G.F. Carey, *Local refinement of simplicial grids based on the skeleton*, Appl. Numer. Math., **32**, 195-218 (2000).

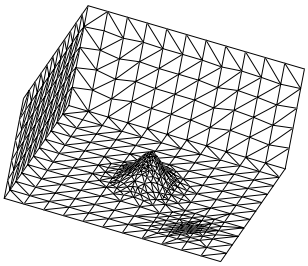
[12] M.C. Rivara, *A grid generator based on 4-triangles conforming. Mesh-refinement algorithms*, Int. J. Num. Meth. Eng., **24**, 1343-1354 (1987).



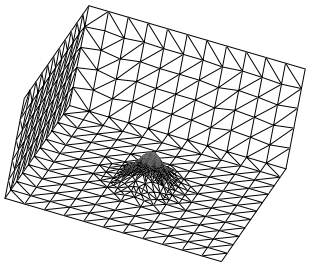
(a) Malla τ_0



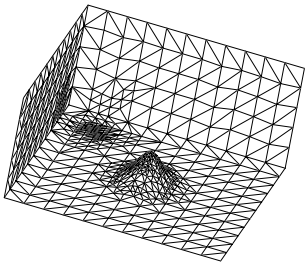
(b) Malla τ_1



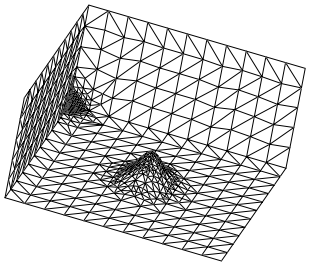
(b) Malla τ_2



(b) Malla τ_3



(b) Malla τ_4



(b) Malla τ_5

Figura 6. Resultado del refinamiento/desrefinamiento de la malla τ_0

[13] M.C. Rivara y C. Levin, *A 3-d refinement algorithm suitable for adaptive multigrid techniques*, J. Comm. Appl. Numer. Meth., **8**, 281-290 (1992).

[14] G. Winter, G. Montero, L. Ferragut y R. Montenegro, *Adaptive strategies using standard and mixed finite elements for wind field adjustment*, Solar Energy, **54**, 1, 49-56 (1995).

[15] D.N. Arnold, A. Mukherjee y L. Pouly, *Locally adapted tetrahedral meshes using bisection*, SIAM J. Sci. Comput., **22**, 2, 431-448 (2000).